UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

Leveraging Large Language Models to Support Web Accessibility Evaluation

João Miguel Marques da Costa

Mestrado em Engenharia Informática

Dissertação orientada por:
Prof. Doutor Carlos Alberto Pacheco dos Anjos Duarte
Profª. Doutora Letícia Seixas Pereira

2025

I dedicate this work to all the family and friends that helped shape my path in becoming who I am today.

.

Acknowledgments

First and foremost, i'd like to express my gratitude to my supervisors Prof Carlos Duarte and Prof Letícia Pereira, that throughout this last year have helped the most with the devolopment of this work, providing valuable guidance and insights while also always being kind and supportive with me. I'd like to thank my family, specially my parents and sister, that since I remember have always supported me regardless, and a special thanks to both my grandfathers that were always supportive and asking how my work was going. I also would like to thank my friends and colleagues that both helped me and distracted me, but most of all made this journey much more joyful. Lastly, I'd like to thank my cat Silvestre for keeping me company during all the long hours spent in front of the desk.

Resumo

A acessibilidade na Web continua a ser um dos maiores desafios no desenvolvimento de interfaces digitais, assumindo uma relevância crescente num contexto em que serviços essenciais como a saúde, educação e plataformas governamentais, dependem cada vez mais de plataformas online. Apesar de existirem diretrizes internacionais consolidadas, como as Web Content Accessibility Guidelines (WCAG), e de as ferramentas de avaliação automática de acessibilidade permitirem identificar um conjunto significativo de problemas, a realidade mostra que muitas barreiras permanecem, sobretudo em contextos onde a interpretação depende do significado, da clareza das mensagens ou da interação dinâmica com o utilizador. Ferramentas automáticas são eficazes em verificações objetivas, mas não têm a capacidade de avaliar situações que exigem interação e raciocínio contextual. Nestes casos, as avaliações manuais assumem um papel indispensável, pois permitem que especialistas analisem aspetos mais subjetivos, como a clareza das instruções ou a adequação das mensagens de erro. No entanto, estas avaliações exigem tempo, conhecimento especializado e não são facilmente escaláveis, o que cria um bloqueio na adoção generalizada de boas práticas de acessibilidade.

Neste cenário, os Modelos de Linguagem de Grande Escala (LLMs) surgem como uma oportunidade promissora. Estes modelos, treinados sobre vastos conjuntos de dados textuais, apresentam capacidades notáveis de interpretação semântica e raciocínio contextual, permitindo-lhes avaliar conteúdos e interações de forma mais próxima ao julgamento humano. O seu potencial de análise contextual traz possibilidades para complementar métodos existentes e reduzir a dependência em avaliações manuais, contribuindo para processos mais rápidos e escaláveis.

Esta dissertação procura investigar como os LLMs podem apoiar a avaliação da acessibilidade em formulários Web. Formulários são pontos de interação que mediam o acesso a serviços fundamentais, desde registos e autenticações até operações financeiras ou submissão de pedidos. Por envolverem múltiplas etapas sequenciais tornam-se particularmente vulneráveis a barreiras de acessibilidade. O trabalho aqui apresentado propõe uma solução que integra interação real com formulários Web, observando os comportamentos do DOM após submissões e captando *feedback* dinâmico. Este aspeto distingue a investigação da maioria dos estudos anteriores, que ou se concentram em verificações puramente estáticas, ou abordam critérios de acessibilidade de forma isolada.

Para concretizar esta abordagem, foi desenvolvida uma ferramenta baseada em LLMs que combina interação automática com formulários, observação de mutações no DOM (Modelo de

Objeto de Documento) e integração de técnicas de geração aumentada de recuperação (RAG). O desenvolvimento recorreu ao DSPy como plataforma de desenvolvimento, permitindo estabilizar as interações com o modelo e introduzir consistência nos resultados. Foram ainda construídos conjuntos de treino específicos para diferentes problemas de acessibilidade, de modo a orientar, otimizar e estabilizar as respostas dos modelos.

A investigação incidiu sobre um conjunto específico de problemas de acessibilidade em formulários, identificados a partir do mapeamento das WCAG 2.2. Entre os casos analisados encontram-se: identificação do uso correto do atributo *required*, falha na identificação de erros ou identificação inválida, mensagens de erro ambíguas ou genéricas e estados de erro indicados apenas por cor. Estes problemas foram selecionados por representarem barreiras frequentes e de elevado impacto, que não são cobertas adequadamente pelas ferramentas de avaliação automática de acessibilidade.

Para validar a abordagem, foram recolhidos e analisados formulários reais de páginas Web de elevada relevância, incluindo páginas nacionais e internacionais com grande tráfego. Cada página foi sujeita a uma avaliação manual, realizada segundo critérios de referência, de modo a criar uma referência para comparação dos resultados. Os resultados obtidos pela ferramenta foram então contrastados com as avaliações humanas e com os relatórios produzidos pelas ferramentas de avaliação automática de acessibilidade já existentes.

Os resultados demonstraram que os LLMs conseguem fornecer julgamentos significativos em contextos não cobertos pelas ferramentas de avaliação automática de acessibilidade. Em particular, revelaram maior eficácia em detetar ausência de *feedback* após submissão e ausência de indicação correta de que um campo é obrigatório. Também apresentaram resultados promissores na distinção entre mensagens de erro vagas e precisas e na avaliação da clareza das instruções fornecidas ao utilizador. A comparação entre diferentes modelos evidenciou variações de desempenho, sublinhando a importância da escolha do modelo e da qualidade dos dados de suporte. Modelos mais recentes e com maior capacidade demonstraram maior consistência, enquanto versões mais limitadas apresentaram variabilidade acentuada, sobretudo em cenários de maior ambiguidade.

Por outro lado, ficaram evidentes algumas limitações. A escala reduzida do conjunto de treino limitou a capacidade de generalização. A variabilidade nas respostas, dependente da formulação das instruções passadas aos modelos e do contexto fornecido, mostrou que a consistência ainda é um desafio. Foram igualmente observadas fragilidades na extração de informações a partir do DOM, especialmente em páginas com elevada complexidade estrutural, o que pode comprometer a fiabilidade em contextos mais dinâmicos. Estas limitações reforçam que os LLMs devem ser encarados como uma camada complementar aos métodos existentes, e não como substitutos diretos.

Do ponto de vista comparativo, o estudo revelou que os LLMs oferecem vantagens claras na avaliação de critérios que dependem de interpretação contextual, cobrindo lacunas das ferramentas automáticas.Contudo, não oferecem a mesma cobertura sistemática e objetiva destas ferramentas. A integração das duas abordagens surge, portanto, como a estratégia mais eficaz para uma maior expansão da avaliação automática: recorrer às ferramentas automáticas para deteção consistente

de violações bem definidas e utilizar os LLMs como camada adicional capaz de lidar com ambiguidade, linguagem e contexto. Esta complementaridade tem potencial para reduzir o esforço humano nas auditorias manuais e, ao mesmo tempo, preservar a profundidade analítica necessária.

Para além disso, a investigação apresenta orientações práticas sobre como estruturar ferramentas de avaliação baseadas em LLMs. Mostra que a utilização de DSPy e de técnicas de RAG é eficaz para estabilizar resultados, e que a formulação de conjuntos de treino específicos para cada problema melhora a precisão da análise. Assim, o trabalho não só demonstra viabilidade técnica, mas também oferece contributos metodológicos reutilizáveis para investigação futura.

Do ponto de vista ético e social, a utilização de LLMs em acessibilidade também levanta desafios. Embora demonstrem oportunidades para reduzir barreiras, é necessário garantir que não introduzem problemas adicionais, falsos positivos ou falsos negativos que possam comprometer auditorias manuais ou consequentemente processos de correção de problemas inexistentes. A confiança nestas ferramentas deve ser construída de forma gradual, sempre articulada com práticas de revisão humana.

Em termos de contributos, este trabalho distingue-se de estudos anteriores em três dimensões: concentra-se em problemas de acessibilidade web, uma área essencial para diversas tarefas relativas a *websites* e aplicações porém sub-explorada no contexto de automatização de avaliação de acessibilidade; introduz uma ferramenta que recorre à interação direta com formulários reais, ao invés de depender de análise estática; e utiliza conjuntos de exemplos cuidadosamente preparados para alinhar as capacidades dos LLMs com problemas concretos de acessibilidade.

Para investigação futura, destaca-se a necessidade de expandir a cobertura a mais problemas de acessibilidade e a um conjunto maior de casos de teste, incluindo outros desafios associados a formulários, para os quais a ferramenta aqui desenvolvida pode ser adaptada. É igualmente importante aprofundar a integração dos LLMs com ferramentas de avaliação de uso generalizado e desenvolver métodos que assegurem maior consistência nos resultados. Para além disso, a abordagem poderá ser estendida a outros contextos, como conteúdos multimédia e interfaces altamente dinâmicas, onde as barreiras de acessibilidade continuam a ser significativas. O avanço nestas direções poderá contribuir para transformar de forma substancial a forma como a acessibilidade é avaliada em larga escala, tornando os processos mais robustos, abrangentes e eficazes.

Em síntese, este trabalho demonstra que os Modelos de Linguagem de Grande Escala oferecem um contributo relevante em identificar problemas de acessibilidade em formulários, permitindo julgamentos contextuais que até agora dependiam sobretudo da análise manual. Apesar dos desafios identificados, a investigação confirma o seu potencial como um avanço promissor no caminho para uma Web mais inclusiva e equitativa, em particular em áreas críticas de interação como os formulários.

Palavras-chave: Acessibilidade na Web, Grandes Modelos de Linguagem, Formulários HTML, Avaliação Automática de Acessibilidade, Interação Pessoa-Máquina

Abstract

Web accessibility remains a critical challenge, as automated tools identify only a subset of issues, and conformance checking still requires extensive manual verification. Large Language Models (LLMs), with their ability to process natural language and reason over structured inputs, offer an opportunity to address evaluation tasks that involve contextual or interaction-dependent understanding. This thesis investigates how LLMs can support the assessment of form-related accessibility criteria that are not reliably captured by existing rule-based tools. To this end, an evaluation pipeline was developed using DSPy, combining automated form interaction, DOM mutation observation, retrieval grounding, and fine-tuning trainsets. A small annotated dataset was constructed to guide and stabilize model outputs. In addition, experiments compared different LLMs against manual benchmarks and traditional accessibility checkers. The pipeline was applied to four representative criteria that are traditionally not addressed by automated evaluation tools, including correct use of required attribute, failure to identify errors or invalid identification, ambiguous or generic error messages and error states indicated only by color. Results indicate that LLMs can provide meaningful and actionable judgments in these contexts, particularly when supplied with structured inputs and retrieval-based grounding. They showed strengths in detecting missing feedback and distinguishing vague from precise error messages, while also revealing limitations related to dataset scale, model variability, and extraction fragility. Rather than replacing rule-based engines, the findings suggest that LLMs are best positioned as a complementary layer, extending coverage to nuanced cases and reducing the burden of manual auditing.

Keywords: Web Accessibility, Large Language Models, HTML Forms, Automated Accessibility Evaluation, Human-Computer Interaction

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Web has become an essential part of daily life, shaping how people communicate, learn, and even access essential services. Ensuring accessibility for all users, including those with disabilities, is both a design priority and a legal and ethical requirement [1]. The Web Content Accessibility Guidelines (WCAG) [2], developed by the World Wide Web Consortium (W3C), provide a widely adopted framework for creating inclusive digital environments. Although WCAG provides a comprehensive and widely recognized standard, many websites and applications continue to fall short of compliance [3].

1.1 Motivation

Web accessibility remains a persistent challenge despite the proven guidance of WCAG and the availability of automated evaluation tools, which are systems that automatically scan webpages in order to detect accessibility issues. These automated tools play an important role in detecting certain violations, but they only cover part of the accessibility criteria and are limited in addressing issues that require contextual interpretation or semantic understanding [3]. Human-based evaluations, on the other hand, involve detailed inspection by accessibility experts and are resource-intensive, requiring both time and specialized knowledge [4]. This imbalance creates a bottleneck for achieving and maintaining accessibility at scale, particularly on large, dynamic, or content-rich websites. In practice, the two approaches complement each other as automated tools make it possible to scale the detection of common issues, while expert reviews provide the depth needed to assess more complex cases. Together, they highlight both the progress made in automation and the remaining challenge of achieving and maintaining accessibility at scale, particularly on large, dynamic, or content-rich websites.

Within the broader context of web accessibility, web forms are a particularly relevant area for exploration. Forms are common interaction points for tasks such as registration, login, and checkout, and they are especially prone to accessibility issues that are harder to detect automatically because they involve contextual elements, such as unclear instructions or ambiguous error messages [5]. As forms often mediate access to essential services, their accessibility has a direct impact on inclusivity.

Large Language Models (LLMs) present a timely opportunity to enhance the evaluation of these issues, by supporting the detection of problems that require contextual interpretation. LLMs demonstrate strong capabilities in interpreting both natural language and structured data such as HTML [6], and their contextual reasoning potential has already been explored for accessibility-related tasks [7]. Leveraging these capabilities could extend evaluation beyond rigid rule-based systems, reducing exclusive reliance on manual testing and moving toward more scalable, context-aware approaches.

Building on this potential, this thesis investigates how LLMs can be integrated into an accessibility evaluation pipeline. It focuses on form-related accessibility issues as a representative and high-impact case study, aiming to understand how LLMs might complement or extend existing evaluation methods.

1.2 Objectives

The main objective of this thesis is to investigate the potential of LLMs to support web accessibility evaluation, with a focus on form-related accessibility issues that require contextual understanding. Rather than developing a new evaluation tool in itself, the aim is to explore how LLMs can complement existing methods by addressing limitations that automated tools alone cannot overcome.

The specific goals of this research are:

- Understand how LLMs can be applied to identify form-related accessibility issues that depend on contextual interpretation.
- Design and implement a pipeline to test and validate this potential, using real-world web forms.
- Compare the outcomes of LLM-based evaluations with those obtained through traditional automated tools and manual evaluations.
- Identify the limitations, reliability concerns, and practical implications of integrating LLMs as complementary components in accessibility evaluation.

To support these goals, a prototype pipeline was developed as a practical framework to evaluate how LLMs address accessibility issues. This pipeline serves as a basis for assessing both the strengths and limitations of LLMs, and for identifying the conditions under which they may provide added value in practice.

These goals are explored through the following research questions:

RQ1 To what extent can LLMs accurately identify and evaluate context-dependent form-related accessibility issues?

RQ2 How valid and reliable is the rationale produced by the LLMs for evaluating form accessibility?

RQ3 What limitations emerge when using LLMs for accessibility evaluation, and what are the main sources of errors observed in their outputs?

RQ4 How do LLM-based evaluations compare with existing automated tools, and in what ways can they complement these tools when assessing form accessibility?

1.3 Contributions

This work makes the following contributions to the field of web accessibility evaluation using LLMs:

- **Creation of a dataset for Retrieval-Augmented Generation (RAG):** A comprehensive dataset was developed mapping WCAG A and AA success criteria related to web forms. This dataset supports retrieval-augmented generation by providing relevant contextual information that LLMs can reference during the evaluation process. By stabilizing the input data, this dataset improves the reliability of LLM-based assessments of web form accessibility.
- **Development of fine-tuning trainsets for accessibility issues:** Four different fine-tuning trainsets were created, each focused on a specific accessibility issue. These trainsets provided LLMs with diverse input-output examples, helping the model learn to identify and generate appropriate responses for accessibility issues in web forms. This targeted fine-tuning improves the model's accuracy in evaluating forms and detecting accessibility violations.
- **Evaluation on real-world web forms with manual benchmarking:** The study evaluated 10 pages from globally and locally popular websites, focusing on form-related accessibility issues such as missing programmatic cues for required fields, invalid or missing error identification, ambiguous or generic error messages, and error or required field indicators conveyed only by color. Each page was manually audited to create a gold standard benchmark, used to validate the accuracy of LLM-based evaluations. This benchmark ensured alignment with accessibility standards and enabled reliable comparison of the model's performance.
- **Form interaction automation:** Custom scripts were developed to interact with dynamic form elements on webpages, capturing relevant data to feed into the LLM evaluation pipeline.
- **Analysis of LLM strengths and limitations:** This study offers a comprehensive analysis of the types of accessibility issues where LLMs perform well and where they fall short, also highlighting cases where traditional automated or manual testing remains essential. These findings provide actionable insights for improving the future of LLM-based accessibility tools and offer a balanced view of their current capabilities and limitations in real-world applications.

The materials developed in this thesis, including the evaluation pipeline code, experimental results, the form-related WCAG dataset, the four fine-tuning trainsets, and the automation scripts for interaction and data extraction, are available at `https://github.com/Costa13J/LLM-accessibility-evaluator`.

1.4 Thesis Overview

The structure of this thesis is as follows:

Chapter 2 presents the background and related work, covering foundational concepts in web accessibility, including WCAG guidelines, common accessibility issues in forms, and existing automated and manual evaluation techniques. It further introduces the use of LLMs in accessibility contexts, highlighting challenges such as hallucinations, prompt engineering, and the retrieval-augmented generation approach adopted in this work.

Chapter 3 describes the methodology and research design. It details the mapping and selection of form-related WCAG success criteria, the identification of common accessibility issues, and the selection of webpages for evaluation. This chapter also explains the design of the evaluation pipeline, including automated webpage interaction, dynamic feedback capture, LLM-guided interactions, dataset and trainset construction, and the prompting and retrieval strategies employed, as well as refinements made across pipeline iterations.

Chapter 4 presents the evaluation and results. It outlines the evaluation setup and manual benchmarking process, compares different LLM models, and provides quantitative results both at field and page levels. The chapter also discusses per-criterion accuracy across models, analyzes the reasoning behind LLM evaluations, compares their performance against traditional automated tools, and assesses the effectiveness of retrieval and prompting strategies, while addressing validity concerns and limitations.

Chapter 5 discusses the findings, reflecting on the research questions, summarizing key contributions, and sharing insights learned. It further explores the scalability and generalization potential of the approach, its integration with existing automated tools, and its practical limitations.

Chapter 6 concludes the thesis and outlines directions for future work aimed at extending and improving the use of LLMs in web accessibility evaluation.

Chapter 2

Background and Related Work

This chapter provides the foundation for the work carried out in the thesis. It introduces WCAG and the principles behind accessibility, followed by an overview of how automated and manual evaluation methods address compliance. Then, it reviews how accessibility is usually tested, contrasting the reach of automated tools with the nuances of manual evaluation. The chapter next turns to Large Language Models, describing how they work, the challenges of using them in evaluation tasks, and the techniques developed to improve their reliability. It concludes with an overview of recent research in this area, outlining current approaches, their limitations, and the gaps that motivate the contributions of this thesis.

2.1 Web Accessibility

Web accessibility is defined and assessed through established standards and evaluation practices. This section introduces the Web Content Accessibility Guidelines (WCAG) and Accessibility Conformance Testing (ACT) Rules, key concepts and standards in this context, and also automated and manual methods that shape how accessibility is tested in practice.

2.1.1 Web Accessibility and WCAG

The Web Content Accessibility Guidelines (WCAG), maintained by the World Wide Web Consortium (W3C) [1], provide a structured framework to assess and improve accessibility on the Web. The guidelines are built around four core principles: Perceivable, Operable, Understandable, and Robust (POUR), which are made actionable through a set of Success Criteria (SC).

These criteria are organized into three conformance levels: Level A, which sets the minimum requirements to avoid excluding users; Level AA, which adds requirements to ensure accessibility for diverse users across different platforms; and Level AAA, which defines requirements aimed at providing an optimal user experience for all. They cover a wide range of requirements, such as ensuring that essential information is not conveyed by color alone (SC 1.4.1), providing text alternatives for non-text content (SC 1.1.1), and offering users clear and specific error messages when input validation fails (SC 3.3.1).

While many criteria can be evaluated with automated tools, others require nuanced interpretation. For example, determining whether instructions are sufficiently clear, whether error messages are specific to the issue, or whether interactive elements have meaningful labels often demands contextual judgment. These limitations create a gap in existing accessibility evaluation pipelines, a gap that LLMs may help address.

2.1.2 ACT Rules

Accessibility Conformance Testing (ACT) rules[1] are a set of test rules developed by the W3C to standardize the evaluation of web content for compliance with WCAG and WAI-ARIA (Web Accessibility Initiative - Accessible Rich Internet Applications)[2]. WAI-ARIA defines attributes that enhance web content accessibility, particularly for users relying on assistive technologies, and is frequently referenced in ACT rules to promote alignment with WCAG and other accessibility standards. Each ACT rule outlines a specific procedure for determining whether a piece of content meets a particular aspect of a WCAG success criterion. For instance, an ACT rule might specify how to detect missing alternative text for images or identify instances where form elements lack accessible labels. These rules, despite not being mandatory for determining conformance to WCAG, are adopted by automated evaluation tools, such as QualWeb, to improve the consistency and reliability of accessibility testing. By integrating ACT rules, these tools can better align their automated checks with WCAG standards. However, not all ACT rules can yet be integrated into automated testing tools, as some of these rules still need to be tested manually.

2.1.3 Automated Accessibility Evaluation Tools

Automated tools such as QualWeb[3] and WAVE[4] are widely used to evaluate web accessibility against WCAG standards. These tools primarily operate on rule-based algorithms, scanning HTML, CSS, and DOM structures to detect violations such as missing text alternatives for images (SC 1.1.1) or insufficient color contrast between foreground and background elements (SC 1.4.3) [4].

Despite these benefits, automated evaluation tools also have significant limitations, many of which stem from their reliance on static analysis:

- **False Positives and False Negatives:** A false positive occurs when a tool flags an issue that is not actually a violation, while a false negative occurs when a genuine problem is missed [8]. Both undermine trust in automated results, can waste developer time, divert resources toward resolving non-existent problems, and even introduce new issues during remediation, while false negatives allow real barriers to persist [4, 7, 9, 10]. In practice, false positives are often more damaging, as they reduce confidence in automated evaluations and

[1] https://www.w3.org/WAI/standards-guidelines/act/rules/about/
[2] https://www.w3.org/WAI/standards-guidelines/aria/
[3] https://qualweb.di.fc.ul.pt/evaluator/
[4] https://wave.webaim.org/

risk obscuring genuine accessibility barriers that require attention. False negatives, while also undesirable, can more often be identified through complementary manual testing.

- **Need for Human Input:** Many WCAG success criteria require contextual interpretation beyond what rule-based tools can provide. Examples include verifying whether a link text clearly conveys its purpose (SC 2.4.4), ensuring that error messages are specific and actionable (SC 3.3.1), or confirming that instructions and labels are meaningful and unambiguous (SC 3.3.2). Automated tools excel at detecting objective, syntax-based violations but struggle with these more subjective assessments. Recent research [4] has shown that LLMs such as GPT-4 can outperform traditional tools on nuanced criteria by leveraging natural language understanding. For example, an LLM can identify when the declared language of a page (SC 3.1.1) does not match the actual textual content [9], a task that requires both semantic analysis and contextual reasoning. This ability suggests that LLMs can serve as a bridge between the broad coverage of automated tools and the nuanced interpretation typically provided by human evaluators.

These challenges have prompted interest in approaches that explore how AI models, particularly Large Language Models, can complement existing accessibility evaluation methods; these approaches are further discussed in Section 2.3.

2.1.4 Manual Accessibility Evaluations

Manual evaluations complement automated tools by allowing human judgment to assess issues that automated checkers often miss. For example, determining whether the heading hierarchy communicates a clear document structure (WCAG SC 2.4.6) typically requires contextual interpretation beyond what automation can provide [11].

However, manual testing is inherently time-consuming, costly, and difficult to scale across large or frequently changing websites [3, 9, 12–14]. For this reason, some accessibility workflows adopt a mixed strategy, leveraging automated tools for broad coverage and using manual evaluations to validate context-sensitive or ambiguous cases [4, 15].

This combination ensures higher overall coverage but still leaves important gaps in scalability and consistency. Large Language Models offer a potential middle ground, capable of applying contextual reasoning at scale while reducing the manual effort required, making them a promising direction for advancing accessibility evaluation [4].

2.1.5 Accessibility of Web Forms

Forms are one of the most critical interaction points on the web, providing access to essential services such as registration, authentication, e-commerce, and communication. Unlike static content, they require users to complete tasks through multiple dependent steps, including perceiving labels, understanding instructions, providing input, and responding to validation feedback. Failures at any

of these stages can block task completion, making forms a frequent source of accessibility barriers. Prior work has shown that forms are especially problematic for blind and visually impaired users, with recurring issues such as missing or ambiguous labels, inaccessible error handling, and poor focus management, which collectively limit independent use of online services [16, 17].

2.2 LLMs for Accessibility

Large Language Models (LLMs) are advanced neural network-based architectures trained on extensive datasets, allowing them to analyze and generate logical and contextually relevant text based on input prompts [18]. Models, like OpenAI's GPT-4o and Anthropic's Claude, operate primarily through autoregressive processes, which means that they generate outputs by predicting each subsequent word or token in a sequence based on prior context [19]. This setup enables LLMs to capture complex language patterns, producing outputs that closely resemble human-like comprehension and expression.

LLMs have demonstrated their adaptability in various fields. They exhibit advanced Natural Language Processing (NLP) capabilities, including the ability to interpret complex linguistic structures and generate context-aware responses [20]. These models can process diverse content types and linguistic forms, facilitating nuanced understanding and generation tasks that traditional rule-based systems struggle with.

2.2.1 Reliability Challenges of LLMs

LLMs exhibit strong natural language understanding and reasoning capabilities, making them attractive for tasks such as interpreting complex accessibility guidelines. However, these same models are also prone to hallucinations, outputs presented as factual but that are either incorrect, unsubstantiated, or entirely fabricated [10, 13, 21]. In the context of accessibility evaluation, such hallucinations may include misidentifying the compliance status of a feature, fabricating code elements that do not exist in the page source, or misinterpreting user interface cues.

In the context of LLM-based accessibility evaluation, hallucinations can reduce accuracy either by creating false positives or missing genuine issues through false negatives. As with automated tools, both outcomes are problematic, but in accessibility issue identification false positives are often the bigger concern, since they can undermine trust and obscure actual barriers.

The reliability of LLM-based evaluations is further complicated by the high sensitivity of outputs to prompt phrasing, structure, and context [4, 7, 10, 15]. Small changes in wording or task setup can significantly affect the model's interpretation of accessibility requirements, leading to inconsistent results in real-world testing scenarios. This variability means that prompt engineering and context management become critical factors, but even well-crafted prompts cannot fully eliminate the risk of hallucinations or inconsistent reasoning [6, 13].

2.2.2 Prompting and Prompt Engineering

This sensitivity to phrasing and contextual framing is directly related to how LLMs are prompted [7, 10, 20, 22]. In this context, a prompt defines both the instructions and the situational context given to the model, acting as the primary control mechanism for shaping its behavior. Different prompting approaches exist, including zero-shot prompting, where no examples are provided, and few-shot prompting, which provides a small set of examples to guide the model's reasoning [18, 21, 23]. Even subtle changes in wording, ordering, or emphasis can produce markedly different outputs, particularly in tasks such as accessibility evaluation, where small interpretive shifts may change compliance judgments [13].

The practice of prompt engineering refers to the deliberate design and refinement of prompts to improve task alignment and reduce uncertainty [7, 10, 13]. Common strategies involve providing worked examples, breaking complex tasks into smaller instructions, or encouraging step-by-step reasoning [21]. Although approaches like this can increase reliability, they do not fully address the persistent challenge often referred to as prompt-induced variability, the tendency of model outputs to change when the prompt's wording, structure, or surrounding context is altered, even if the underlying task remains identical [15, 24]. In the context of accessibility testing, this variability can lead to inconsistent results between otherwise equivalent evaluations, making stable and well-documented prompt structures an important foundation for LLM-based assessments.

2.2.3 DSPy and Retrieval-Augmented Generation

These challenges of prompt sensitivity and variability have motivated the development of frameworks that structure and manage LLM interactions more systematically. One such framework is DSPy [25], a modular environment for programmatic prompting that enables the creation of reproducible and adaptive LLM pipelines. Rather than relying on manually crafted, static prompts, DSPy allows prompts to be expressed as parametrized templates, where inputs, outputs, and evaluation criteria can be programmatically controlled. This approach not only improves consistency, but also facilitates the integration of additional reasoning or verification steps within the pipeline [26].

An important feature of DSPy is its ability to support Retrieval-Augmented Generation (RAG), a technique in which an LLM is coupled with an external retrieval model [21]. In this setup, a retrieval model searches a knowledge base or dataset to identify relevant reference material that can be incorporated into the prompt before generation. By adding the retrieved information directly into the prompt, the model can base its reasoning on relevant and domain-specific examples. This grounding helps it stay aligned with factual reference material, which in turn lowers the risk of hallucinations and improves the reliability of its outputs.

The combination of structured prompting and evidence grounding through RAG has been shown to improve accuracy and keep model outputs aligned with the task domain [21, 27]. In these settings, giving the LLM consistent instructions, relevant context, and concrete examples can make its outputs more stable, easier to understand, and easier to check. Using trusted reference

material as part of the prompt also helps reduce common problems such as hallucinations and false positives [21].

2.3 State of the Art

Recent research on the use of Large Language Models in accessibility has explored their role in detecting accessibility issues, remediating inaccessible content, and, in some cases, performing both tasks in a unified workflow. Work spans both web and mobile contexts, with approaches ranging from source-code inspection to reasoning over rendered content or analyzing user interaction data.

This section first covers issue identification, then reviews remediation techniques, hybrid approaches, and concludes with identified research gaps.

2.3.1 Issue Detection in Mobile/Web Accessibility

Several studies have investigated the use of LLMs as a means to enhance traditional accessibility evaluation. In the web domain, López and Pereira [4] evaluated three WCAG success criteria, 1.1.1 Non-text Content, 2.4.4 Link Purpose (In Context), and 3.1.2 Language of Parts, using GPT-4, Anthropic Claude, and Google Bard with dedicated prompts, achieving 87% accuracy compared to 0–59% for conventional tools. Sergelius [7] applied prompt refinement techniques to ACT test cases for 2.4.2 Page Titled, obtaining high accuracy in passing cases but lower performance for ambiguous or failing ones, highlighting the importance of iterative prompt design. Fuglerud et al. [9] extended detection to multimodal analysis using Contrastive Language-Image Pre-Training (CLIP) for image–text similarity (1.1.1), Optical Character Recognition (OCR) for identifying images of text (1.4.5), and language detection for 3.1.1 and 3.1.2, reporting better performance than static tools but persistent difficulties with edge cases. Duarte et al. [11] investigated heading-related accessibility issues in HTML documents, such as improper hierarchy, excessive use, and confusing labels. The study evaluated the performance of LLMs, including GPT-4o and Llama 3.1, in detecting these issues using various prompt strategies, session configurations, and context. The findings indicated that with properly crafted prompts, LLMs could identify subtle heading problems often missed by traditional tools, highlighting their potential for enhancing accessibility evaluations. This research contributes to the field by emphasizing the role of prompt optimization for accurate detection of heading-related accessibility barriers.

A related study by He et al. [12] advanced this line of work by developing GenA11y, an automated LLM-based checker covering 37 WCAG success criteria. The authors combined element extraction with criterion-specific prompting to mitigate token overhead and reduce hallucinations when analyzing real-world websites. In evaluations across two benchmark datasets and 36 live sites, GenA11y achieved high effectiveness (94.5% precision, 87.6% recall) and detected, on average, eight more violation types per page than existing tools. This work provides the most comprehensive investigation to date of LLM-based detection in web accessibility, complementing prior studies focused on smaller subsets of criteria or multimodal cases.

Some work shifts focus from static content to interaction-driven evaluation. Huq et al. [10] introduced Reca11, which processes transcripts from GPT-4 user testing sessions to generate structured reports of accessibility barriers. This approach enables the detection of issues often overlooked by automated scans, such as navigational inconsistencies and unclear feedback.

In the mobile domain, Taeb et al. [28] developed AXNav, a multi-agent system that combines pixel-level UI models with LLMs to interpret natural language test instructions, execute navigation tasks, and generate annotated replays of the interaction. Liu et al. [20] proposed GPTDroid, where an LLM with a functionality-aware memory mechanism explores mobile interfaces efficiently, avoiding redundant actions while maintaining test coverage.

2.3.2 Remediation and Issue Correction

Other studies have focused on automatically correcting identified accessibility issues by prompting LLMs to modify HTML or application code. Othman et al. [29] combined WAVE detection with ChatGPT remediation for two websites that failed WCAG 2.1. The model, trained on WCAG guidelines, was tasked with generating code fixes for issues such as missing or empty form labels and unclear link descriptions. Of the 39 issues flagged by WAVE, 37 were successfully remediated, although subjective checks such as color contrast remained problematic.

Delnevo et al. [30] applied ChatGPT to HTML snippets containing forms, tables, and images, prompting the model with "Is the following HTML code accessible?" and requesting both evaluation and correction. The LLM successfully addressed basic issues such as missing alt text, incorrect semantic elements, and unlabelled form fields, but struggled with complex layouts, dynamically generated content, and nuanced image descriptions. The authors concluded that LLM remediation works best when paired with human review.

In mobile contexts, Liu et al. [27] developed HintDroid, which uses GPT-3.5 Turbo with a feedback-based inspection loop to generate descriptive hint text for input fields, addressing the high percentage of mobile inputs lacking accessible hints. This in-context learning approach iteratively refines outputs using GUI metadata, significantly outperforming rule-based and machine learning baselines. Mehralian et al. [31] introduced FixAlly, a multi-agent system that detects, localizes, and fixes accessibility issues in mobile apps using a "plan–localize–fix" workflow. FixAlly produced targeted code-level patches for a variety of issues, achieving a 77% effective fix rate without breaking app functionality.

2.3.3 Accessible Code Generation

Beyond remediation of existing issues, several studies have examined the ability of LLMs to directly generate accessible code. Suh et al. [23] conducted a study comparing human developers with LLMs in producing accessible HTML and ARIA-enhanced components. Their findings show that while LLMs can reliably reproduce accessibility patterns for common structures such as buttons, forms, and navigation menus, they often generate incorrect or redundant ARIA attributes. Human-written code still outperformed LLMs in nuanced design cases, but the study highlights

significant potential for LLM-assisted workflows when coupled with validation tools.

Pedemonte et al. [32] proposed an LLM-based tool to improve the accessibility of STEM images, a domain often neglected in accessibility research. Their system prompts LLMs to generate textual alternatives for mathematical and scientific figures, producing preliminary but promising results. While the generated descriptions were often less detailed than those written by experts, the approach shows promise for scaling accessible content generation in specialized domains, as long as domain expertise and human validation remain part of the process.

2.3.4 Hybrid Techniques (Detection and Correction)

Some research combines detection and remediation in a single automated workflow. Huang et al. [13] developed ACCESS, integrating Playwright-based interaction testing with ReAct prompting in GPT-3.5 Turbo 16k. The system detects DOM-level violations such as missing landmarks, duplicate content, and absent ARIA-required attributes, and generates direct code fixes. In tests, ACCESS reduced violation severity by 51% and achieved 100% correction rates for certain criteria, including landmark-no-duplicate-content, label, skip-links, and ARIA-required attributes. However, it struggled with more complex visual or layout-related barriers, such as contrast issues or responsive design flaws, which require deeper visual-semantic understanding.

2.3.5 Research Gaps

While LLM-based approaches have demonstrated promising results across identification, remediation, and even accessible code generation, several limitations persist:

- **Narrow coverage of WCAG success criteria:** Existing studies have explored only a limited subset of WCAG guidelines. Some works focus on individual criteria such as SC 2.4.2 Page Titled, SC 2.4.4 Link Purpose (In Context), or SC 3.1.1 Language of Page, showing that LLMs can outperform static tools in text-based or attribute-based checks [4, 13, 29]. GenA11y [12] expands this scope by systematically categorizing success criteria into static, dynamic, and non-testable groups, but its evaluation focused solely on the 37 criteria detectable through static analysis. Despite these contributions, the overall coverage of WCAG remains limited, leaving significant gaps in dynamic, interactive, and context-dependent areas.

- **Lack of integration with established evaluation tools:** Although some studies have coupled LLMs with auxiliary frameworks, such as Playwright for interaction testing or multi-agent setups [13, 31], no study has directly integrated them into widely used accessibility evaluation tools. This limits reproducibility and hinders adoption in standard practitioner workflows.

- **Prompt dependence and variability:** The effectiveness of LLMs remains highly sensitive to prompt phrasing and task setup. Approaches such as ReAct prompting [13] or in-context learning for mobile hint generation [27] help mitigate variability but still rely heavily on

handcrafted strategies. This dependence poses challenges for scalability and consistency, making it difficult to reproduce results across contexts.

- **Scalability constraints:** Existing work is generally evaluated on small-scale datasets or targeted case studies [4, 27, 29]. Applying LLM-based evaluation across diverse, real-world websites or large app ecosystems remains resource-intensive and unexplored.
- **Limits of accessible code generation:** Studies assessing LLMs' ability to generate accessible HTML, ARIA components, or alternative text for STEM images [29, 32] show potential but highlight recurring problems, such as redundant or incorrect ARIA attributes and insufficient coverage of complex or dynamic components. Human review and validation are still necessary to ensure correctness.
- **Fragmentation across domains:** Work in web and mobile accessibility often progresses in parallel, with limited cross-pollination of findings. For example, mobile-specific systems such as AXNav [4] or FixAlly [31] demonstrate techniques that could be valuable in the web domain, but these connections remain underdeveloped.

Together, these limitations suggest that while LLMs are promising for addressing context-sensitive and judgment-based criteria, their current use is still fragmented, narrow in scope, and difficult to scale. The contributions presented in this thesis build on these insights, advancing some of the gaps identified, particularly those related to contextual interpretation and the consistency of evaluations, while leaving others, such as broader WCAG coverage or tool integration, open for future work. In this sense, the work represents a partial but promising step toward more scalable and consistent accessibility evaluations.

Chapter 3

Methodology and Evaluation Pipeline Design

This chapter presents the methodological approach adopted in this research, describing how we employed LLMs to evaluate accessibility issues in web forms. The approach was structured around the requirements of WCAG, while also adapting to the challenges of testing real-world websites. Central to this approach is an evaluation pipeline that integrates automated interaction, data collection, and model-based reasoning, providing a structured way to translate guideline criteria into field-level assessments.

To guide this process, the chapter unfolds in four stages. It begins with a systematic mapping of WCAG 2.2 success criteria, focusing on which can be addressed through automation and which require contextual reasoning. Building on this, the next section identifies common accessibility issues specific to web forms and relates them back to WCAG, narrowing the scope to the barriers most relevant to our study. The third section turns from this conceptual mapping to practice, describing how representative live web pages were selected to serve as the evaluation base. Finally, the chapter introduces the design of the evaluation pipeline, detailing how the different elements of automation, interaction with real pages, and model-based reasoning were integrated into a single workflow.

3.1 WCAG Success Criteria Mapping and Selection

The first stage of the methodology involved systematically reviewing all WCAG 2.2 success criteria at levels A and AA. These two conformance levels were chosen because they represent the most widely adopted compliance targets in practice: Level A sets the minimum baseline for accessibility, while Level AA is the standard most often required by regulations, policies, and industry benchmarks. Level AAA, although valuable, is not usually employed in real-world contexts and was therefore excluded from this mapping.

To structure the review, table A.1 was constructed (available in Appendix), and each criterion was analyzed using the following categories:

- **Success Criterion (SC)**

- **Level:** A or AA.
- **Description:** summary of the requirement to comply with the SC.
- **Manual Evaluation:** how compliance is typically verified by a human evaluator.
- **LLM Potential:** whether an LLM could provide additional support or contextual reasoning.
- **Automation Feasibility:** categorized as *Yes*, *No*, or *Partially*, with justification.
- **Existing LLM Applications:** whether prior studies have proposed or demonstrated LLM use for the evaluation or correction of the criterion.

Our mapping then made it possible to identify three main groups of criteria:

1. **Criteria already well-covered by automated tools.** These typically involve objective checks that can be determined by analysing HTML and CSS structure, such as color contrast ratios (SC 1.4.3) or the presence of alternative text attributes (SC 1.1.1).
2. **Criteria potentially approachable through LLM-based reasoning.** These criteria require contextual interpretation, semantic understanding, or evaluation of subjective clarity. For example, SC 3.3.1 Error Identification requires not only detecting that an error message exists but also determining whether the message clearly identifies the issue and specifies what went wrong. A static automated tool can confirm the presence of an error string, but it cannot judge whether the message "Invalid input" is informative enough compared to "The password must be at least eight characters." Likewise, SC 3.3.3 Error Suggestion goes further by requiring that the message provide useful hints for correction. Such evaluations extend beyond static automated tools, but LLMs offer the potential to address these cases by analyzing textual content and providing reasoned judgments about its clarity and usefulness.
3. **Criteria unfeasible to automate, even with LLMs.** These criteria depend on factors that cannot be captured from source code or textual content alone. For example, SC 2.2.1 Timing Adjustable requires that users be able to turn off, adjust, or extend time limits, something that cannot be verified without interactive testing and consideration of user context. Likewise, criteria involving cognitive effort or user-specific abilities remain outside the scope of automation.

From this mapping, a set of criteria was retained for our case study, chosen for their particular relevance to form interactions. While some criteria, such as SC 1.3.1 Info and Relationships, already benefit from partial automated coverage, others remain difficult for traditional tools but align well with the reasoning capabilities of LLMs. To focus on barriers most specific to form interactions, and after filtering out criteria already well-covered in prior work, Level AAA criteria, and those identified as unfeasible to automate even with LLMs, we retained six success criteria for

Table 3.1: Selected WCAG 2.2 Success Criteria retained for our case study

Success Criterion	How could an LLM enhance the evaluation?	Can the test be automated?
1.3.5 Identify Input Purpose	Contextually analyze labels and placeholders to determine whether they align with the input's actual purpose.	Yes, although nuanced cases might be subjective.
2.4.6 Headings and Labels	Evaluate the clarity, relevance, and descriptive quality of headings and labels by comparing them to the content they introduce or describe.	Yes, although evaluating heading and label clarity and relevance in context remains subjective
2.5.3 Label in Name	Detect inconsistencies between visible text and accessible names.	Yes, although it can be subjective.
3.3.1 Error Identification	Validate the usability and correctness of the error message provided.	Partially; verifying contextual linkage to the relevant field can be challenging.
3.3.2 Labels or Instructions	Assess whether the label or instructions are clear, meaningful, or contextually appropriate. Automated tools can only identify presence, not clarity.	Yes, although it can be subjective.
3.3.3 Error Suggestion	Assess whether the provided guidance is sufficient and not overly vague (e.g., just "Error").	Yes/Partially; depends on contextual interpretation of helpfulness.

further analysis. Table 3.1 presents these criteria, emphasizing their relevance to form usage, their potential for LLM-based enhancement, and the extent to which their evaluation can be automated:

The outcome of this mapping was a focused set of six success criteria that reflect both persistent challenges in form accessibility and areas where LLMs may offer meaningful support. These criteria represent a balance between objective structural checks and more nuanced evaluations requiring contextual reasoning, such as assessing label clarity or the usefulness of error messages. By narrowing the scope in this way, the study aims to concentrate the analysis on areas where LLM-based methods are most likely to contribute. With this foundation in place, the next step was to map these criteria onto the concrete accessibility issues that directly arise in web forms.

3.2 Mapping of Form-specific Accessibility Issues

Building on the earlier identification of form-related barriers (section 3.1), this step aimed to capture the range of accessibility issues commonly found in web forms. Unlike static content, forms involve sequential tasks, such as perceiving labels, following instructions, providing input, and responding to validation feedback, where breakdowns at any step can block completion.

While the previous WCAG success criteria mapping helped identify success criteria amenable to automation or LLM-based reasoning, its application to forms revealed key limitations. In practice, form-related issues rarely map neatly onto a single success criterion, often spanning across multiple guidelines. For instance, a required field indicated only by color implicates *1.4.1 Use of Color*, *1.3.1 Info and Relationships*, and *3.3.2 Labels or Instructions* simultaneously. This overlap complicates efforts to assess accessibility solely in terms of individual criteria.

For this reason, when applying the mapping to forms in practice, it became more relevant to identify issues directly, rather than reasoning solely from the criteria. This perspective allowed us to capture barriers as they manifest in real scenarios, while still linking them back to the WCAG framework. It also revealed cases that might be overlooked if considered only from the success criterion level. For instance, when assessing automation feasibility across all criteria, *1.4.1 Use of Color* did not appear directly relevant, as it is typically addressed by static tools and was not included among the selected criteria. Yet, when this criterion was reframed as a concrete, form-specific issue, such as **"error states indicated only by color"**, it became feasible to evaluate within the pipeline.

Thus, rather than treating success criteria as the primary unit of analysis, we constructed a broad mapping of form issues to WCAG success criteria. This more focused approach ensured that the pipeline targeted the issues most representative of real-world barriers, while still grounding the analysis in compliance with WCAG.

Initial Broad Mapping

The first step in building this mapping was to enumerate a broad set of form-specific accessibility issues and link each one to the most relevant WCAG success criteria they span across. This mapping included a wide range of accessibility issues, such as missing labels, incorrectly associated labels, and the use of placeholders instead of labels, as well as more interaction-oriented issues like missing error identification, ambiguous or color-only error messages, missing programmatic cues, or dynamic updates not being announced. Each issue was then mapped to its related WCAG success criteria that fully or partially addressed it:

- **Missing Labels** — 3.3.2 Labels or Instructions; 2.4.6 Headings and Labels; 1.3.1 Info and Relationships.
- **Labels that Do Not Match Visible Text** — 1.3.1 Info and Relationships; 2.5.3 Label in Name; 2.4.6 Headings and Labels.
- **Incorrectly Associated Labels** — 1.3.1 Info and Relationships; 4.1.2 Name, Role, Value.
- **Unclear or Missing Instructions for Form Fields** — 3.3.2 Labels or Instructions.
- **Incorrect Input Purpose Identification (Autocomplete invalid)** — 1.3.5 Identify Input Purpose.

- **Failure to Identify Errors or Invalid Identification** — 3.3.1 Error Identification; 1.3.1 Info and Relationships.
- **Ambiguous or Generic Error Messages** — 3.3.1 Error Identification; 3.3.3 Error Suggestion.
- **Auto-Submit on Input Changes** — 3.2.2 On Input.
- **Empty Accessible Name on Form Elements** — 2.4.6 Headings and Labels; 4.1.2 Name, Role, Value.
- **Time Limits Without Warnings or Extensions** — 2.2.1 Timing Adjustable.
- **Missing Programmatic Cues (Disabled, Read-Only, Required)** — 1.3.1 Info and Relationships; 4.1.2 Name, Role, Value; 3.3.2 Labels or Instructions.
- **Unclear Labels** — 2.4.6 Headings and Labels.
- **Placeholders Used Instead of Labels** — 1.3.1 Info and Relationships; 3.3.2 Labels or Instructions.
- **Inaccessible CAPTCHA** — 1.1.1 Non-text Content.
- **Form Submission Without Reversibility (Legal, Financial, Data)** — 3.3.4 Error Prevention (Legal, Financial, Data).
- **Form Submission Without Reversibility** — 3.3.6 Error Prevention (All) (AAA).
- **Keyboard Navigation Issues** — 2.1.1 Keyboard.
- **Receive Focus Issues** — 3.2.1 On Focus.
- **Improper Focus Order** — 2.4.3 Focus Order.
- **Small Touch Targets** — 2.5.8 Target Size (Minimum).
- **Required Fields Indicated Only by Color** — 1.4.1 Use of Color; 1.3.1 Info and Relationships.
- **Error States Indicated Only by Color** — 1.4.1 Use of Color; 1.3.1 Info and Relationships; 3.3.1 Error Identification.
- **Related Fields Not Grouped for Clarity** — 1.3.1 Info and Relationships.
- **Missing Autocomplete** — 1.3.5 Identify Input Purpose.
- **Dynamic Form Updates Not Announced** — 4.1.3 Status Messages.

This mapping provided a clear overview of the accessibility issues present in forms. Many issues span multiple criteria and are only partially addressed by available tools, which further emphasizes the complexity of automating form evaluations.

Focused Target Issues

From the broad set, we excluded issues already covered by automated tools, those not feasible for reliable automation in our pipeline, such as requiring human-level interpretation or complex multi-step interactions, and those corresponding to WCAG AAA criteria. The excluded issues are grouped below, followed by the final focused set of issues targeted by the pipeline.

Issues already covered by automated tools:

- Missing Labels
- Incorrectly Associated Labels
- Empty Accessible Name on Form Elements
- Missing Autocomplete

Issues not feasible for reliable automation in this pipeline:

- Inaccessible CAPTCHA
- Time Limits Without Warnings or Extensions
- Form Submission Without Reversibility (legal/financial/data)
- Dynamic Form Updates Not Announced
- Keyboard Navigation Issues
- Receive Focus Issues
- Improper Focus Order
- Auto-submit on Input Changes

Issues corresponding to out-of-scope WCAG AAA criteria:

- Small Touch Targets
- Form Submission Without Reversibility

Finally, the resulting focused set of issues is presented on Table 3.2, aligned with their WCAG success criteria and indicating whether coverage is partial or complete.

Table 3.2: Final selection of form issues and their corresponding WCAG criteria.

Type of Issue	Related WCAG Criteria
Missing Programmatic Cues – Required	**1.3.1 Info and Relationships** — "Information, structure, and relationships can be programmatically determined or are available in text." *Covered Partially:* Checks whether a required field is programmatically marked using `required` or `aria-required`, and whether there's a visible cue (e.g., an asterisk) that matches the semantic indication. **4.1.2 Name, Role, Value** — "For all user interface components the name and role can be programmatically determined." *Covered Partially:* By checking for `required`/`aria-required`, ensures that the role of the input field is programmatically available. **3.3.2 Labels or Instructions** — "Labels or instructions are provided when content requires user input." *Covered Partially:* Checks whether the field's required status is clearly indicated through semantic markup and/or visual cues. **3.3.1 Error Identification (not directly evaluated)** — Pipeline checks whether a visible, textual error message is displayed when a required field is left empty, but does not fully evaluate compliance.
Failure to Identify Errors or Invalid Identification	**3.3.1 Error Identification** — Requires that the error must be shown in text and linked to the field. **1.3.1 Info and Relationships (Partial)** — If the error is shown visually (e.g., red outline) but not programmatically.
Ambiguous or Generic Error Messages	**3.3.1 Error Identification (Partial)** — Requires errors to be described in text, though not necessarily helpful text. **3.3.3 Error Suggestion (Partial)** — Requires suggestions if input errors are automatically detected.
Required Fields Indicated Only by Color	**1.4.1 Use of Color** — Color cannot be the only visual means of conveying information. **1.3.1 Info and Relationships (Partial)** — If color is used to imply "required", that information is not programmatically conveyed.
Error States Indicated Only by Color	**1.4.1 Use of Color** — Color cannot be the only visual means of conveying information. **3.3.1 Error Identification (Partial)** — Error must be described in text; color-only feedback is insufficient. **1.3.1 Info and Relationships (Partial)** — If the field changes color without a programmatic change, semantic value is lost.

It is worth noting that although the accessibility issue **"Required fields indicated only by color"** was included in the final set, in practice, we could not meaningfully evaluate it within the context of our pipeline. Detecting whether a field is identified as required only by color is a static property of the page, whereas our approach determines if the field is required by observing validation feedback after submission. Because of this mismatch, the check partially overlapped with "Error states indicated only by color", and its non-overlapping aspects fell outside the interaction-

based logic adopted in this work. As a result, despite being part of the final selection, the accessibility issue "Required fields indicated only by color" ultimately falls outside of the scope of issues the pipeline can evaluate.

Having refined the analysis to a focused set of form-specific accessibility issues, the next stage was to ground this framework in practice by selecting a representative set of web pages for evaluation.

3.3 Web Page Selection

For this work, it was important to evaluate real-world web pages containing forms. Rather than relying on artificially constructed test cases, we selected live web pages to better capture the diversity and complexity of modern interfaces. Using real forms instead of created examples makes the results more robust, as it captures the diversity and complexity of modern interfaces.

To capture this diversity and ensure representativeness, we selected high-traffic websites both globally and in Portugal, drawing on external references such as the Ahrefs' list of most visited sites[1]. Eligible pages had to include at least one interactive form (e.g., registration, login, checkout, or contact), ensuring a balance between internationally popular services and locally relevant platforms. This approach increased the likelihood of encountering varied form structures and validation mechanisms.

Challenges of Live Page Selection

One of the first challenges encountered was that some pages changed their structure or behaviour during the study, which was problematic since these pages served as the recurring test base and needed to remain stable to ensure result consistency.

To address this, we initially considered freezing the pages by capturing their static HTML, in order to preserve them even if the live versions later changed. However, this approach proved to be unreliable. Many of the selected pages rely heavily on dynamic implementations, such as JavaScript-rendered elements, session tokens, or asynchronous requests, which were not preserved in the static captures. In addition, essential interactions (e.g., field validation or enabling the submit button) often stopped working when the page was frozen. As a result, the static versions did not accurately reflect the real-world behaviour of the selected pages, and the evaluations would not have been reliable.

Given these limitations, we ultimately decided that working exclusively with the live versions of the pages was the lesser trade-off, as it preserved realistic functionality even at the cost of potential structural changes over time.

[1] `https://ahrefs.com/websites`

List of Tested Pages

Building on this selection process, we compiled an initial set of live web pages covering diverse form types, from registration and checkout to contact and search utilities. Because these pages were live and not frozen, their structures and availability could change over time. As a result, not all of them were ultimately retained in the analysis, a breakdown of which, along with the reasons for exclusion, is provided in Chapter 4.

The complete initial list is presented below, with URLs included for reference.

- **Steam** — Registration form (`https://store.steampowered.com/join`).
- **MEO** — Registration form (`https://login.telecom.pt/Public/Register.aspx`).
- **NBA** — Registration form (`https://www.nba.com/account/sign-up`).
- **Bible Gateway** — 'Advertise with us' contact form(`https://www.infinite.media/bible-gateway/`).
- **Continente** — Newsletter form (`https://www.continente.pt/loja-online/contactos/`).
- **Amazon** — Registration form (`https://www.amazon.com/ap/register`).
- **Staples Portugal** — Registration form (`https://www.staples.pt/pt/pt/registo`).
- **IPMA** — Contact form (`https://www.ipma.pt/pt/siteinfo/contactar.jsp`).
- **CTT** — Postal Code Search Form (`https://www.ctt.pt/feapl_2/app/open/postalCodeSearch/postalCodeSearch.jspx`).
- **RedditAds** — Registration form (`https://ads.reddit.com/register`).
- **WebMD** — Doctor profile contact form (`https://doctor.webmd.com/learnmore/profile`).
- **iLovePDF** — Contact form (`https://www.ilovepdf.com/contact`).
- **Fandango** — Contact form (`https://support.fandango.com/fandangosupport/s/contactsupport`).
- **Medical News Today** — BMI calculator (`https://www.medicalnewstoday.com/articles/323586#bmi-calculators`).
- **Reddit Business (Speak with Expert)** — (`https://www.business.reddit.com/speak-with-a-reddit-ads-expert`).

- **CTT Station Search** — (https://appserver2.ctt.pt/feapl_2/app/open/stationSearch/stationSearch.jspx).
- **AccuWeather** — Contact form (https://www.accuweather.com/en/contact).
- **Trustpilot Business** — Signup (https://business.trustpilot.com/signup).
- **Continente Checkout** — Delivery form (https://www.continente.pt/checkout/entrega/).
- **Decathlon Support** — (https://suporte.decathlon.pt/hc/pt/requests/new).
- **GSMArena** — (https://www.gsmarena.com/tipus.php3).
- **Merriam-Webster** — Registration (https://cam.merriam-webster.com/registration).
- **Discord Support** — (https://support.discord.com/hc/en-us/requests/new).
- **CTT Enrollment** — (https://appserver2.ctt.pt/femgu/app/open/enroll/showUserEnrollAction.jspx).
- **iStockPhoto Support** — (https://www.istockphoto.com/customer-support).
- **Quora Business** — Contact form (https://business.quora.com/contact-us).
- **Cookpad** — Premium signup (https://cookpad.com/us/premium_signup/unified_subscription_purchases/new).
- **IMDb Contribution** (https://contribute.imdb.com/updates/edit).
- **Etsy Onboarding** (https://www.etsy.com/your/shops/me/onboarding).
- **Genius New Submission** (https://genius.com/new).

Having established both the accessibility issues to be tackled and the set of web pages on which they would be tested, the next step was to design the evaluation pipeline.

3.4 Pipeline Design and Workflow

This section outlines the evaluation pipeline's workflow, as illustrated in Figure 3.1. The diagram presents the main modules and their interactions, showing how automated browser-based interaction, mutation tracking, contextual retrieval, and LLM-based evaluations are combined into a cohesive process.

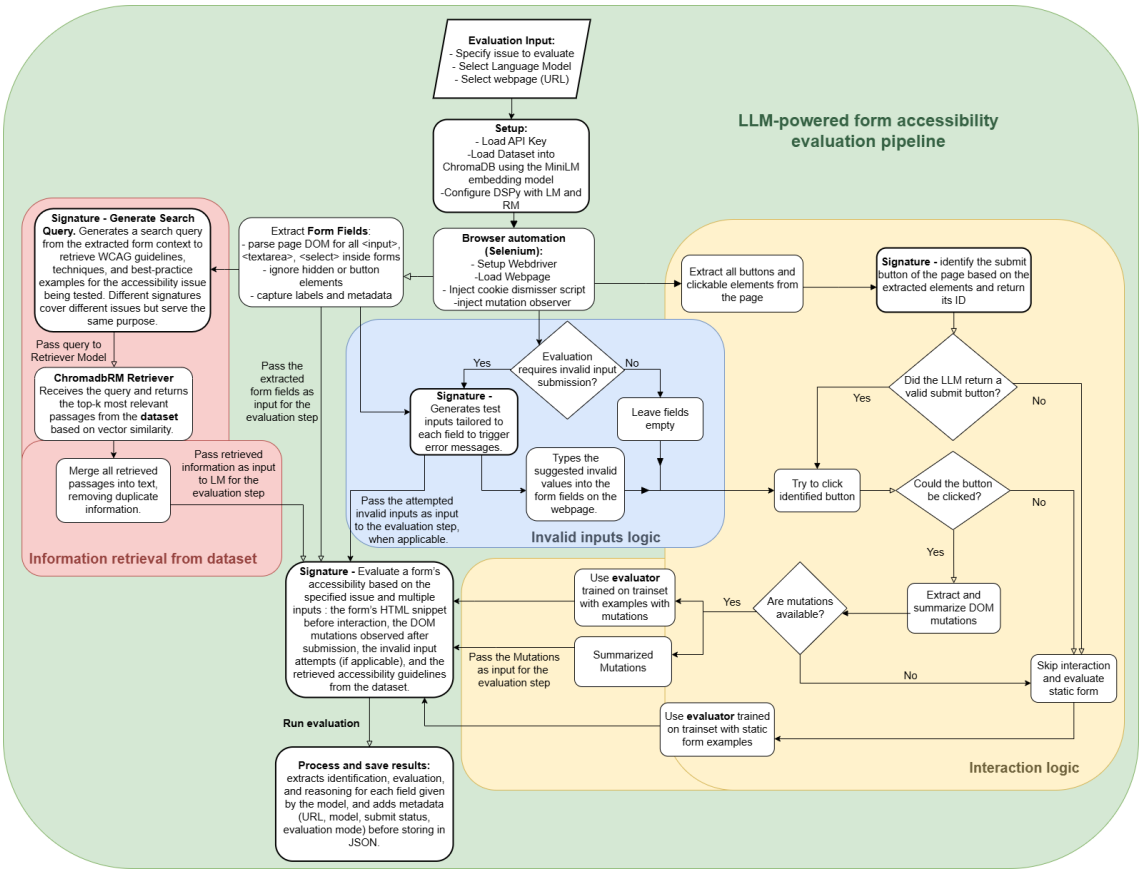

Figure 3.1: Workflow of the accessibility evaluation pipeline.

The design of the pipeline directly addresses the limitations identified in the previous sections. While automated tools reliably detect structural issues, they struggle with context-sensitive problems. Manual testing can handle these, especially when subjective judgment or interaction is required, but it is costly and difficult to scale. To bridge this gap, the pipeline integrates controlled browser interaction with real pages, automatic tracking of DOM mutations, retrieval of relevant accessibility guidance, and interpretative evaluation by an LLM.

The workflow connects these components in a structured sequence. It begins with the extraction of form fields and clickable elements from the page. Among these, the system identifies the submit button, which serves as the entry point for interaction. The browser is then automated to simulate form submissions: depending on the accessibility criterion under evaluation, fields may be left empty or filled with invalid inputs to trigger different types of error messages. The submission process is monitored for DOM changes, which are summarized into structured mutation traces when detected. If interaction is not possible or no mutations occur, the pipeline falls back to static evaluation, meaning only the HTML structure and attributes are analyzed.

In parallel, the system retrieves contextual information from the accessibility dataset using automatic query generation, returning relevant WCAG guidance and best-practice examples. These retrieved passages are combined with the extracted form fields and any detected mutations to form a structured evaluation signature, which is passed to the LLM. The model combines all available

information, i.e., extracted form fields, detected DOM change, and retrieved WCAG guidance, to perform the accessibility assessment. The output is returned in JSON format and contains, for each form field, a pass/fail outcome and a reasoning trace explaining the model's decision.

3.4.1 Web page automated interaction

The first stage of the pipeline establishes a controlled browser session using Selenium WebDriver[2], which provides a stable environment to simulate user behaviour on real pages. To avoid inconsistencies caused by browser personalization, all final evaluations were performed on a fresh Chrome instance with a temporary user profile, meaning no cached data, saved credentials, or logged-in accounts influenced page rendering. This choice was made because early tests revealed inconsistencies, most notably on Amazon's sign-in form page, where the form layout changed depending on whether the browser was tied to a logged-in Chrome account.

Once the browser is launched, the target webpage is loaded. A cookie-dismissal script is then injected to automatically detect and interact with consent banners that would otherwise block access to form elements. During initial evaluations, these cookie pop-ups frequently prevented interaction with the forms on the page. To address this, a dedicated script was developed, tailored to the selected set of pages but designed with layered detection strategies. It attempts to click on well-known banner IDs, like OneTrust and Cookiebot, generic cookie-related attributes, and fallback patterns using accessibility labels. This ensured that the evaluation proceeded without any manual intervention.

After the browser environment is established and cookie banners are automatically dismissed, the pipeline proceeds with the automated extraction of form fields. At this stage, the DOM is parsed to collect all `<input>`, `<textarea>`, and `<select>` elements contained within forms, while ignoring hidden inputs or buttons. For each extracted element, the corresponding labels and metadata are captured to preserve the context needed for the subsequent evaluation.

Dynamic behaviour is handled separately via a custom mutation observer, explained in Section 3.4.2, which records accessibility-relevant changes triggered during interaction.

We then extract all buttons and clickable elements on the page. Since submit buttons are not always explicitly labeled or distinctive, the extracted candidates are passed to an LLM, which returns the element most likely to be the primary submit control. The automation then attempts to trigger this submit action directly, submitting the form to reveal the errors needed for evaluation. The submission strategy depends on the accessibility issue under evaluation, but for each issue, only a single submission is performed.

When testing required fields and error identification, the form is submitted with all inputs left empty to trigger validation. For error suggestion and error states conveyed only by color, the fields are instead populated with deliberately invalid inputs. These inputs are chosen to mismatch the expected format of each field (e.g., an email without "@" or a password that is too short) and are further refined with LLM-generated candidates, as further detailed in Section 3.4.4. This ensures

[2]https://www.selenium.dev/documentation/webdriver/

that each submission meaningfully triggers the type of error feedback relevant to the criterion being tested.

Finally, once submission is attempted, the pipeline briefly pauses to allow client-side validation scripts to execute before retrieving the records from the mutation observer. These records are preserved for later evaluation, where they are aligned with the extracted form fields and the relevant retrieved information.

3.4.2 Mutation observer logic

To capture the dynamic behaviour of forms, the pipeline integrates a custom mutation observer into the page. In the earliest version, the evaluation relied on passing the full HTML of the form both before and after submission. While this provided a complete snapshot, it was ill-suited for more complex cases: the resulting HTML was often very large, sometimes exceeding token limits, and offered no clear indication of which parts of the DOM had actually changed. This made it difficult for the model to reason about detailed validation behaviours or subtle accessibility cues.

The decision to integrate a dedicated mutation observer was therefore a significant step in the right direction. Instead of comparing two full DOM states, the observer tracks only the relevant changes that occur during interaction related to the accessibility issues we tackled. This not only reduces the volume of data but also highlights the signals needed to evaluate whether errors and different field states are conveyed programmatically, visually, or both.

The observer catches three main categories of changes:

- **Attribute modifications** - Changes in attributes, including ones such as `aria-invalid`, `aria-describedby`, `style`, `class`, `hidden`, and `data-error` are recorded. Each update is logged with the old and new values. When classes or styles change, the system also stores the before and after values of selected visual properties, specifically `border-color`, `background-color`, `color`, `outline-color`, and `box-shadow`, capturing when a field becomes visually marked as invalid through styling.

- **Structural changes** - Node insertions and removals are captured, with the recording of the full `outerHTML` of added or removed elements. Because error messages are often injected dynamically, the observer also inspects sibling and parent nodes for short, human-readable text, which it records as potential error messages linked to the relevant input.

- **Native validation states** - After each mutation cycle, all invalid fields detected by the built-in browser mechanisms are queried. Any `validationMessage` strings are stored, ensuring that native feedback is preserved even if it does not appear as a DOM node.

The raw mutation records collected in the browser are retrieved and then filtered into compact JSON objects. Each record preserves essential metadata (if available): mutation type, target tag and ID, field label/name, changed attributes, style diffs, error-related classes, and possible error messages. Noise from hidden fields, tokens, and large dropdowns is filtered out. Since each

evaluation run covers only a single form submission, all records observed after the submit button is triggered are collected together. These are then reformatted into a structured text summary before being passed to the LLM that will perform the evaluation.

Instead of dumping raw DOM changes, the pipeline turns them into readable lines such as "[ARIA Invalid] `<input>` marked as invalid" or "[Linked Error Message] 'Password is required' for field 'Password'." If the change is purely visual (like a color update) and has no semantic backup, it gets annotated as a potential SC 1.4.1 failure. If an error message shows up without being tied to a specific field, it is flagged with a warning. These annotations are intentionally framed as "potential" or "warning" so the model treats them as cues, not as guaranteed failures.

This way, instead of the LLM receiving a messy HTML dump as input for evaluation, it gets a cleaned and annotated log that highlights accessibility-relevant behaviours. Paired with the static extraction of form fields, this structured summary gives the dynamic side of the evidence needed to evaluate the selected accessibility issues.

3.4.3 Form submission logic

Form submission is the trigger for exposing the dynamic feedback required in this study. The issues targeted fall into four different checks: missing programmatic cues for required fields, failures in error identification, ambiguous or generic error messages, and error states conveyed only by color.

In the initial stages, the focus was on evaluating the use of the required attribute and error identification. These could be meaningfully evaluated by submitting the form with empty inputs, since such a submission reliably activates the built-in validation mechanisms responsible for flagging missing or incorrectly identified required fields. However, when expanding the scope to ambiguous error messages, it became evident that an empty submission would be too limited: it consistently produced only required-field messages, usually identical across fields, and therefore did not provide sufficient variation to evaluate the clarity or specificity of feedback. At this point, the submission strategy was adapted to include invalid test values for each field, so that different error states could be triggered and assessed. The same approach was later extended to the criteria involving use of color, since those rely on the visual presentation of invalid states rather than on the absence of input alone.

Technically, this process begins by identifying the candidate submit controls on the page and selecting the most likely form submit button with the aid of the LLM, using a different call than the one to evaluate. Once identified, the pipeline performs a single submission attempt in one of two modes: empty submission, used for required field and error identification issues, or invalid input submission, used for error message clarity and error states indicated only by color. Invalid values can be generated through predefined patterns but are refined by LLM suggestions, which increase the likelihood of triggering realistic validation failures for the given field type. Limiting each evaluation run to only one submission ensures that the automated interaction reliably produces the necessary validation responses for the criteria under evaluation without introducing ambiguity

from repeated attempts.

3.4.4 LLM-guided interactions

In addition to the final evaluation step, the pipeline also relies on LLMs to guide intermediate interactions with the form. These are handled in separate calls, with inputs and output formats specifically designed for the task at hand. Rather than performing the interaction itself, LLM provides short, structured decisions that resolve ambiguity in relevant moments of the submission process. In this context, the two cases where LLMs are employed are the identification of the correct submit button among all clickable elements from the form, and the generation of contextually meaningful invalid input values that expose different error feedback than empty submissions alone. This approach makes each reasoning step reproducible and easier to integrate into the broader automation, despite the inherent variability of model outputs.

Submit button identification

Webpages and their forms often have multiple clickable elements, navigation buttons, or links styled as buttons, which makes automating the submission of the form a harder task to perform. To handle this possible constraint, the pipeline integrates an LLM call in this process. The language model takes as input the structured list of button elements previously extracted from the page, including their attributes such as `tag`, `text`, `title`, `id`, `name`, `type`, `value`, and `onclick`. Using this information, the model is prompted to select the element most likely to act as the form's submission button, or to return *'None'* if no such element is found. Once the model provides this identifier, the pipeline locates the corresponding button from the list, determines its XPath, and attempts to click it through Selenium. If the direct click is not possible, it retries with alternative strategies such as JavaScript execution.

Invalid input suggestion

To capture a wider range of error states beyond empty submissions that would only trigger required fields, the pipeline prompts the language model to suggest intentionally incorrect values for each form field. The model receives two types of information as input: a structured list of the fields extracted from the page, including their labels, input types, and any visible hints, and a goal statement that explicitly instructs the model to generate values likely to trigger error messages. The output is a structured mapping where each field is paired with one invalid value tailored to its type. For example, the model may propose an email without an "@" symbol or alphabetic characters in a numeric field. This targeted approach ensures that the invalid values are realistic enough to expose meaningful feedback, rather than arbitrary or random strings.

Once these suggestions are produced, the automation layer attempts to apply them to the live form. Each field is located by its identifier or name and filled with the proposed value using Selenium. If a field cannot be used—because it is hidden, disabled, read-only, or of an unsupported type such as file uploads, selects, or checkboxes—it is skipped. When no model suggestion can

be applied, the system falls back on a set of predefined patterns that cover the most common input types. These include:

- **Email fields:** `userexample.com` (missing "@")
- **Passwords:** `1234` (too short/simple)
- **Dates:** `32/13/2023` (impossible date)
- **Phone numbers:** `abc123` (non-numeric string)
- **Numeric fields:** `abc` (non-numeric string)
- **URLs:** `htp:/invalid` (malformed url)
- **Other input types:** `!` (special character)

By combining LLM-generated suggestions with these targeted fallbacks, the pipeline ensures that nearly every eligible field is tested with an invalid value. This approach produces consistent coverage while still benefiting from the contextual creativity of the model when generating more nuanced test cases.

In summary, the LLM here does not replace automation, but complements it with short, structured decisions that remove ambiguity and make interactions more adaptive. Each call is narrow in scope, tied to a specific issue, and backed by fallbacks so that the process can continue even if the model output is not ideal. This differs from the evaluation stage, where the LLM takes on a broader role, interpreting the form state and producing the final assessment with reasoning.

3.4.5 Dataset Construction and Trainset Compilation

The evaluation module requires two complementary forms of grounding: external knowledge to guide model reasoning, and internal examples to constrain outputs into predictable formats. To address these needs, the pipeline combines a retrieval-augmented generation (RAG) dataset with DSPy trainsets. The dataset provides compact, field-level accessibility cases, each linking a form element to one or more WCAG criteria, a failure example, a best-practice correction, and an explanation, that can be retrieved during evaluation. On the other hand, the trainsets provide structured demonstrations aligned with the actual inputs and outputs of an evaluation run. Each entry pairs raw HTML, captured mutations, and retrieved guidelines with the expected outputs (field identification, pass/fail judgment, and reasoning), ensuring that signature compilation produces consistent and usable results.

Dataset and RAG retrieval

Initially, the plan for the workflow was to provide the model with long passages from WCAG documentation, such as success criteria, ACT rules, sufficient techniques, failures, and best practices,

alongside the form's HTML and any captured mutations. In practice, this approach produced oversized prompts, making the evaluation unstable, and diluted the relevance of the retrieved guidance information.

To make retrieval more focused, we constructed a structured dataset at the level of individual form fields. Each entry links a field type to one or more WCAG criteria, a failure example, a best-practice correction, and a short explanation.

An example entry is shown below:

```
{
  "id": 10,
  "field_type": "error message",
  "guideline": "WCAG 3.3.3 Error Suggestion",
  "failure_example": "<span class='error'>Invalid input</span>",
  "best_practice": "<span class='error'>Invalid input. Please enter in
YYYY-MM-DD format.</span>",
  "explanation": "Error messages should provide suggestions for
correction, not just indicate failure."
}
```

These entries act as compact knowledge units that the model can consult during evaluation. For instance, one entry describes a text field that signals errors only by applying a red border, without any textual or programmatic indication. The case is tied to WCAG 1.4.1 (Use of Color) and 3.3.1 (Error Identification) and is paired with a corrected version that adds aria-invalid, a descriptive error message, and an aria-describedby reference. The explanation clarifies why color alone is insufficient and how combining textual and programmatic cues makes the error perceivable and operable for all users.

The dataset is stored in a persistent ChromaDB[3] collection, with each entry embedded using a SentenceTransformer model (all-MiniLM-L6-v2)[4] and indexed for retrieval. To ensure that the most relevant entries are retrieved for the current evaluation, the pipeline generates search queries dynamically through a complementary LLM call. This step is similar in scope to the interactions described earlier: the model receives a structured summary of the form fields together with the evaluation goal and returns a single query string that directs retrieval toward the accessibility aspect under analysis. Retrieval is executed through ChromadbRM, returning the top-k closest dataset entries (k=8 was found optimal). To increase robustness, the pipeline performs multi-hop retrieval with up to two iterations (`max_hops=2`), selecting two new passages per step (`passages_per_hop=2`). At each retrieval step, the query is refined, previously unseen passages are aggregated with earlier results, and if no relevant results are found the system falls back to broader queries.

The retrieved passages, each containing a failure case, a best practice, and an explanation linked to the WCAG criteria, are supplied to the evaluation signature together with the extracted HTML summary and, where applicable, mutation logs or invalid input attempts. In practice, this

[3] https://docs.trychroma.com/docs/overview/introduction
[4] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

makes the evaluation resemble a guided reasoning process, where the model works with targeted accessibility precedents instead of long undifferentiated text. This helped reduce irrelevant output, kept reasoning closer to WCAG expectations, and provided more stable results across varied forms.

Trainsets for fine-tuning

While the dataset provides external precedents, the trainsets are used to fine-tune the evaluation module so that outputs follow predictable formats. Each trainset consists of direct examples showing how the evidence collected from a form should map to an accessibility evaluation. These examples are built around a consistent structure with two sides:

- **Input side:** Includes the extracted form fields before interaction, any dynamic evidence captured after submission (such as error messages or attribute changes), and the relevant accessibility guidance retrieved from the dataset.
- **Output side:** Contains the expected identification of the field, the evaluation result (*pass*, *fail*, or *inapplicable*), a short reasoning explaining the decision made by the model and a structured output of everything formatted together.

The number of examples varies across trainsets, since we aimed to cover a range of scenarios for each criterion without overwhelming the signature compilation. Therefore, each trainset contains a mix of positive and negative cases, with examples of fields that pass, fail, or are marked inapplicable. This ensured that the model could generalize across different situations while still keeping the training examples balanced and representative.

For each accessibility check, two trainsets were created. The primary trainset contained examples with the full range of evidence collected during interaction, including captured mutations when available. Alongside it, a complementary fallback trainset was prepared for cases where interaction with the form could not be completed, such as when the submit button could not be clicked or when mutations could not be extracted. These fallback trainsets relied only on static form extraction, which often led to results marked as *inapplicable* due to lack of evidence. These fallback outputs were excluded from the final evaluation metrics, but the trainsets themselves were still necessary to keep the pipeline functional when interaction failed.

An example entry from the error identification trainset is shown below. It demonstrates a case where an error message is present in the HTML but is not programmatically associated with the input field:

```
# FAIL: Error message present but not linked
dspy.Example(
 html_snippet_before="""<input type='text' id='username'>
<span class='error'>Username required</span>""",
    mutations="[Message Revealed] Username is required.
(Field: 'username')",
    retrieved_guidelines="Error messages must be programmatically
```

```
associated with the input using aria-describedby or similar
mechanisms..",
    identification="username",
    evaluation="fail",
    reasoning="An error message exists in the HTML but is not
linked to the input with aria-describedby or 'for' attributes."
).with_inputs(
    "html_snippet_before",
    "retrieved_guidelines",
    "mutations",
)
```

This example illustrates how each trainset entry provides the model not only with the relevant field-level evidence and guidance but also with the expected form of the output. By compiling evaluation signatures with such examples, the model learns to produce judgments that are both contextually grounded and aligned with the pipeline's structured output requirements.

In the evaluation module, the dataset and trainsets play complementary roles. The dataset supports retrieval: the model generates search queries and receives compact WCAG-aligned precedents that act as external grounding. The trainsets, by contrast, are used at compilation time to shape the evaluation signatures with concrete demonstrations of input–output mappings. Together, they ensure that the model has both knowledge to consult and examples to imitate, balancing adaptability with consistency.

3.4.6 Preparing Inputs for LLMs

As mentioned before, in early stages of development we tried simply passing the full HTML and raw mutation logs straight into the model. While this looked like the most complete option, in practice it caused problems: prompts were noisy, sometimes hit token limits, and the model lost track of the evaluation task. This highlighted the need to filter and structure the inputs so that the model receives only what is relevant, for it to be more focused to perform the task in hand.

At this point, it is also important to clarify which inputs are actually passed to the LLMs throughout the pipeline. The LLM is invoked in four distinct contexts:

- **Main call**
  - **Accessibility evaluation** - issuing the final evaluation.
- **Complementary calls**
  - **Submit button identification** - selecting the element most likely to trigger form submission.
  - **Invalid input generation** - proposing test values designed to trigger varied error messages.
  - **Retrieval query formulation** - producing focused queries to fetch relevant WCAG-related guidance.

With one exception, all inputs passed to the LLMs in the complementary calls overlap with the main accessibility evaluation call. The only complementary call that receives unique input information is the submission button identification, which takes a specifically prepared list of all extracted buttons from the page and asks the model to select the one most likely to trigger the form submission. All other calls make use of the same core evidence that later drives the evaluation: the extracted form fields, the mutation logs (when available), the invalid inputs for testing error variation, and the retrieved guidance from the dataset. Among these, the extracted fields and mutation logs required the most attention in terms of preparation, since they were by far the largest and noisiest sources of data.

Extracted fields To show why this preparation step is so important, we can look at the raw HTML of a registration form like the one on Amazon. Even a single field comes with extra containers, hidden inputs, ARIA references, and lots of classes. Passing this directly would overload the model with irrelevant detail:

```
<div id="ap_customer_name_container" class="a-row a-spacing-base">
    <label for="ap_customer_name" class="a-form-label">
        Your name
    </label>
    <input type="text" maxlength="50" id="ap_customer_name"
           name="customerName" tabindex="1"
           class="a-input-text a-span12 auth-autofocus
auth-required-field"
           aria-required="true" autocomplete="name" />
    <input type="hidden" name="metadata1" value="random-token" />
</div>
```

For evaluation, most of this doesn't matter: extra divs, CSS classes, and hidden inputs do not affect whether the field can be understood or used. If we passed the entire block as it is, the model would have to work through a lot of noise, increasing the chance of confusion and even hitting token limits on larger forms.

Instead, the parsing step turns this raw markup into a structured version that keeps only the attributes relevant for accessibility. For the same Amazon field, the extracted output looks like this:

```
{
  "label": "Your name",
  "label_source": "label[for]",
  "name": "customerName",
  "id": "ap_customer_name",
  "type": "text",
  "value": "",
  "required": "yes",
  "required_source": "aria-required",
  "disabled": "no disabled attribute",
  "readonly": "no read-only attribute",
  "autocomplete": "name",
  "inputmode": "no inputmode"
}
```

This structured block feeds other steps like generating invalid inputs or search queries, but for evaluation we parse it once more. The evaluator model does not need every detail, as it only needs a clear view of the attributes that matter most for reasoning about WCAG compliance.

For this reason, the fields are reformatted into short, consistent text blocks such as:

```
Your name (text): required=yes, disabled=no disabled attribute,
readonly=no read-only attribute, autocomplete=name
```

This preparation happens in two layers: first, the raw HTML is distilled into a structured form that captures accessibility cues, and then that form is rewritten into a compact summary that the model can reason over more effectively.

Mutation Logs The second input that needed preparation was the set of DOM mutations captured after form submission. These logs come directly from the `MutationObserver`, which records every change during interaction: added nodes, attribute updates, styles modifications, and so forth. The browser produces a large JSON dump, precise, but overloaded with detail. A fragment looks like this:

```
{
  "addedNodes": [
    {
      "tag": "div",
      "class": "a-alert a-alert-error",
      "text": "Enter your name"
    }
  ],
  "attributeChanges": [
    {
      "id": "ap_customer_name",
      "attribute": "aria-invalid",
      "oldValue": null,
      "newValue": "true"
    }
  ],
  "styleChanges": []
}
```

The essential information here is simple: an error message *"Enter your name"* appeared, and the field was flagged as invalid with `aria-invalid=true`. But that signal is buried inside a lot of technical detail the model does not need. To make it usable, the raw logs are reduced into short, field-level summaries. The same example becomes:

```
Field: Your name (id=ap\_customer\_name) \\
- Error message added: ``Enter your name'' \\
- Attribute changed: aria-invalid set to true
```

This way the model only sees what matters for the accessibility evaluation. The raw observer output stays available if needed, but the evaluation pipeline always works on the parsed version. That makes it possible to connect static field descriptions with dynamic feedback, giving the evaluator a clear unit of evidence for each input.

Other Inputs Besides extracted fields and mutation logs, the pipeline also prepares three smaller inputs: retrieved guidelines, invalid inputs, and extracted buttons. These are less noisy but still go through a quick parsing step.

Guidelines - Each dataset entry links a field type to a WCAG criterion and a short best-practice explanation. During evaluation, retrieval pulls a handful of entries, which are then cleaned up and flattened into plain text. The result is a compact set of references the model can draw on without duplication.

Invalid inputs - These are only generated for criteria where error variation matters: error clarity and use of color. The model suggests test values tailored to the field type, for instance, an email without "@", a password that is too short, or a number field filled with letters. These suggestions are turned into simple field, value pairs and re-submitted, so the evaluator can reason about error messages at the field level, not just the form level.

Buttons - The parser walks through the DOM to collect all visible, clickable elements that could act as submit triggers: standard `<button>` and `<input>` elements, but also others with `onclick` handlers. Each is reduced to a structured record (id, text, role, attributes). The model then picks the one most likely to be the real submit button. This is the only extra model call outside the evaluation loop, but it follows the same logic: cut down raw HTML into focused evidence.

All of these steps follow the same principle: take noisy raw data and turn it into compact, field-level evidence the model can actually work with. Keeping the granularity at the level of individual inputs avoids token bloat and keeps the reasoning focused on one criterion at a time.

3.4.7 Signature Design and Retrieval Integration

An important part of the pipeline design was the definition of DSPy signatures and their connection with the retrieval process. In DSPy, a signature provides a declarative specification of an LLM call by defining its inputs and expected outputs. Inputs capture structured evidence passed to the model, such as the extracted fields captured from the page, while outputs constrain the model's response into a desired format. This design gave the pipeline a stable structure: each model call had a clear definition of inputs and outputs, reducing ambiguity and making the results easier to interpret.

For each accessibility issue selected in Section 3.3, we created a dedicated evaluation signature. These were not defined all at once, but developed and refined one by one through the analysis of results. This made it possible to evaluate each issue individually, since the pipeline can only run one evaluation task at a time using the set of signatures corresponding to the issue. While all signatures shared a common output structure, field identification, evaluation (*pass/fail/inapplicable*), and short reasoning, their evaluation rules and some inputs varied depending on the type of issue.

In what follows, we present each accessibility issue, listing its evaluation rule and highlighting distinctive inputs where relevant.

Required fields: This issue checks whether the `required` attribute (or equivalent) is correctly declared. The *EvaluateInteractiveCues* signature takes as input the form fields before submission, DOM mutations after an empty submission, and the retrieved guidelines.

Evaluation rule:

> For each field, determine only whether the 'required' attribute (or an equivalent mechanism) is used appropriately, based on whether the field behaves as required after submission. Classify each field as:
>
> - **pass**: The field is required and clearly uses the `required` attribute or an equivalent semantic cue (e.g., `aria-required`).
> - **fail**: The field is required but lacks the `required` attribute or any equivalent semantic indicator (e.g., `aria-required`).
> - **inapplicable**: The field is optional (not marked as required in behaviour) or not subject to required validation.
>
> Do not fail a field for unrelated issues such as missing error messages or lack of ARIA descriptions. This evaluation is only about required status and how it is declared.

Error identification: This issue addresses whether validation errors are not only shown but also associated with the correct field in an accessible way. The *EvaluateErrorIdentification* signature captured the HTML structure before submission and the mutations after submission (such as the addition of `aria-invalid` or an error message). It required the model to confirm that feedback was linked, ensuring users could identify which input failed and why.

Evaluation rule:

> Individually assess each field for accessible error identification after submission, and classify each one as 'pass', 'fail', or 'inapplicable':
>
> - **pass**: A specific error message is shown for this field, programmatically and visually associated.
> - **fail**: No specific error message is shown, or the message is not programmatically or visually associated with the field.
> - **inapplicable**: The field is optional or does not require validation when left empty.

Error clarity: This issue goes beyond identifying the presence of an error to assessing whether the validation feedback is understandable and not ambiguous or generic. For this evaluation, invalid inputs were entered in the form fields before submission for it to trigger different messages. The *EvaluateErrorClarity* signature extended the inputs with the invalid test values used, making it possible to test how clear the returned messages were.

Additional input:

`invalid_inputs`: List of values submitted into specific fields to test error handling. These values are intentionally unusual, malformed, or nonstandard, but not necessarily guaranteed to trigger validation.

Evaluation rule:

Assess whether the system provided clear guidance when a validation error was triggered. Use these options per field:

- **pass**: A validation error was triggered, and the message clearly explains the problem and how to fix it.
- **fail**: A validation error was triggered, but the message was missing, vague, or unhelpful.
- **inapplicable**: No validation was triggered, or no error was expected for the given input.

Error States Indicated Only by Color: This issue evaluates whether error cues rely solely on visual styling. The *EvaluateUseOfColor* signature considers invalid input submissions and resulting DOM mutations, such as red borders, background color changes, or added messages, as well as the extracted form fields and retrieved entries from the dataset.

Evaluation rule:

For each field, classify the Error cue based only on invalid input submissions.

- **pass**: Error is indicated with both programmatic and textual support such as `aria-invalid`, an error message linked via `aria-describedby`, a message with `role="alert"`, or visible text associated with the input.
- **fail**: The only error indicator is a visual treatment (red border, color change) with no textual or programmatic indication of error.
- **inapplicable**: No error was triggered for invalid input.

Only evaluate error cues in terms of reliance on color versus programmatic or textual alternatives. Don't evaluate whether there is programmatic association cues and other unrelated accessibility issues.

Retrieval integration. Alongside the evaluation signatures, we defined four different complementary retrieval signatures, one for each issue, that operationalize the connection to the accessibility dataset. Each of these signatures receives the same structured field representation used in evaluation, but instead of returning a judgment, the model is tasked with producing a focused search query for the issue at hand, with context on the form fields. The resulting query retrieves compact precedents from the dataset, each linking a WCAG reference to a failure case, a corrected example, and a brief explanation. These retrieved entries are then injected as additional context into the corresponding evaluation signature to provide contextual grounding for the evaluation. If the model fails to produce a valid query, the pipeline attempts a set of predefined fallback queries specific to that accessibility issue; when no query can be obtained at all, the evaluation proceeds without retrieval and the run is recorded as having been performed without retrieved context.

Shared design choices. Some choices were common across all signatures. Evaluations were always carried out per field instead of per form, which avoided grouping and made the results easier to read. Prompts were written in a direct style, with the task stated clearly. The output format was kept short and structured so that results could be compared automatically but also inspected by hand. These choices helped keep the evaluations consistent and linked to the evidence provided.

The design of the signatures provided a stable structure for evaluation, but also showed clear limitations. Early runs revealed missing reasoning steps, inconsistent formatting, and generalized answers when retrieval failed. These issues motivated the iterative refinements described in the next section.

3.5 Pipeline Iterations and Refinements

When starting the development of the pipeline, the process was incremental, adding and validating one accessibility check at a time: required fields, error identification, error clarity, and finally use of color. Each check revealed different weaknesses in input preparation, prompting, or interaction capture, so refinements were applied as needed. In practice, a fix made for one issue frequently revealed or enabled improvements for the next.

The first checks implemented focused on required fields. In the early stages we relied mainly on the Mistral Large model, since it was free to use and offered large token limits. At that point, the pipeline was not yet operating at the field level: we were passing entire raw page extracts into the model, including complete markup, container elements, hidden inputs, long class lists, and unfiltered mutation dumps. This turned out to be an issue, because outputs were noisy or broken, larger forms often exceeded token capacity. The mutation logs were also overwhelming, filled with irrelevant style and DOM changes that obscured the signals the model should have been focusing on.

To address this, we restructured the pipeline to work at the field level, with each input field extracted individually and described by a smaller set of attributes. Mutations were still noisy, but at least results could be tied to specific fields rather than entire forms. However, at this stage,

the performance was very inconsistent with false positives appearing frequently. For example, addressing the required check on a search input, the model incorrectly marked it as failing for missing required attributes and using a placeholder instead of a label, even though the field was optional and the check was for the use required attribute.

At this point, dedicated trainsets were already created to guide the model's evaluation. These were populated primarily with examples derived from WCAG, but each example was adapted to reflect how the model actually receives information through the pipeline. Balance was maintained across pass, fail, and inapplicable cases to avoid overfitting to any particular outcome. Throughout the entire development phase, recurring errors appeared during test runs, and new examples were added or changed to the trainsets to fill the gaps and steer the model toward correct judgments. Prompts in the signatures, in a similar way, were also refined, and while the overall structure remained consistent, instructions were sharpened to focus the model to produce what we aimed to evaluate, as sometimes it steered to evaluating different issues.

After we tightened the trainsets and refined the prompts, this type of reasoning, such as incorrectly evaluating the use of a placeholder instead of focusing on the required attribute task, appeared far less frequently, allowing the model to focus on the intended criteria more reliably. The output structure was also sometimes broken for some pages, or with fields grouped in one single evaluation, which was not the desired output but with the refinements made, the model started performing consistently in this sense.

As new checks were implemented, adjustments to the inputs passed to the LLM became necessary. Error identification exposed gaps in the mutation logs. Initially, messages were not clearly linked to their corresponding fields, which led to outputs that flagged unrelated issues such as missing labels. Mutation capture was revised so that visual and textual cues were explicitly associated with field metadata, and prompts were rewritten in a more declarative style to anchor the model's reasoning on the intended evaluation.

Evaluating error clarity introduced an additional layer of complexity. We began generating invalid input values and submitting forms with these variations to observe the resulting feedback. This required further refinements in input preparation and signature design. Prompts were adjusted to guide the model step by step, ensuring it could not only detect validation errors but also assess whether the feedback was clear and contextually appropriate. It was during the development of this check that the reasoning instructions were rewritten to be more targeted. Instead of asking the model to simply justify its evaluation, we required it to explain which indicators were detected (for example, aria-invalid, linked error messages, or red text), how those indicators supported the classification (pass, fail, inapplicable), and whether the feedback relied only on color or lacked proper programmatic association. This sharper structure helped reduce vague or unfocused explanations and made outputs more targeted. Once validated, we applied this refinement across every developed evaluation.

For the use of color check, it quickly became clear that what we had in place up to this point was not sufficient, as the pipeline was not capturing anything related to color changes. To address

this, we extended the mutation capture so that style-only error indicators, such as red borders or highlights, were recorded alongside textual or programmatic cues. Mutation parsing was then refined into concise, human-readable summaries that directly linked these visual changes to the associated field. The retrieval dataset also had to be revised, since until then it contained no color-related examples. We expanded it with adapted WCAG cases, ensuring the model could ground its evaluations on relevant precedents.

Different language models were used throughout development, shaping the refinement process. Early refinements were carried out mainly with Mistral Large (version mistral-large-2411), which was practical to use due to its free availability and large token limits. This version has since been retired, and the alias now refers to a different model (mistral-medium-2508), but all reported results correspond to the earlier version. Near the end of development, we introduced GPT-4o, 4o mini, 4.1, and 4.1 mini to run late-stage tests and begin gathering results. We also tested the pipeline with GPT-5, as it is the new flagship model.

Throughout development, outputs were repeatedly cross-checked against our manual evaluations of the same forms. This feedback loop was a crucial detail, because whenever the model produced inconsistent or incomplete results, the cause could usually be traced to weaknesses in input preparation, unclear prompts, insufficient retrieval coverage, or gaps in the trainsets. Each refinement was re-inserted into the pipeline, steadily improving both reliability and interpretability.

The cumulative effect of these refinements was a pipeline that matured from an early, noisy prototype into a structured and task-aware system. By incrementally addressing required fields, error identification, error clarity, and use of color, the pipeline evolved into a framework that balances general design choices with criterion-specific adaptations. At the same time, this iterative process highlighted both the conditions under which the approach is most effective and the limitations that remained despite refinement.

With the design phase complete, the next step was to test the refined pipeline systematically. The following chapter introduces the evaluation setup and presents the results, detailing which webpages were selected, how accessibility outcomes were benchmarked through manual annotation, and how the pipeline's outputs compared across different language models. Beyond accuracy, the analysis also examines the quality of the models' reasoning, assessing whether their explanations were clear and correct.

Chapter 4

Evaluation and Results

Building on the design and refinements described in the previous chapter, this chapter shifts the focus to the performance and results of the pipeline. The evaluation measures how accurate the system performs when evaluating the selected webpages and examines the quality of the reasoning produced by the model. Accuracy is assessed by comparing the pipeline's outputs against a manually created benchmark, while reasoning quality is evaluated based on the clarity, correctness, and usefulness of the explanations. Results are also contrasted with those obtained from automated accessibility tools for the same checks, while also comparing results across the five different language models used. In addition, the chapter considers the validity of the findings, outlining limitations found on our pipeline after the analysis of the results.

4.1 Evaluation Setup and Process

The evaluation started from the complete set of candidate pages introduced in Section 3.2. Each of these pages was tested end-to-end with the pipeline to determine whether form interaction, submission, and accessibility feedback could be achieved in practice. Not all pages met these conditions: several became unusable during testing, while others exposed barriers that prevented the pipeline from completing its process. As a result, only ten pages were ultimately retained for analysis. This section reports the outcome of this selection, presents the insights that emerged from testing, and outlines the benchmarking and evaluation metrics used to measure the pipeline's performance.

Outcome of Page Testing

Not all pages from the original selection could be included in the final evaluation. While some were successfully processed end-to-end by the pipeline, others failed due to either LLM-related misinterpretations or page-level constraints. For clarity, the outcomes are grouped into the different categories, with the first being the final subset used on our work:

Final subset of pages that successfully worked using the Pipeline:

- Steam - Registration form

- MEO - Registration form
- NBA - Registration form
- Bible Gateway - Contact form
- Cricbuzz - Feedback form
- Amazon - Registration form
- Staples Portugal - Registration form
- IPMA - Contact form
- CTT - Postal Code Search Form
- RedditAds - Registration form

Pages Excluded Due to Page-related constraints

- **WebMD** - Doctor profile contact form. Initially functional, but changes to the pages blocked automated interaction on the page.
- **iLovePDF** - Contact form. Initially functional, but stopped being able to catch mutations due to page changing.
- **Fandango** - Contact form. Initially functional, but stopped being able to catch mutations due to page changing.

Pages Excluded Due to LLM-related Constraints: These cases are tied to the model's inability to correctly identify or select the relevant elements in the form, which prevented the pipeline from interacting as intended. At this stage, all tests were conducted using Mistral's Large model[1] (version `mistral-large-2411`), which was responsible for identifying form fields and submit buttons. This detail is important, as some observed failures were directly tied to the model's limitations in interpreting form structures.

- Wrong or blocked button selection:
  - **Medical News Today** - BMI calculator.
  - **Reddit Business** - Contact form.
  - **CTT** - Station Search Form.
- Pages Where Fields/Buttons Were Not Identified
  - **AccuWeather** - Contact form.
  - **Trustpilot Business** - Signup Form.
  - **Continente Checkout** - Delivery form.

[1] `https://docs.mistral.ai/getting-started/models/models_overview/`

Pages Excluded Due to Page-related Constraints: These cases were not caused by the model directly, but by the structure or behaviour of the page, which blocked or prevented the pipeline from capturing accessibility feedback.

- Pages with Captcha or Anti-Bot Checks
  - **Merriam-Webster** - Registration Form.
  - **Discord Support** - Submit request Form.
- Pages with Blocked Buttons due to Consent or Preconditions
  - **CTT** - Create account.
  - **iStockPhoto Support** - Customer Support Form.
- Pages Without a Form at Load
  - **Quora Business** - Contact form.
  - **Cookpad** - Premium signup.
- Pages Requiring Authentication
  - **IMDb** - Contribution Form.
  - **Etsy** - Onboarding submission.
  - **Genius** - New Submission.

The page testing process revealed not only technical constraints of the pipeline but also the impact of page-level changes over time. Some of the excluded pages, such as WebMD, iLovePDF, and Fandango, were initially functional and produced valid results during early iterations of the pipeline. However, subsequent changes to their structure eventually blocked automated interaction, making them unsuitable for inclusion in the final evaluation subset. This highlights the fragility of working with live production websites, where even minor updates can compromise reproducibility.

An important consequence of these changes was that WebMD, the only page in the initially functional set that presented violations for the error states indicated only by color evaluation, could no longer be included in the final analysis. As a result, the final subset of ten pages contained no such violations. This absence is relevant when interpreting results for this criterion since all drops in accuracy came exclusively from false positives, offering insight into whether models tended to report violations that were not present in the form.

Beyond these specific cases, several insights also emerged regarding the conditions under which the pipeline can operate effectively:

- Dynamic updates and mutation observation are only effective when feedback is shown on the page after interaction. If the submission triggers a full page reload, no accessibility feedback is captured.

- Captcha and anti-bot mechanisms blocked the pipeline from reaching the form interaction stage. While most could be bypassed with the cookie dismissal script, others persisted and prevented access to the form entirely.
- LLM misinterpretations led to errors in interaction, either by selecting the wrong button for submission or by failing to identify form fields altogether.
- Session tokens and expiration events created instability during longer evaluations.

These insights explain why only a subset of the initial list of pages could be included in the final analysis presented in the following sections.

Benchmarking Setup

To measure performance, the pipeline's outputs were compared against a set of manual benchmarks that we created, that specify the expected accessibility outcome for each form field. These benchmarks served as the reference standard when calculating performance. To ensure comparability, the evaluation was framed as a binary classification task. Fields labeled in the benchmark as fail (expected violation) were treated as positive cases, while those labeled as pass (expected no violation) were treated as negative cases. Pipeline predictions of fail were mapped to positives, while predictions of pass, inapplicable, or absent were mapped to negatives. Fields labeled in the benchmark as inapplicable or absent therefore only contribute to the analysis when the model predicts fail, in which case they are counted as false positives. Under this scheme, a true positive occurs when both the benchmark and prediction indicate a violation; a false negative when the benchmark indicates a violation but the prediction does not; a true negative when both benchmark and prediction indicate no violation; and a false positive when the prediction indicates a violation but the benchmark does not.

Evaluation Metrics

Performance was assessed using standard classification metrics: accuracy, precision, recall and F1-score [33]. These metrics were calculated by comparing the pipeline's results directly against the manual benchmarks. In addition to this benchmark-based evaluation, results were also analyzed in two further dimensions:

- Comparison across models running the pipeline with five different language models on five representative webpages, to examine consistency and variation in outcomes.
- Comparison with automated accessibility tools to contextualize the pipeline's strengths and limitations in relation to existing evaluation approaches.

The pipeline itself was set up and executed exactly as described in Chapter 3, combining browser-based interaction, mutation observation, retrieval, and structured LLM evaluation.

4.2 Manual Evaluation Benchmarking

To establish a reliable reference for comparison, we manually evaluated all selected webpages and created benchmarks, one for each accessibility check, covering all expected form fields from the selected pages. These benchmarks specify the expected outcome in alignment with the relevant WCAG success criteria and are labeled as *pass*, *fail*, or *inapplicable*, in the same way that the pipeline evaluation was carried out. They provide the primary reference against which pipeline outputs were measured.

Constructing these benchmarks required detailed inspection of every field under different submission conditions. For required fields and error identification, forms were submitted with empty inputs to verify whether validation feedback was triggered and how it was conveyed. For error clarity and use of color, invalid inputs were introduced to capture the resulting error messages or visual changes, and the benchmarks for these checks were created using the fallback inputs introduced in Section 3.4.4. When running the pipeline evaluation, for error clarity and use of color, the inputs used were suggested by an LLM call, which meant that not every run triggered the same response as different inputs could reveal different messages. To account for this, we cross-checked the inputs logged in each evaluation run and, where necessary, manually reassessed the corresponding fields with the same inputs the model had used.

Once constructed, the benchmarks were aligned with the pipeline outputs field by field. This mapping enabled the calculation of the performance metrics: accuracy, precision, recall, F1-score, and coverage; and highlighted divergences between system and human judgments. Later, these divergences were examined to identify error patterns and assess the quality of the model's reasoning.

Both the pipeline outputs and the manual benchmarks were stored in JSON format, which allowed systematic alignment and cross-checking during analysis.

The manual benchmarks serve two purposes in the evaluation: first, as the primary reference for calculating performance metrics, and second, as a stable baseline for comparing outputs across different language models and against automated accessibility tools.

Table 4.1 summarizes these benchmarks. Each cell is divided into three categories: **'P'** (positive cases, where violations were identified), **'N'** (negative cases, where no violations were found), and **'I'** (inapplicable cases, where the issue did not apply to the given field). These categories combined capture all form fields expected to be evaluated in each page, for each accessibility issue targeted.

4.3 Comparison Across LLM Models

A central part of the evaluation was to test the pipeline across different language models and compare how each one performed under identical conditions. This section reports those comparisons, first outlining the models used, then analyzing performance at the level of individual WCAG criteria, and finally identifying broader patterns in model behaviour.

Table 4.1: Summary of Benchmark Annotations with Positive, Negative and Inapplicable Cases for each Form and Accessibility Issue

	Missing Programmatic Cues – Required	Failure to Identify Errors or Invalid Identification	Ambiguous or Generic Error Messages	Error States Indicated Only by Color
MEO - Registration form	P:4 — N:0 — I:2	P:4 — N:0 — I:2	P:1 — N:1 — I:4	P:0 — N:3 — I:3
Steam - Registration form	P:4 — N:0 — I:0	P:3 — N:- — I:1	P:1 — N:1 — I:2	P:0 — N:2 — I:2
NBA - Registration form	P:1 — N:0 — I:2	P:1 — N:0 — I:2	P:1 — N:0 — I:2	P:0 — N:1 — I:2
Bible Gateway - Contact form	P:0 — N:4 — I:1	P:0 — N:4 — I:1	P:1 — N:3 — I:1	P:0 — N:3 — I:2
Cricbuzz - Feedback form	P:0 — N:4 — I:2	P:0 — N:5 — I:1	P:2 — N:2 — I:2	P:0 — N:4 — I:2
Staples - Registration form	P:4 — N:0 — I:2	P:4 — N:0 — I:2	P:1 — N:1 — I:4	P:0 — N:2 — I:4
Amazon - Registration form	P:0 — N:4 — I:0	P:1 — N:3 — I:0	P:1 — N:2 — I:1	P:0 — N:3 — I:1
RedditAds - Registration form	P:1 — N:0 — I:1	P:1 — N:0 — I:1	P:0 — N:1 — I:1	P:0 — N:1 — I:1
CTT - Postal Code Search Form	P:0 — N:1 — I:4	P:0 — N:1 — I:4	P:0 — N:1 — I:4	P:0 — N:1 — I:4
IPMA - Contact form	P:0 — N:3 — I:4	P:3 — N:0 — I:4	P:1 — N:0 — I:6	P:0 — N:1 — I:6
Total	P:14 — N:16 — I:18	P:17 — N:13 — I:18	P:9 — N:12 — I:27	P:0 — N:21 — I:27

4.3.1 Models and Evaluation Setup

The model comparison was carried out under identical conditions for every language model. The same pipeline configuration, prompts, retrieval mechanisms, and evaluation signatures were applied in all runs. The only distinction was on the timing that results for GPT-5 were gathered, due to it being released after the other results were consolidated, but both the setup and the webpages targeted remained unchanged. This ensures that any differences reported in this section reflect model behaviour rather than changes in the evaluation procedure.

The evaluation covered six models in total: Mistral Large, GPT-4o, GPT-4o mini, GPT-4.1, GPT-4.1 mini, and GPT-5. For each model we performed one run for each page evaluated. We selected five pages from the full subset introduced in Section 4.1. This reduced set was necessary to keep the multi-model comparison tractable, while also ensuring that the evaluation remained representative. The five pages were selected because they capture diverse structures and validation behaviours, and because they remained stable throughout the evaluation period:

- Steam registration form -`https://store.steampowered.com/join/`
- Bible Gateway contact form - `https://www.infinite.media/bible-gateway/`
- Cricbuzz contact form - `https://www.cricbuzz.com/info/contact`
- Staples Portugal registration form - `https://www.staples.pt/pt/pt/registo`
- CTT postal code search form `https://www.ctt.pt/feapl_2/app/open/postalCodeSearch/postalCodeSearch.jspx?lang=def#fndtn-postalCodeSearchPanel`

By standardizing both the pipeline setup and the page selection, the comparison isolates differences in how accessibility issues were detected and reasoned about at the model level.

4.3.2 Per-criterion accuracy across models

The results presented in table 4.2 highlight clear differences across models. The strongest overall performance was clearly achieved by GPT-4o that achieved above 90% accuracy in three of

Table 4.2: Accuracy per-issue across models.

Accessibility Issue	Mistral Large	GPT-4o	GPT-4o mini	GPT-4.1	GPT-4.1 mini	GPT-5
Required fields	75.0*	94.7**	57.7	87.5	76.9	**95.7****
Error identification	70.6	**100**	47.6	82.4	40.0**	76.9*
Error clarity	83.3	**90.9****	55.6	85.7	56.2	76.9
Color-only error states	81.8**	81.8**	43.8	81.8**	33.3	80.0**
Average	77.7	**91.9**	51.2	84.4	51.6	82.4

* All errors concentrated on 2 pages.
** All errors concentrated on a single page.

the four checks, and, particularly, for the error identification evaluation it identified every issue without introducing false positives. Surprisingly, the new flagship model GPT-5 only excelled in the required field evaluation (95.7%), but was more inconsistent across the other checks. GPT-4.1 also performed consistently well across all evaluations, but without the same quality of GPT-4o. Mistral Large also performed fairly well, achieving the most accuracy while evaluating the error clarity accessibility, even outscoring GPT-5, but for the other issues it didnt perform as well as the OpenAi's models. By contrast, the mini models (4o mini, 4.1 mini) consistently underperformed across all checks. These variants are explicitly designed to be lighter-weight, with smaller parameter counts and reduced inference-time compute, making them faster and cheaper to deploy than their full-sized counterparts, and this difference reflected well within the context of the evaluations performed.

For the color-only error state criterion, results require more careful interpretation. The CTT page could not be evaluated for this check because the pipeline reached the token limit. This only occurred for this specific check, since its mutations were larger than in the others. As noted earlier, none of the pages in the final selection contained actual violations for this check, meaning that the drop in accuracy comes only from false positives, and across Mistral Large, GPT-4o, GPT-4.1, and GPT-5 the same 2 false positives, that occured on the Steam page, were the only drop in accuracy. The asterisks in Table 4.2 reflect this concentration of errors on just one or two pages, suggesting that these results point more to pipeline limitations in mutation capture or field extraction than to differences in model behaviour, especially when the issues lie on one page only.

When averaged across all criteria, GPT-4o achieved the highest accuracy (91.9%), followed by GPT-4.1 (84.4%), GPT-5 (82.4%), and Mistral Large (77.7%). The mini variants, by contrast, both scored close to 50% on average, a gap that underscores the limitations of reduced-capacity models in this evaluation setup.

Given this distribution, GPT-4o and GPT-5 were selected for the full evaluation across the entire page set. GPT-4o was chosen because it combined strong overall accuracy with low rates of false positives. GPT-5, released during the consolidation of results, was included to provide a direct comparison with the newest flagship model under identical conditions. Importantly, the page-level runs used in this model comparison are the same as those employed in the subsequent

full evaluation, ensuring methodological consistency across analysis.

4.4 Quantitative Results

The following section presents the quantitative results for the evaluation of the models selected (GPT-4o and GPT-5) across the full set of pages, providing a detailed view of how the pipeline performed. Similarly to the evaluations carried out earlier, we performed a single run per page for each model, but this time across ten pages instead of only five. This broader setup provides a more comprehensive view of performance and makes it possible to check whether the patterns observed in the smaller comparison hold consistently when applied to a larger set of pages. All the different elements of the pipeline also remained the same, as they represent the final state of the pipeline after the refinements made throughout development.

The analysis begins at the field level, reporting the accuracy of the pipeline in detecting accessibility issues relative to the manual benchmark. Results are then broken down by WCAG criterion to highlight which checks were handled most reliably and which remained more challenging. Performance is also aggregated at the page level, showing how accuracy varied across different forms and site structures. Finally, common error patterns are analyzed by examining false positives and false negatives, which helps identify the circumstances under which the pipeline struggled most.

For each accessibility issue a table is provided with the classification metrics, along with the amount of false positives (FPs), true positives (TPs), false negatives (FNs) and true negatives (TNs), and also the number of fields evaluated, excluding inapplicable fields for the specific accessibility issue under evaluation, across all pages (Samples). These absolute results were included to allow a direct comparison with our manual evaluations (Table 4.1), since the models sometimes skipped fields, combined multiple fields into one evaluation, or analyzed additional fields beyond those expected.

4.4.1 Missing Programmatic Cues – Required

Table 4.3: Performance on Missing Programmatic Cues – Required

Model	Accuracy	Precision	Recall	F1-score	TPs	FPs	TNs	FNs	Samples
GPT-4o	0.946	0.929	0.929	0.929	13	1	22	1	37
GPT-5	0.972	1.000	0.929	0.963	13	0	22	1	36

Both models performed similarly while evaluating the use of the required attribute on form fields, achieving accuracy scores well above 90%, as presented in table 4.3. The models both produced a single wrong assessment for `<select>` elements, where the presence of a required attribute can be ambiguous depending on how browsers interpret the default option. GPT-4o reported a false positve on the `<select>` element of the MEO form and, on the other hand, GPT-5 missed a violation of the same element but on the Steam form. In addition, GPT-4o missed a

required checkbox on the Staples registration form, a case that highlights the challenge of recognizing non-textual cues.

These results suggest that both models are highly reliable at detecting the correct use of the required attribute on form fields, particularly when the `required` attribute or validation messages are explicit after submitting the empty form. However, they also underline the difficulty of edge cases such as dropdowns and checkboxes, where the semantics can be less clear-cut and the boundary between required and optional behaviour may depend on implementation details. Overall these results suggest that required field evaluation represents one of the strongest areas of performance for the pipeline.

4.4.2 Failure to Identify Errors or Invalid Identification

Table 4.4: Failure to Identify Errors or Invalid Identification

Model	Accuracy	Precision	Recall	F1-score	TPs	FPs	TNs	FNs	Samples
GPT-4o	0.864	0.913	0.889	0.889	8	1	13	1	23
GPT-5	0.742	0.696	0.941	0.800	16	7	7	1	31

On this accessibility check, presented in table 4.4, the number of samples varied considerably between models. This happened due to GPT-4o concatenating multiple fields into a single evaluation for 3 different pages (Staples, Meo and IPMA), which reduced the overall sample count. The evaluation performed by the model simply comprized one evaluation judgement across all fields, and for all three pages they were correct, which also meant that the amount of true positives and negatives to be slightly lower. On the other hand, the opposite happened when using GPT-5, as this model produced repeated evaluations for the same fields for the Staples form on a single run, inflating the number of samples and true positives without increasing the range of cases actually covered. The correct but repeated evaluations of the fields for this page were not included on the metrics calculations, allowing only one field to be represented for this page, which reduced the amount of 'true positives' from 24 to 16 across all pages.

Both models struggled with the structure of outputs for this check, but the source of errors did not come from the pages where this issue lied. GPT-5 reported seven different false positives and a missed violation across 4 pages, while GPT-4o performed much better, producing only a single false positive on the NBA form, and one missed violation on the Amazon page. Overall, GPT-4o did not perform flawlessly as when evaluating the smaller subset reported on table 4.2, but it still proved much more consistent and reliable than GPT-5, which had a tendency of over reporting issues for this evaluation.

4.4.3 Ambiguous or Generic Error Messages

This was one of the more subjective checks, since error clarity depends on whether messages are specific enough to guide the user toward correcting the error. Ambiguous phrases such as

Table 4.5: Performance on Ambiguous or Generic Error Messages.

Model	Accuracy	Precision	Recall	F1-score	TPs	FPs	TNs	FNs	Samples
GPT-4o	0.800	0.667	0.857	0.750	6	3	10	1	20
GPT-5	0.818	0.714	0.714	0.714	5	2	13	2	22

"Invalid input" or prompts that rely on assumed context made this criterion harder to evaluate in a consistent way, as seen in Table 4.5.

For GPT-4o, the main issue lied in reporting false violations. Out of the twenty fields applicable to this evaluation, it produced one case where an inapplicable field was marked as a failure, and two cases where compliant fields were also marked as failures, introducing three false positives, each for a different page. It also missed a violation on the Staples Form, corresponding to one false negative. These errors lowered precision substantially, even though recall remained fairly high.

GPT-5, by contrast, showed a more balanced behaviour. Out of the twenty-two evaluated samples, it produced two false positives and two false negatives. One interesting example, that perfectly highlights the challenge of evaluating error clarity, was the evaluation on Bible Gateway's 'Website' field, which was incorrectly flagged as fail despite being benchmarked as a pass. For this field, the input typed on the field by the pipeline was '!', which raised the error "Website is not valid. Must be a valid URL.", which is an ambiguous case to evaluate. For the manual evaluation for this accessibility issue we considered it as a violation, since it provided a vague suggestion on how to correct, whilst the model considered it sufficient to pass this evaluation. Furthermore, the results illustrate how GPT-5 occasionally over-reported adequate feedback as unclear, and even though its overall tendency was more cautious than GPT-4o, the results were similar.

These results highlight how much interpretation is involved in deciding whether an error message is truly helpful. In the end, both models reveal not just their own strengths and weaknesses, but also how subjective it can be, even for humans, to draw the line between a clear instruction and an ambiguous one.

4.4.4 Error States Indicated Only by Color

Table 4.6: Performance on Error States Indicated Only by Color.

Model	Accuracy	Precision	Recall	F1-score	TPs	FPs	TNs	FNs	Samples
GPT-4o	0.842	-	-	-	0	3	16	0	19
GPT-5	0.842	-	-	-	0	3	16	0	19

For the color-only error state check, presented in table 4.6, the pipeline successfully captured style changes such as background or border colors. However, none of the evaluated pages in the subset actually contained violations for this check. As a result, the models had no opportunity to generate false positives, meaning that precision, recall, and F-measure could not be meaningfully

computed.

While this makes the check weak in terms of benchmarking, we chose to include its results as it still provides useful insights. In this evaluation, both models had identical performance, raising three false positives each, and these happened on the same exact fields for both. When analyzing the results, the reasoning provided by the models suggested that the pipeline could not fully capture the mutations that occured. For example, in the Reddit email field, an error message is shown in text and the color of the border changes, but the reasoning provide was 'Only visual class/style changes occurred; no aria-invalid or linked error message. The error is indicated by visual styling alone'. As this happened for both models, it is likely that for these two pages the mutations could not be accurately captured for this check. Despite these two cases, both models performed flawlessly on the remaining pages.

4.4.5 Page-Level Accuracy Across Checks

To complement the field-level analysis, performance was also aggregated by page, broken down by accessibility check. Table 4.7 reports accuracy values for GPT-4o and GPT-5 across the ten webpages, showing how results varied depending on both the form structure and the type of accessibility requirement.

Table 4.7: Page-level accuracy across the four accessibility checks.

Page	**Required Fields**		**Error Identification**		**Error Clarity**		**Use of Color**	
	GPT-4o	GPT-5	GPT-4o	GPT-5	GPT-4o	GPT-5	GPT-4o	GPT-5
Steam (Registration)	100%	75%	75%	100%	100%	50%	0%	0%
MEO (Registration)	80%	100%	100%	100%	50%	100%	100%	N/A
NBA (Registration)	100%	100%	50%	100%	100%	100%	N/A	100%
Cricbuzz (Feedback)	100%	100%	100%	0%	100%	100%	100%	100%
Reddit (Registration)	100%	100%	100%	0%	0%	100%	0%	0%
Amazon (Registration)	100%	100%	50%	75%	100%	100%	100%	100%
Staples (Registration)	80%	N/A	100%	100%	50%	100%	100%	100%
IPMA (Contact)	N/A	100%	100%	100%	50%	100%	100%	100%
CTT (Postal Code Search)	N/A	N/A	100%	N/A	N/A	0%	TL	TL
Bible Gateway (Contact)	100%	100%	100%	50%	N/A	100%	100%	100%

TL - model reached token limit for evaluation
N/A - no applicable fields on this evaluation

This representation makes it possible to compare performance across pages while also distinguishing which accessibility checks were more or less challenging. An important detail is that when one model shows N/A and the other reports an accuracy value, this is often due to how true negatives were treated. As mentioned previously, we ignored cases where an inapplicable evaluation was matched against a pass benchmark, but we did not ignore a pass evaluation matched against a pass benchmark.

At the page level, accuracy tended to be higher on forms with more explicit validation rules, such as those found on Amazon and Staples, where the model could rely on a larger number of fields for evaluation. However, for pages with fewer fields, like Reddit and NBA, the results were

more volatile. A single misclassification could cause significant swings in accuracy, leading to extreme variations between 0% and 100%. This variability means that accuracy alone may not be the best indicator of overall performance on these pages, as the limited number of fields can make results more sensitive to errors. To gain a more comprehensive understanding of the model's performance, it's important to consider not only the final accuracy but also the reasoning provided for each evaluation. This deeper analysis helps contextualize the results and will be explored further in the next section.

4.4.6 Analysis of the LLM's evaluation reasoning

Beyond raw accuracy, it is important to analyze the reasoning outputs provided by the models. The reasoning field in each output often reveals whether the model not only reached the correct classification but also understood the accessibility principle at stake. This analysis is critical, since developers and auditors depend on clear explanations to act on reported issues.

To analyze this aspect of the results gathered, we went through the evaluations performed using GPT-4o and GPT-5 across the ten webpages, identifying cases where the reasoning was misaligned, either with the evaluation for the same fields or with general guidelines. In most cases, the models produced justifications that were both relevant to the field under evaluation and well aligned with the evaluation rules. With the output rule requesting the model to produce concise but detailed reasoning focused on the specific issue, nearly all outputs followed this format successfully, but there were some particular cases of where the reasoning provided by the model was not entirely aligned with the evaluation result.

Misaligned reasoning

The examples of misalignment between the evaluation result and the reasoning statement present in the results represent rare occurrences, but remain important to highlight as potential sources of problems. These examples took 3 different formats: (1) the evaluation result from the pipeline aligned with the benchmarks but the reasoning statement was incorrect or out-of scope for the target evaluation, targeting different issues than the one intended; (2) the evaluation result from the pipeline differed from the benchmarks but the reasoning statement was aligned with the expected result, but also evaluated issues out-of-scope for the target evaluation; (3) the evaluation result from the pipeline and the reasoning statement were contradictory.

Their distribution is worth noting, as formats (1) and (2) only appeared during the error clarity evaluation. For format (1), we observed a single instance with GPT-5 and two instances with GPT-4o. These amounted to what could be described as "false true positives", where the evaluation result itself matched the benchmark but the reasoning pointed to the wrong issue. For example, on the NBA signup page GPT-4o marked the email field as a violation, which aligned with the benchmark but its reasoning was: "The error message (...) clearly explains the problem (...) However, the field lacks a visible label, which is a WCAG 3.3.2 violation", in other words it justified the result using an unrelated criterion.

Format (2) occurred three times with GPT-4o, all linked to false positive cases. Here the model gave a reasoning that sounded aligned but evaluated the wrong aspect of accessibility. For instance, on the IPMA form it flagged the message field with: "The empty input did not trigger a validation error or message. As a required field, it should have prompted an error message (...)" While coherent, this reasoning targeted the presence of required-field validation rather than the clarity of the error message, which was out of scope for this check.

By contrast, format (3) only occured once, and it was found in the required-field evaluation with GPT-5, where the evaluation result and reasoning directly contradicted each other. On the CTT form, GPT-5 classified the locality field as "inapplicable" but simultaneously reasoned: "Shows required behavior (...) but lacks required or aria-required", implicitly treating the field as a failure. Importantly, this unique case did not affect overall performance metrics, whereas formats (1) and (2) did contribute to errors in the error clarity check.

Suplementary observations in reasoning

Although the misalignment cases described above were rare, we also observed instances where both models raised additional points that went beyond the scope of the specific evaluation but did not conflict with the result. In these cases, the evaluation remained correctly focused on the intended criterion, but the reasoning included side notes about related accessibility considerations. For example, on the Staples form, GPT-5 evaluated the email field for use of color in error states as a pass, and its reasoning message noted that, while the field was compliant for the target issue, it noted that no programmatic association was in place for this error message: "A visible alert error message was displayed for the email field, providing textual feedback; therefore the error is not indicated by color alone (programmatic association not specified)." Here, the model correctly judged that the error was not conveyed by color alone and kept the evaluation result focused on the target issue.

Model differences

The level of detail and consistency of the reasoning outputs also varied between models. GPT-5 outputs tended to be more concise and aligned with the prompt, while GPT-4o often produced longer explanations that occasionally drifted beyond the requested scope. Unlike the previous examples, which highlighted cases of scope drift or mismatched reasoning, the following illustrate two strong outputs, one from each model. Both refer to the `Name` field on the Cricbuzz page during the evaluation of error clarity, as it was the one that introduced most issues. In this case, both GPT-4o and GPT-5 correctly classified the vague error message as a failure, but their reasoning styles differed noticeably:

> **GPT-4o:** `The input '12345' triggered a validation error with the message 'This is not a valid Name.' This message indicates the input does not match the expected pattern, but it does not provide guidance on what constitutes a valid name. The message could be improved by specifying the expected format (e.g., letters only).`

GPT-5: `"This is not a valid Name." is vague and does not provide corrective guidance (e.g., allowed characters or length).`

While both explanations reached the same conclusion, GPT-4o tended to elaborate more broadly, often introducing more details, whereas GPT-5 gave shorter, more focused reasoning that aligned more closely with the evaluation scope. These differences, though subtle in correct cases, became more pronounced in ambiguous or incorrect evaluations, where GPT-5 generally maintained tighter alignment with the intended task.

Practical considerations for reasoning quality

These results show that usefulness comes not only from achieving the correct final result but also from keeping the reasoning focused on the intended check. Short, targeted explanations make the outcome easier to act on, while drifting into other accessibility issues can create noise and reduce clarity. Even when the model's evaluation matches the benchmark, reasoning that goes out of scope weakens trust in the evaluation. Since our pipeline and prompts were designed to test one accessibility check at a time, this kind of drift, despite being a rare occurrence, is a clear limitation. For LLM-based accessibility evaluation to be reliable in practice, outputs need to balance accuracy with reasoning that stays aligned with the specific check under review.

4.4.7 Comparison with Automated Tools

To contextualize the performance of the LLM-based pipeline, it is important to compare it with existing automated accessibility evaluation tools. For this purpose, we tested the same pages using QualWeb[2] and axe DevTools[3] automated checks, focusing in particular on those where the pipeline had produced false positives. This allowed us to examine whether the tools could identify the same errors or provide comparable feedback.

The results achieved highlight a clear difference in scope. With QualWeb, the automatic evaluation reports did not capture any of the errors that our pipeline was designed to test. For example, no rules were triggered for required field checks or use of color, and in the case of error identification and error clarity, the tool issued only generic warnings such as:

```
Error messages describe invalid form field value.  Check that text
error messages provided identify the cause of the error or how to
fix the error.
```

While this aligns with WCAG success criterion 3.3.1 (Error Identification), it falls short of evaluating the specific interaction-based checks carried out in our pipeline. The warning essentially delegates responsibility back to the evaluator, without confirming whether the error message is actually clear or actionable.

A similar limitation appeared with axe DevTools. Using its automated evaluation through browser devtools, none of the target errors were detected. To go further, we ran the "intelligent

[2] https://github.com/qualweb
[3] https://www.deque.com/axe/devtools/

guided tests," which are semi-automated. These allow field-level evaluation: for each form input, the tool prompts the user to perform interactions such as submitting the form empty or with invalid input, and then asks a sequence of questions about the observed behaviour. This setup is closer to our pipeline, since it works at the level of individual fields and requires simulating interactions. However, even when answering the questions accurately for fields that failed required, error identification, or error clarity checks, the tool condensed the outcome into a single generic violation for that field. For example:

```
Additional instructions are needed for the input field but are not
provided for people with disabilities.  People without disabilities
have access to additional instructions.
```

This means that although axe DevTools is able to evaluate fields individually, the result did not differentiate between the specific issues present. Instead, it generalized them into one single warning, which does not reflect the range of accessibility failures captured by our pipeline, such as vague or missing error messages.

Strengths and limitations of both approaches.

The comparison made here focused on the specific accessibility checks we developed for the pipeline, and one of the reasons these issues were chosen was because existing automated tools could not cover them. Nonetheless, automated tools such as the ones we put to the test, excel in rule-based checks that can be verified directly from markup, like missing labels or invalid ARIA attributes. They scale efficiently and provide consistent reports across entire pages, which makes them reliable for broad compliance testing. However, they cannot reliably evaluate interaction-dependent checks, such as required field handling, error clarity, or feedback that only appears after form submission. When they do attempt these, the results are usually reduced to generic warnings that shift responsibility back to the evaluator.

Our pipeline, in contrast, was able to simulate form interactions and reason about the resulting error states at the level of individual fields. This allowed it to capture problems that automated tools did not detect at all, such as vague error messages or reliance on color alone. At the same time, it remains limited by model variability, prompt sensitivity, and dependence on correct DOM capture. It also lacks the breadth of static, rule-based checks that automated tools handle so effectively.

In short, this comparison reveals that automated tools and our pipeline address different dimensions of accessibility testing: the former ensure reliable detection of markup-level violations, while the latter extends evaluation into areas requiring reasoning and interaction. With these distinctions established, the next step is to consider the validity and limitations of our results, examining how the evaluation setup, benchmarks, and pipeline constraints affect both the reliability and the generalization of the findings.

Chapter 5

Discussion

5.1 Reflections on Research Questions

This section revisits the research questions defined in Chapter 1, interpreting the results obtained in Chapter 4 in relation to the objectives of our work.

RQ1: To what extent can LLMs accurately identify and evaluate context-dependent form-related accessibility issues?

This question refers specifically to the subset of form-related, context-dependent accessibility issues issues selected for this study. The results in Section 4.4 provide a detailed breakdown of model performance at both the field and page level, allowing a clearer interpretation of which accessibility issues LLMs can most reliably identify under the evaluation setup.

Overall, the strongest performance was observed for the use of the required attribute accessibility check. Both GPT-4o and GPT-5 achieved accuracy around 95%, with GPT-5 reaching 0.972 and perfect precision, meaning no false positives were raised here. Errors for this evaluations came from two edge cases with `<select>` elements on two different pages, where the required status depends on how browsers handle default options and if the form can be submitted with the fields 'empty'. This revealed cases where even human judgment may allow for more than one valid interpretation. Another error was a missed violation in a checkbox on the Staples form, in the case of GPT-4o. These findings suggest that, with the support of the pipeline developed, LLMs can be automated to evaluate the correct use of the required attribute, with the only identified frailties being reduced to ambiguous cases. These results also indicate that a similar method of evaluation is feasible for the use of other programmatic cues such as read-only and disabled, that are less prominent in forms but relevant in the context of an accessible page.

The check for error identification showed a clear trade-off between the two models. GPT-4o achieved higher accuracy (0.864) and precision (0.913), with only one false positive and two false negatives across 23 applicable fields. Its main errors involved misclassifications, such as marking a field as inapplicable when it should have been flagged as a violation, or overlooking violations when no label was programmatically associated. By contrast, GPT-5 excelled in recall (0.941), successfully capturing almost all genuine violations but at the cost of introducing many

false positives. Out of 31 samples, GPT-5 misclassified seven accessible fields as failures, which reduced its precision to 0.774. At the same time, it missed only one genuine violation, showing strong sensitivity but weaker discrimination compared to GPT-4o. Taken together, these results suggest that GPT-4o provided a more reliable evaluation, while GPT-5 leaned toward broader coverage, ensuring that few violations were overlooked but increasing the rate of false alarms, which is a major concern for accessibility evaluation.

The most subjective issue to evaluate was error clarity, where judgments are based on assessments of whether error messages are sufficient to guide users in completing the form. Despite this, both models performed reasonably well. GPT-4o only missed one violation, regarding the generic error message presented for the email field on the Staples form, but it also introduced a substantial number of false positives, reporting valid passes and inapplicable cases as violations. This resulted in high recall but low precision, highlighting a greater struggle for this model when compared with previous checks. On the other hand GPT-5 reduced false positives substantially, reaching precision of 0.833 while maintaining perfect recall. It still misclassified some adequate messages, but as mentioned in Section 4.4, the reasoning provided by GPT-5 was often more aligned with human judgment. While both models were sensitive to unclear error messages and didn't particularly excel for this accessibility evaluation, GPT-5 still demonstrated superior discrimination, indicating progress toward reliable evaluation of this more subjective evaluation.

For the check of error states indicated by color only, both models achieved the same performance, with issues raised by both models on the same pages (Steam and Reddit). For these pages the models could not capture the error messages correctly, most likely due to the substantially larger mutation context for this evaluation, while both identified that changes in color happened, meaning the ability of correctly identifying these changes. However, since the webpage subset contained no genuine violations, precision, recall, and F1-scores could not be meaningfully computed. Despite this, the absence of false positives on the remaining pages suggests that the models did not over-report stylistic changes as accessibility violations, even when color changes such as borders or backgrounds were detected. This accessibility check seemed promising, but further validation on pages with known violations will be necessary to establish reliability.

In summary, the clearest strengths of both models emerged in the detection of required fields, where accuracy was consistently high and errors were limited to marginal edge cases. For error identification, GPT-4o proved the more reliable option, with steadier precision and fewer false alarms, while GPT-5 favored broader coverage, capturing nearly all violations but at the cost of high false positive rates, proving unreliable. Error clarity remained the most challenging of the tested checks, yet GPT-5 showed greater promise here than GPT-4o, benefiting from the reasoning nature of the task and producing fewer false alarms while still detecting most unclear messages. By contrast, GPT-4o struggled with over-reporting. Finally, the evaluation of error states indicated only by color could not be meaningfully validated within this study, since no genuine violations were present in the dataset; still, both models showed restraint in not over-flagging purely stylistic changes. Taken together, these results indicate that required-field detection is the most reliable area

for automation, GPT-4o offers stronger dependability for error identification, and GPT-5 provides early evidence of added value for inherently subjective checks such as error clarity.

RQ2: How valid and reliable is the rationale produced by the LLMs for evaluating form accessibility?

This research question examines whether the rationale produced by the models provided valid and reliable support for evaluating form accessibility, both in terms of alignment with the targeted checks and consistency across cases. In general, the reasoning aligned with the evaluation for the targeted criterion, with only a small number of misalignments observed. As described in Section 4.4.6, these misalignments followed three formats and represented only a small number of cases, but they remain important since they show that reasoning can still drift outside the scope of the intended task. Notably, this occurred despite the use of retrieval grounding and fine-tuning, which provided models with WCAG context and task-specific examples, and also despite the prompts explicitly instructing them to focus on the target check. GPT-4o was more prone to this kind of scope drift, whereas GPT-5 produced more focused and balanced reasoning.

The comparison between models highlights differences in how reasoning was expressed. GPT-4o generally performed well for less subjective criteria such as required fields and error identification, but its explanations more often included out-of-scope elements, and in some cases repeated similar justifications across fields on the same page without adding detail. By contrast, GPT-5, even when slightly less accurate in classification, provided reasoning that was more consistently aligned with the intended evaluation. This was particularly visible for error clarity, where GPT-5's explanations were more balanced, capturing the nuances of ambiguous feedback without overextending to unrelated issues. In error identification, GPT-5 also produced more granular, field-level insights, distinguishing between individual fields instead of grouping them together under a single explanation.

Overall, the results suggest that LLM-generated reasoning can support accessibility evaluation within the defined scope, but this reliability should be taken with caution. GPT-5 demonstrated a greater ability to remain on target and provide detailed insights, while GPT-4o showed stronger results in more objective checks but introduced more misaligned reasoning. These findings indicate that reasoning quality improves with larger, more recent models, yet even with retrieval and fine-tuning in place, occasional drift in scope remains a limitation that evaluators need to account for.

RQ3: What limitations emerge when using LLMs for accessibility evaluation, and what are the main sources of errors observed in their outputs?

The evaluation highlighted that while LLMs can provide valuable support in assessing form accessibility, their use also revealed several limitations. Some of these arise from the pipeline itself: when element extraction or DOM mutation capture failed, the models lacked the necessary context to make reliable judgments. These dependencies show that model performance cannot be

fully separated from the stability of the surrounding system.

At the model level, two recurring challenges were observed. First, contextual sensitivity meant that small differences in prompts or input context occasionally led to divergent outputs, especially in earlier iterations of the pipeline. Retrieval and fine-tuning, combined with more structured prompts, helped reduce this variability, but did not eliminate it fully. This aligns with prior studies that found persistent instability when relying on prompting alone, suggesting that grounding strategies are essential but still leave some inconsistency [4, 13]. Second, model size had a clear impact on reliability as smaller models often missed straightforward issues, while larger ones aligned more closely with human benchmarks but still produced occasional mismatches.

Error patterns further illustrate these limitations. GPT-5 tended to over-report violations, especially for the error identification evaluation, leading to false positives, whereas GPT-4o occasionally overlooked genuine failures, producing false negatives. In both models, there were also cases where the evaluation result matched the benchmark but the reasoning was misaligned, pointing to the wrong criterion or to an out-of-scope issue. These patterns account for most of the discrepancies observed in performance metrics.

Finally, broader constraints also shaped the results. Evaluations of error clarity were inherently subjective, in some cases, what the model flagged as unclear was considered acceptable by our manual evaluation, showing that differences sometimes can come from interpretation rather than actual mistakes. The group of webpages targeted to our evaluation also limited analysis in certain areas, as none of the tested pages contained real violations for error states indicated only by color. Moreover, the evaluation was carried out on the same webpages that were initially used to design and refine the pipeline, which raises the risk of tuning to dataset-specific characteristics.

Taken together, these findings show that the main limitations of LLM-based evaluation stem from technical dependencies in the pipeline, variability in model reasoning, and systematic error patterns. These limitations do not rule out the use of LLMs for accessibility evaluation, but they show that their outputs need to be interpreted carefully and not taken as definitive without human review.

RQ4: How do LLM-based evaluations compare with existing automated tools, and in what ways can they complement these tools when assessing form accessibility?

The results indicate that LLMs extend the coverage of automated tools by addressing their blind spots, rather than replacing them. Rule-based engines such as QualWeb or aXe excel at providing scalable coverage of structural criteria, reliably detecting missing attributes, syntactic violations, or contrast failures [3]. However, when applied to more nuanced criteria such as SC 3.3.1 Error Identification or SC 3.3.3 Error Suggestion, these tools are either not able to flag issues or collapse them into generic warnings. For example, in the evaluated forms, automated tools reported a broad error on ambiguous feedback, whereas the LLM pipeline provided more detailed distinctions, such as identifying when error messages were unclear. This difference illustrates the added value of LLM reasoning: rather than simply indicating that “an error exists,” the model could ex-

plain whether it was communicated with sufficient clarity, a task beyond the scope of rule-based checking.

This complementarity suggests a hybrid model of accessibility evaluation, where automated tools provide the baseline coverage of objective checks and LLMs extend the analysis to contextual, semantic, and user-facing aspects of accessibility. Such integration would reduce the burden of manual audits, which currently fill this gap, while preserving the reproducibility and efficiency of automated workflows. Although semi-automated guided tests already exist in tools like aXe, they still rely on human evaluators to interpret results. By contrast, LLMs can provide first-order judgments on clarity and error communication, narrowing the scope of what requires human confirmation. As such, LLM-based evaluations should be understood as a middle layer between automation and manual review, able to extend evaluative depth without eliminating the need for human oversight. At the same time, the tendency of some models to over-report violations highlights the risk that without careful integration, LLMs could introduce noise that undermines the efficiency gains of automation.

Taken together, existing tools remain the most reliable option for structural and rule-based checks, while LLMs can contribute added value for some contextual and user-facing aspects. Their complementarity lies not in replacement but in extending coverage into areas where automated tools alone fall short, aiming for the reduction on the reliance in manual testing.

5.2 Summary of Key Contributions

This work presents several contributions to the field of accessibility evaluation, particularly in addressing web form issues that are unexplored in current automated tools.

One of the primary contributions is the development of an innovative LLM-based evaluation pipeline specifically designed for form accessibility compliance at the field level. Unlike static, rule-based tools, this pipeline incorporates live interactions with forms, observes DOM mutations, and evaluates accessibility criteria that depend on user interactions, such as error handling and required field validation. This dynamic evaluation process captures context-dependent accessibility issues that traditional tools often miss.

Adapting the pipeline for accessibility tests beyond forms is a realistic possibility, requiring targeted updates to its current components, which are optimized for form interactions. The mutation observer and extracted elements can be redirected to capture different interface components and state changes, while trainset examples, prompt design, and dataset entries can be updated to guide judgments for new criteria. For interaction-dependent evaluations, the logic currently tied to the submit button would need remapping to the relevant interactive element in focus, with corresponding changes to the complementary LLM call. At the same time, it is also straightforward to strip away the interaction layer and use the pipeline in a static evaluation mode, as was done when interaction wasn't possible, although its greatest advantage remains in scenarios where dynamic feedback or user interactions are central, as the pipeline can capture and reason over evidence that static evaluations cannot.

Another key contribution is the integration of retrieval-augmented generation (RAG) and fine-tuning resources, grounded in a dataset of labeled field examples aligned with WCAG criteria. Alongside this dataset, a set of dedicated trainsets was developed for issue-specific prompting, ensuring that LLMs could reliably generate outputs for each form accessibility issue. These resources stabilized evaluation across models, particularly for criteria such as required fields and error identification, where unguided models produced inconsistent or contradictory judgments. Paired with DSPy-based dynamic prompting, this framework reduced sensitivity to formatting variations and enabled more consistent evaluations.

Additionally, this study introduces a mutation observer mechanism for capturing dynamic validation behaviour, recording how errors and field states change upon interaction. Though applied to a limited number of fields in this work, the mechanism provided structured signals, such as attribute changes and injected error messages, that are essential for evaluating criteria dependent on interaction feedback. This complements the dataset and trainsets by ensuring that evaluation can incorporate both static and dynamic evidence.

Finally, the research developed represents one of the first structured explorations of LLMs for WCAG field-level evaluation, systematically combining interaction traces, fine-tuning resources, and retrieval grounding. Unlike prior work, which has focused on isolated criteria or static content, this study demonstrates how LLMs can be integrated into accessibility evaluation workflows. The reusable resources and structured methodology introduced here provide a solid foundation for future research to build upon and extend.

5.3 Insights Learned

One of the most notable outcomes of this work was the ability of models to handle ambiguous or loosely specified error states. They showed strong performance in identifying required fields and in interpreting cases where feedback was unclear or vague. These results indicate that LLMs are particularly effective when evaluation depends on contextual interpretation rather than purely syntactic checks, that are already well covered.

Another key insight was the crucial role of input preparation. Since models have clear limits on the amount of information they can process, passing full page HTML or raw, unfiltered data proved ineffective. Instead, providing focused field-level data, paired with structured formatting, was essential to achieving more consistent evaluations. The quality of the prompt also shaped the results, with clear task descriptions ensuring that models interpreted the context in the intended way.

Finally, early pipeline iterations revealed that hallucinations and inconsistencies were significantly more common when models operated without grounding or structured prompting. The integration of fine-tuning, retrieval-augmented generation, and modular prompting through DSPy reduced these issues, making outputs more stable and reproducible. This progression suggests that stability is not an intrinsic property of LLMs but can be improved through deliberate preparation of context, structured inputs, and grounding strategies.

Together, these insights suggest that while models are not ready to replace either automated testing or manual evaluation, they can meaningfully extend automated evaluations when their use is supported by a focused pipeline design.

5.4 Scalability and Generalization

Scalability remains one of the main limitations to overcome in LLM-based automated web accessibility evaluation. While the proposed pipeline demonstrates a modular design, its ability to scale is still bounded by the efficiency of DOM parsing and the reliability of mutation capture. If these stages fail, such as when JavaScript-rendered states are missed or when dynamic interactions are not properly recorded, the throughput of the entire pipeline is constrained. Furthermore, the use of LLMs, particularly more advanced models, can incur significant computational costs, which can become a bottleneck when evaluating large websites or high volumes of pages. These costs, combined with potential performance limitations in handling complex or dynamic content, impact the scalability of the system. A pipeline feature that potentially adds value toward addressing scalability is the use of structured JSON outputs per field, which include the field's identification, the evaluation result, and the reasoning trace. This format makes outputs more consistent and could support aggregation, caching, or integration into larger workflows.

Beyond scalability, generalization raises equally important challenges. The evaluation conducted in this work was limited to a relatively small set of forms and four accessibility criteria. It remains uncertain how well the pipeline would extend to unseen form structures, more diverse accessibility issues, or different domains of interaction. Interestingly, the evaluation included both Portuguese and English pages, and despite our prompts and fine tuning examples being in English only, the pipeline was able to interpret error messages in both languages without additional adaptation, suggesting that cross-lingual evaluations are feasible. On another note, the current design already shows a path toward generalization within forms as the pipeline covers four distinct types of issues, and the evaluation logic remains the same across them, with only prompts and fine-tuning examples being different for each. This modularity suggests that the approach can extend to other form-related issues with relatively small adjustments. Specifically, it requires creating two new signatures: one for evaluating the chosen accessibility issue and another for the complementary LLM call that retrieves relevant WCAG guidance. These signatures should mirror the structure of those already in place, with output fields adapted to the targeted issue. When invalid inputs are needed, they must be specified in both the evaluation signature and the pipeline's main module, as done for the existing checks. Finally, the workflow also depends on a trainset for fine-tuning, structured to resemble real evaluations and provide the model with concrete, task-specific examples.

Taken together, these observations highlight that scalability and generalization are still open challenges, but the structured design of the pipeline and its modular evaluation strategy provide a foundation to build upon. The ability to generate consistent, per-field outputs and adapt the same workflow across multiple issue types points to a framework that could be extended incrementally.

Rather than standing alone, such an approach may prove most effective when embedded into broader accessibility evaluation workflows, where integration with existing automated tools can balance scalability with contextual depth.

5.5 Integration with Automated Tools

The results of this work suggest that the proposed pipeline should not be seen as a replacement for automated accessibility tools but as a way of extending their coverage. Rule-based engines such as QualWeb or aXe are very effective at detecting structural or attribute-based violations and scale easily across large sites. However, they are limited when the evaluation depends on context. In the evaluations carried in this study, these are precisely the situations where the LLM pipeline was able to provide more meaningful judgments.

With this in mind, a natural next step forward is a hybrid approach between automated tools and LLMs. Automated tools can continue to deliver the broad, objective checks they already handle well, while the reasoning capabilities of the LLM pipeline could be integrated to fill in the gaps where context matters. Combining this reasoning into automated workflows could reduce some need for manual evaluations in accessibility, which are time-consuming and often costly. In this way, the LLM component would act less as a stand-alone evaluator and more as an enhancement tool that gives automated tools the ability to cover criteria they currently miss.

Still, such integration has to be approached carefully. Even when grounded with retrieval and fine-tuning, LLMs can drift in scope or produce broken outputs, and these risks cannot be ignored if the results are to be trusted in automated workflows. Refinement of prompts, datasets, and evaluation logic is necessary before such a system could reliably support large-scale accessibility evalutions. With these precautions, integration becomes an interesting possibility that could shift automated accessibility testing to cover more subjective and interactive criteria.

5.6 Limitations and Practical Considerations

Throughout this work, some limitations have already been mentioned, but it is worth highlighting them explicitly, as they shape how the results should be read and also point toward directions for future research. Limitations are not only constraints but also part of the contribution itself, since they clarify the scope of what has been achieved and what remains to be explored.

The evaluation was based on a small set of pages and only four different accessibility issues related to forms, which means the findings cannot be generalized to all different form types or accessibility contexts, despite them being from real-world. Expanding the scope to additional checks and more diverse form types remains an avenue for future work.

Another relevant aspect is that testing was conducted on a small group of language models, with most of the experiments relying on OpenAI systems. Other models might perform differently, and performance can also shift as models evolve. Newer versions may even outperform the ones tested here, potentially changing the trade-offs between accuracy and cost. A concrete ex-

ample of this challenge is the Mistral Large model. During this study, results were gathered with version 2411, which was accessible through the alias mistral-large-latest. Since then, the alias has been updated to point to a different model (mistral-medium-2508). This version shift means that identical prompts run today may produce outputs that differ from those reported in this thesis, underscoring the importance of noting model versioning when discussing reproducibility.

Scaling the approach in practice also raises challenges. Running evaluations across many fields quickly increases costs, as most current models are not free to use, and response times from the model may not fit easily into development workflows, specifically when fine-tuning the model. Another point to consider is the consistency of outputs. Even with structured prompts and retrieval grounding, the model can drift from the intended task or return broken results. Refinement of prompts, enhanced datasets, or fine-tuning would be needed before the pipeline could be trusted in more demanding evaluation contexts.

Using such a system to support compliance also raises the risk of websites being wrongly validated or flagged, which could create legal and ethical problems. Addressing these concerns requires progress not only in the reliability of outputs but also in the way results are presented and interpreted. Overall, the limitations identified here suggest that LLM-based evaluation should not be seen as a complete solution, but as an emerging approach with clear potential. With further refinement, it can grow into a reliable complement to existing accessibility methods, but for now it remains a step in an ongoing process rather than a final answer.

Chapter 6

Conclusion

This thesis investigated how Large Language Models (LLMs) can support web accessibility evaluation, with a focus on form-related success criteria that often require contextual or interaction-dependent understanding. By designing and implementing a modular DSPy-based pipeline, integrating field extraction, DOM mutation capture, retrieval grounding, and trainsets for fine-tuning, the work demonstrated that LLMs can provide judgments that go beyond rule-based coverage.

Overall, the findings show that LLMs are effective in identifying required fields and error feedback, and are capable of assessing more subjective aspects such as the clarity of error messages. While the evaluation highlighted tangible benefits, it also made evident the limits of current approaches, such as the performance that depends strongly on extraction quality, prompt framing, and model stability. These observations suggest that LLMs may play a valuable complementary role alongside traditional automated tools, though further development is necessary before they can be deployed with confidence at scale.

6.1 Future Work

Looking ahead, several directions emerge from the findings and limitations of this work. Expanding the scope of evaluation to include a broader set of WCAG criteria and collecting a larger, more diverse dataset would help improve the robustness and generalizability of the pipeline.

Strengthening the extraction and interaction layers is also essential, particularly in handling dynamic content, full-page reloads, or feedback that is not embedded directly in the DOM. Another promising avenue is the incorporation of multimodal evidence, where visual cues such as layout, contrast, or color-only indicators could complement DOM-based reasoning. The pipeline could further evolve through integration with established accessibility automated ttools, where LLM-based judgments could be invoked only for cases that require contextual interpretation. Finally, practical concerns such as scalability, cost, and reproducibility must be addressed with strategies like introducing simple caching for repeated structures, experimenting with smaller local retrievers to reduce reliance on remote APIs, and keeping precise records of the models and prompts used would all help make the system more reliable and sustainable in practice.

Final Remarks

This work has shown that LLMs, when carefully guided and embedded in a structured pipeline, can enrich accessibility evaluation by tackling cases that demand contextual understanding and interaction awareness. The pipeline, datasets, and comparative benchmarks presented here establish a foundation for subsequent research and development. While not yet ready to replace existing automated or manual evaluation processes, LLMs have the potential to reduce the burden of nuanced accessibility checks and to contribute meaningfully to more inclusive web experiences.

Bibliography

[1] W3C. W3C - Acessibility. URL https://www.w3.org/mission/accessibility/. Accessed 2024-11-29.

[2] W3C. Web Content Accessibility Guidelines (WCAG) 2.2. URL https://www.w3.org/TR/WCAG22/. Accessed 2024-11-17.

[3] Mahan Tafreshipour, Anmol Deshpande, Forough Mehralian, Iftekhar Ahmed, and Sam Malek. Ma11y: A mutation framework for web accessibility testing. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, page 100–111, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706127. doi: 10.1145/3650212.3652113. URL https://doi.org/10.1145/3650212.3652113.

[4] Juan López and Juanan Pereira. Turning manual web accessibility success criteria into automatic: an llm-based approach. *Universal Access in the Information Society*, pages 1–16, 03 2024. doi: 10.1007/s10209-024-01108-z.

[5] Mirjam Seckler, Silvia Heinz, Javier A. Bargas-Avila, Klaus Opwis, and Alexandre N. Tuch. Designing usable web forms: empirical evaluation of web form improvement guidelines. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, page 1275–1284, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450324731. doi: 10.1145/2556288.2557265. URL https://doi.org/10.1145/2556288.2557265.

[6] Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. Understanding HTML with large language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2803–2821, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.185. URL https://aclanthology.org/2023.findings-emnlp.185.

[7] Sebastian Sergelius. Capabilities of large language models in web accessibility evaluation: A design science approach. Master's thesis, University of Helsinki, 2024.

[8] Somosree Roy. How to avoid False Positives and False Negatives in Testing? https://www.browserstack.com/guide/false-positives-and-false-negatives-in-testing, 2024. Accessed 2025-01-04.

[9] Kristin Fuglerud, Till Halbach, Ingrid Utseth, and Anders Waldeland. *Exploring the Use of AI for Enhanced Accessibility Testing of Web Solutions*, volume 320. 11 2024. ISBN 9781643685526. doi: 10.3233/SHTI241041.

[10] Syed Fatiul Huq, Mahan Tafreshipour, Kate Kalcevich, and Sam Malek. Automated generation of accessibility test reports from recorded user transcripts. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 534–546, Los Alamitos, CA, USA, May 2025. IEEE Computer Society. doi: 10.1109/ICSE55347.2025.00043. URL https://doi.ieeecomputersociety.org/10.1109/ICSE55347.2025.00043.

[11] Carlos Duarte, Miguel Costa, Letícia Pereira, and João Guerreiro. Expanding automated accessibility evaluations: Leveraging large language models for heading-related barriers. pages 39–42, 03 2025. doi: 10.1145/3708557.3716329.

[12] Ziyao He, Syed Fatiul Huq, and Sam Malek. Enhancing web accessibility: Automated detection of issues with generative ai. *Proc. ACM Softw. Eng.*, 2(FSE), June 2025. doi: 10.1145/3729371. URL https://doi.org/10.1145/3729371.

[13] Calista Huang, Alyssa Ma, Suchir Vyasamudri, Eugenie Puype, Sayem Kamal, Juan Belza Garcia, Salar Cheema, and Michael Lutz. Access: Prompt engineering for automated web accessibility violation corrections, 2024. URL https://arxiv.org/abs/2401.16450.

[14] Utkarsha Singh, Jeevithashree Divya Venkatesh, Anujith Muraleedharan, Kamalpreet Singh Saluja, Anamika J H, and Pradipta Biswas. Accessibility analysis of educational websites using wcag 2.0. *Digit. Gov.: Res. Pract.*, 5(3), October 2024. doi: 10.1145/3696318. URL https://doi.org/10.1145/3696318.

[15] Ammar Ahmed, Margarida Fresco, Fredrik Forsberg, and Hallvard Grotli. From code to compliance: Assessing chatgpt's utility in designing an accessible webpage – a case study, 2025. URL https://arxiv.org/abs/2501.03572.

[16] Wafaa Abu Shamlah Hakami and Arwa Yousuf Al-Aama. A framework to improve web form accessibility for the visually impaired. *IEEE Access*, 11:123989–124003, 2023. doi: 10.1109/ACCESS.2023.3330370.

[17] Jonathan Lazar, Abiodun Olalere, and Brian Wentz. Investigating the accessibility and usability of job application web sites for blind users. *J. Usability Studies*, 7(2):68–87, February 2012.

[18] R. Thomas McCoy, Shunyu Yao, Dan Friedman, Mathew D. Hardy, and Thomas L. Griffiths. Embers of autoregression show how large language models are shaped by the problem they are trained to solve. *Proceedings of the National Academy of Sciences*, 121(41): e2322420121, 2024. doi: 10.1073/pnas.2322420121. URL https://www.pnas.org/doi/abs/10.1073/pnas.2322420121.

[19] Andreas Stöffelbauer. How Large Language Models Work, October 2023. URL https://medium.com/data-science-at-microsoft/how-large-language-models-work-91c362f5b78f.

[20] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. Make llm a testing expert: Bringing human-like interaction to mobile gui testing via functionality-aware decisions. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400702174. doi: 10.1145/3597503.3639180. URL https://doi.org/10.1145/3597503.3639180.

[21] Wan Zhang and Jing Zhang. Hallucination mitigation for retrieval-augmented large language models: A review. *Mathematics*, 13(5), 2025. ISSN 2227-7390. doi: 10.3390/math13050856. URL https://www.mdpi.com/2227-7390/13/5/856.

[22] Joshua Gorniak, Yoon Kim, Donglai Wei, and Nam Wook Kim. Vizability: Enhancing chart accessibility with llm-based conversational interaction. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, UIST '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400706288. doi: 10.1145/3654777.3676414. URL https://doi.org/10.1145/3654777.3676414.

[23] Hyunjae Suh, Mahan Tafreshipour, Sam Malek, and Iftekhar Ahmed. Human or llm? a comparative study on accessible code generation capability, 2025. URL https://arxiv.org/abs/2503.15885.

[24] Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting, 2024. URL https://arxiv.org/abs/2310.11324.

[25] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

[26] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather

Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines, 2023. URL https://arxiv.org/abs/2310.03714.

[27] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Yuekai Huang, Jun Hu, and Qing Wang. Unblind text inputs: Predicting hint-text of text input in mobile apps via llm. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703300. doi: 10.1145/3613904.3642939. URL https://doi.org/10.1145/3613904.3642939.

[28] Maryam Taeb, Amanda Swearngin, Eldon Schoop, Ruijia Cheng, Yue Jiang, and Jeffrey Nichols. Axnav: Replaying accessibility tests from natural language. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, page 1–16. ACM, May 2024. doi: 10.1145/3613904.3642777. URL http://dx.doi.org/10.1145/3613904.3642777.

[29] Achraf Othman, Amira Dhouib, and Aljazi Nasser Al Jabor. Fostering websites accessibility: A case study on the use of the large language models chatgpt for automatic remediation. In *Proceedings of the 16th International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '23, page 707–713, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700699. doi: 10.1145/3594806.3596542. URL https://doi.org/10.1145/3594806.3596542.

[30] Giovanni Delnevo, Manuel Andruccioli, and Silvia Mirri. On the interaction with large language models for web accessibility: Implications and challenges. In *2024 IEEE 21st Consumer Communications & Networking Conference (CCNC)*, pages 1–6, 2024. doi: 10.1109/CCNC51664.2024.10454680.

[31] Forough Mehralian, Titus Barik, Jeff Nichols, and Amanda Swearngin. Automated code fix suggestions for accessibility issues in mobile apps, 2024. URL https://arxiv.org/abs/2408.03827.

[32] Giacomo Pedemonte, Maurizio Leotta, and Marina Ribaudo. Improving web accessibility with an llm-based tool: A preliminary evaluation for stem images. *IEEE Access*, 13:107566–107582, 2025. doi: 10.1109/ACCESS.2025.3577519.

[33] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In David E. Losada and Juan M. Fernández-Luna, editors, *Advances in Information Retrieval*, pages 345–359, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31865-1.

[34] Elisa Calì, Tommaso Fulcini, Riccardo Coppola, Lorenzo Laudadio, and Marco Torchiano.

A prototype vs code extension to improve web accessible development, 2025. URL https://arxiv.org/abs/2503.09673.

[35] Wajdi Aljedaani, Abdulrahman Habib, Ahmed Aljohani, Marcelo Eler, and Yunhe Feng. Does chatgpt generate accessible code? investigating accessibility challenges in llm-generated source code. In *Proceedings of the 21st International Web for All Conference*, W4A '24, page 165–176, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400710308. doi: 10.1145/3677846.3677854. URL https://doi.org/10.1145/3677846.3677854.

[36] Sparsh Paliwal, Joshua Hoeflich, J. Bern Jordan, Rajiv Jain, Vlad I. Morariu, Alexa Siu, and Jonathan Lazar. Forma11y—research and development of a tool for remediating pdf forms for accessibility. *ACM Trans. Comput.-Hum. Interact.*, 32(1), April 2025. ISSN 1073-0516. doi: 10.1145/3702317. URL https://doi.org/10.1145/3702317.

Appendix A

Appendix

Table A.1: WCAG 2.2 Success Criteria evaluation automation potential

Success Criteria	**How could a LLM enhance the evaluation?**	**Can the test be automated? (Yes, No, Partially[Why?])**	**Is there any LLM application for its evaluation?**
1.1.1 Non-text Content	Verify if the text alternative is adequate and accurately describe the purpose of the content.	Yes, although nuanced cases might be subjective, in general it may be possible to automate.	**Yes (x2):** **Paper:** *Turning manual web accessibility success criteria into automatic: an LLM-based approach* [4] **Approach:** The authors use LLMs (Google LLM, SalesForce LLM) to automatically describe image content, and GPT-3.5 compares descriptions to Alt attributes to check if they match, flagging violations. **Paper:** *Exploring the Use of AI for Enhanced Accessibility Testing of Web Solutions* [9] **Approach:** The authors use a pre-trained multimodal model (CLIP) to evaluate the descriptiveness of alternative text (Alt text) associated with images. The model calculates a similarity score between the image content and the Alt text, flagging cases where the Alt text is not descriptive enough or irrelevant.
1.2.1 Audio-only and Video-only (Prerecorded)	For audio content, combined with speech-to-text technology, could transcribe the audio and evaluate whether a provided text alternative accurately represents the content.	No/Partially, LLMs can misinterpret complex media content and determining if the alternative provided is enough is also subjective.	No

Success Criteria	How could a LLM enhance the evaluation?	Can the test be automated? (Yes, No, Partially[Why?])	Is there any LLM application for its evaluation?
1.2.2 Captions (Prerecorded)	LLMs can analyze timecodes from the media and captions to determine if captions are properly synchronized with spoken content. Verify if the captions dont obstruct important content.	No/Partially because it can be very costly and subjective.	No
1.2.3 Audio Description or Media Alternative	Verify the quality and accuracy of audio descriptions, ensuring they provide necessary context for visual elements critical to understanding.	Partially because it can be subjective.	No
1.2.4 Captions (Live)	Maybe ensuring synchronization between the captions and the content.	No/Partially, it can be very costly and subjective.	No
1.2.5 Audio Description (Prerecorded)	Verify the quality and accuracy of audio descriptions, ensuring they provide necessary context for visual elements critical to understanding.	Partially because it can be subjective.	No
1.3.1 Info and Relationships	LLMs could identify when relationships are implied visually but not programmatically conveyed. Evaluate whether the content's structure logically conveys the intended relationships like proper heading hierarchy, lists used for enumerations or tables used for tabular data.	Partially, this criterion has many specifications and automating it all could prove very costly, but some aspects are feasible. Difficult to automate in general.	**Not directly, correction only:** **Paper:** *ACCESS: PROMPT ENGINEERING FOR AUTOMATED WEB ACCESSIBILITY VIOLATION CORRECTIONS* [13] **Approach:** GPT-3.5 checked structural HTML elements like ¡section¿ and ¡nav¿ for correct usage to ensure proper information hierarchy and relationships.
1.3.2 Meaningful Sequence	LLMs could analyze text, its structure, and surrounding context to evaluate whether the content is arranged logically for assistive technology users.	No/Partially, can be very subjective.	No
1.3.3 Sensory Characteristics	Detect sensory-dependent instructions on the page and identify missing textual descriptions or ambiguous instructions for the found content.	Partially, it can be subjective at times, and complex pages can be difficult to assess.	No

Success Criteria	How could a LLM enhance the evaluation?	Can the test be automated? (Yes, No, Partially[Why?])	Is there any LLM application for its evaluation?
1.3.4 Orientation	Simulate and compare user experience when switching orientation.	Partially, difficult to assess if a specific display orientation is essential.	No
1.3.5 Identify Input Purpose	Contextually analyze labels and placeholders to determine whether they align with the input's actual purpose.	Yes, although nuanced cases might be subjective, in general it may be possible to automate.	No
1.4.1 Use of Color	Determine if non-color cues are present.	No/Partially, can be tough to assess specific and nuanced cases, where the use of color and non-color cues is very slight.	No
1.4.2 Audio Control	Verify if the autoplaying content is ””justified””.	No, discoverability of pause or stop controls may require human judgment on nuanced cases. Cannot directly ””observe”” behaviours like a user would.	No
1.4.3 Contrast (Minimum)	Provide suggestions when an issue is found.	No/Partially, analyzing the contrast of text in a background of different colors is very difficult to assess automatically.	**Not directly, correction only (x2):** **Paper:** *Fostering websites accessibility: A case study on the use of the Large Language Models ChatGPT for automatic remediation* [29] **Approach:** ChatGPT analyzed the HTML for contrast issues and provided suggestions for better color contrast, validated manually and automatically. **Paper:** *ACCESS: PROMPT ENGINEERING FOR AUTOMATED WEB ACCESSIBILITY VIOLATION CORRECTIONS* [13] **Approach:** GPT-3.5 analyzed and corrected insufficient color contrast in the HTML, ensuring compliance with WCAG 1.4.3.
1.4.4 Resize Text	Could identify clipped text and elements via snapshots.	No/Partially, could identify clipped text and different elements via snapshots but in complex pages with many distinct elements and text that may not be feasible.	No

Success Criteria	How could a LLM enhance the evaluation?	Can the test be automated? (Yes, No, Partially[Why?])	Is there any LLM application for its evaluation?
1.4.5 Images of Text	Can determine whether an image of text serves an essential purpose, and it may be possible to verify if its text in a logo.	Yes/Partially, it can be subjective to fully automate the test.	**Yes:** **Paper:** *Exploring the Use of AI for Enhanced Accessibility Testing of Web Solutions* [9] **Approach:** The authors use Optical Character Recognition (OCR) through the EasyOCR library to extract text from images and compare it to the surrounding webpage content using NLP methods. This helps identify cases where critical text is embedded in images without being made accessible elsewhere on the page.
1.4.10 Reflow	Could identify clipped text and if horizontal scrolling is needed via snapshots.	No/Partially, could identify clipped text and if horizontal scrolling is needed via snapshots, but in complex pages with many distinct elements and text that may not be feasible.	No
1.4.11 Non-text Contrast	Identify contrast boundaries in detected ui components.	Partially, depending on the complexity of the components.	No
1.4.12 Text Spacing	Analyzing visibility of all text with no overlapping, aided by automatic tools to provide the changes.	Partially, can be tough to assess specific and nuanced cases, where the changes or non changes are very slight.	No
1.4.13 Content on Hover or Focus	Interpret and validate how dynamically generated content behaves across various scenarios, such as hover-triggered pop-ups or focus shifts, by simulating user interactions.	No/Partially, it can be very costly and dynamic UI changes can be complex to automate in a test; nuanced cases can prove difficult to automate.	No
2.1.1 Keyboard	Simulate keyboard-only navigation scenarios and identify areas where logical focus order or functionality breaks.	Partially, requires detailed and specific input to evaluate keyboard operability effectively.	**Not directly, correction only:** **Paper:** *ACCESS: PROMPT ENGINEERING FOR AUTOMATED WEB ACCESSIBILITY VIOLATION CORRECTIONS* [13] **Approach:** Missing keyboard navigation aids were identified and rectified by GPT-3.5 in the HTML DOM, ensuring the accessibility of keyboard input functionality.

Success Criteria	**How could a LLM enhance the evaluation?**	**Can the test be automated? (Yes, No, Partially[Why?])**	**Is there any LLM application for its evaluation?**
2.1.2 No Keyboard Trap	Simulate keyboard-only navigation scenarios and identify areas where logical focus order or functionality breaks.	Partially, requires detailed and specific input to evaluate keyboard operability effectively.	**Not directly, correction only:** **Paper:** *ACCESS: PROMPT ENGINEERING FOR AUTOMATED WEB ACCESSIBILITY VIOLATION CORRECTIONS* [13] **Approach:** GPT-3.5 used Chain of Thought (CoT) prompting to detect keyboard traps in the HTML and sequentially address them to ensure users can navigate the site without issues.
2.1.4 Character Key Shortcuts	Analyze event listeners dynamically, simulating real-world scenarios where shortcuts are triggered.	Partially, can be hard to accurately detect dynamic shortcuts, and if the shortcut's effects are subtle it may not detect changes.	No
2.2.1 Timing Adjustable	Simulate real-world workflows to ensure timing adjustments are accessible and logical; identify if users are adequately notified about time limits.	No/Partially, can only automate until a certain point, nuanced cases and conditionally activated timers can be difficult to automate.	No
2.2.2 Pause, Stop, Hide	Check for user control mechanisms and their descriptions. Evaluate whether the animated content is essential based on context or metadata.	No/Partially, moving content can be very complex and hard to automate without human judgment.	No
2.3.1 Three Flashes or Below Threshold	Contextually evaluate the intent of flashing content, if detected.	No, it could be very costly and LLMs can't directly analyze visual content or video frames for actual flashing patterns	No
2.4.1 Bypass Blocks	Detect if the mechanism exists and if it leads to the main content when activated.	Yes/Partially, detecting the accuracy of where the skip link leads to the main content can be somewhat subjective.	No

Success Criteria	How could a LLM enhance the evaluation?	Can the test be automated? (Yes, No, Partially[Why?])	Is there any LLM application for its evaluation?
2.4.2 Page Titled	An accessibility evaluation tool can detect if a website has a title, but a human has to evaluate if the title describes the content of the page.	Yes, although nuanced cases might be subjective, in general it may be possible to automate.	**Yes:** **Paper:** *Capabilities of Large Language Models in Web Accessibility Evaluation: A Design Science Approach* [7] **Approach:** Utilized ChatGPT-3.5 through iterative prompt engineering. The LLM was tasked with determining whether the HTML title tag exists, whether the title is descriptive and relevant to the page content, and whether it identifies the page. The study used ACT test cases and a design science methodology, refining prompts to improve the accuracy and consistency of the LLM's evaluation
2.4.3 Focus Order	Simulate keyboard-only navigation scenarios and identify areas where logical focus order or functionality breaks.	Partially, requires detailed and specific input to evaluate keyboard operability effectively.	No
2.4.4 Link Purpose (In Context)	Comparing the link text or element with the content of the linked page to verify if it serves its purpose	Yes, although nuanced cases might be subjective, in general it may be possible to automate.	**Yes:** **Paper:** *Turning manual web accessibility success criteria into automatic: an LLM-based approach* [4] **Approach:** Claude compares link text with the content of the linked page to verify whether the link description is accurate and relevant. **Not directly, correction only:** **Paper:** *Fostering websites accessibility: A case study on the use of the Large Language Models ChatGPT for automatic remediation* [29] **Approach:** ChatGPT evaluated whether the link text adequately described the page it linked to, results compared with manual testing.
2.4.5 Multiple Ways	Determine if a variety of navigation options are present and evaluate whether the available methods, like search or sitemap, effectively lead to the same key content.	Yes/Partially, complex navigation pages can be hard to assess.	No

Success Criteria	How could a LLM enhance the evaluation?	Can the test be automated? (Yes, No, Partially[Why?])	Is there any LLM application for its evaluation?
2.4.6 Headings and Labels	Evaluate the clarity, relevance, and descriptive quality of headings and labels by comparing them to the content they introduce or describe.	Yes, although it can be subjective.	**Not directly, correction only:** **Paper:** *ACCESS: PROMPT ENGINEERING FOR AUTOMATED WEB ACCESSIBILITY VIOLATION CORRECTIONS* [13] **Approach:** Few-shot prompting helped the model identify missing headings and labels, and automatically generate appropriate elements to ensure compliance with WCAG.
2.4.7 Focus Visible	Verify if the detected focus is sufficient for the element in question.	No/Partially, focus indicators could go undetected in automation. Could analyze snapshots but it's very costly to verify everything in just one page.	No
2.4.11 Focus Not Obscured (Minimum)	Verify if the detected focus is visible.	No/Partially, can be subjective. Could analyze snapshots but very costly to verify everything in just one page.	No
2.5.1 Pointer Gestures	Maybe check the code for gesture dependencies and verify if that single-point alternatives exist.	No, gesture dependent mechanisms can be hard to locate and the whole process can turn very costly	No
2.5.2 Pointer Cancellation	Analyze the code for event handlers and flag inconsistencies.	No, simulating the pointer functions can be very hard and the whole process can turn very costly	No
2.5.3 Label in Name	Detect inconsistencies between visible text and accessible names.	Yes, although it can be subjective.	No
2.5.4 Motion Actuation	Verify motion-based functionality through the code or documentation and analyze if alternatives exist, or if it is essential	No, simulating motion actuation can be very hard and the whole process can turn very costly	No
2.5.7 Dragging Movements	Analyze the code to detect dragging features and verify if alternatives exist, or if it is essential.	No, simulating the pointer functions can be very hard and the whole process can turn very costly	No
2.5.8 Target Size (Minimum)	They can assess whether smaller targets are justified by context, or just small height or width.	No/Partially, difficult and costly to assess the size of all clickable elements in one page, even if just the ””small”” ones.	No

Success Criteria	How could a LLM enhance the evaluation?	Can the test be automated? (Yes, No, Partially[Why?])	Is there any LLM application for its evaluation?
3.1.1 Language of Page	Could detect the primary language of a page automatically; Suggesting the correct language code if the lang attribute is missing or incorrect, because if the ¡HTML¿ element's lang attribute is missing or invalid, it will fail an automated check.	Yes	**Yes:** **Paper:** *Exploring the Use of AI for Enhanced Accessibility Testing of Web Solutions* [9] **Approach:** The authors use the LanguageIdentifier library, which employs pre-trained neural models to detect the primary language of a webpage. The detected language is then compared to the language specified in the HTML's language attribute, highlighting mismatches to flag nonconformance
3.1.2 Language of Parts	Could detect the primary language of a page automatically and check if language tags correctly reflect the language of the text. Could identify blocks of texts different from the main ¡lang¿ attribute and verify if they have the appropriate tag.	Yes, but mixed language sentences could be ambiguous, or even expressions used in different languages like ""deja vu"".	**Yes(x2):** **Paper:** *Turning manual web accessibility success criteria into automatic: an LLM-based approach* [4] **Approach:** The LLMs (Claude, GPT-3.5, GPT-4.0) were used to identify improperly tagged language sections in HTML content. Their methodology involved feeding HTML content to the LLMs, prompting them to check if language tags correctly reflected the language of the text. GPT-4 achieved 100% accuracy on their benchmark test cases for detecting such issues, highlighting its effectiveness over traditional web accessibility tools that often failed to detect these mismatches. **Paper:** *Exploring the Use of AI for Enhanced Accessibility Testing of Web Solutions* [9] **Approach:** The authors use the LanguageIdentifier library with pre-trained neural models to detect the language of individual text parts on a webpage. Detected languages are compared with the specified language attributes in the HTML for those text parts, flagging mismatches to identify nonconformance.

Success Criteria	How could a LLM enhance the evaluation?	Can the test be automated? (Yes, No, Partially[Why?])	Is there any LLM application for its evaluation?
3.2.1 On Focus	If moving focus to a component leads to a description or message update, like an alert, a LLM could be trained to understand if this change is relevant and non-disruptive to the user experience or if it's a violation of accessibility standards; interpreting what constitutes a meaningful vs. disruptive change of context. Could reduce false positives by understanding intent better than rule-based automation alone.	Partially, determining what constitutes an ""unexpected"" context change, rely on subjective interpretation and user expectations, which are challenging to codify completely	No
3.2.2 On Input	Help in analyzing if the page state change is intentional, to determine if the change is truly unexpected or disruptive. If selecting a checkbox causes a pop-up to appear, or to submit the form automatically, an LLM could evaluate whether this feedback is necessary and how it impacts usability.	Partially, detecting all unexpected context changes is very hard. Nuanced changes in visual or contextual information, can be challenging to automate. Submission or page reloads on focus, might not always be caught without manually triggering	No
3.2.3 Consistent Navigation	Analyze and interpret navigation content to determine functional similarity, even if labeled differently. Analyze whether there are changes in navigation order in the different menus of the page	Partially, complex navigation structures could be an issue.	No
3.2.4 Consistent Identification	Interpret and compare accessible names (or other identifiers) to determine if they are semantically equivalent, even if not identical.	Partially, complex navigation structures could be an issue.	No

Success Criteria	How could a LLM enhance the evaluation?	Can the test be automated? (Yes, No, Partially[Why?])	Is there any LLM application for its evaluation?
3.2.6 Consistent Help	Interpret if different mechanisms count as help, like FAQ for example. Although it doesn't require for help to be provided it can identify ""more hidden"" help mechanism and if they are consistent	Yes/Partially, identification of the help mechanism provided could be tricky.	No
3.3.1 Error Identification	Validate the usability and correctness of the error message provided, (helpful in sc 3.3.3).	Partially, verifying if the error is appropriately linked to the field, in context sensitive cases, can be challenging.	No
3.3.2 Labels or Instructions	Assess whether the label or instructions are clear, meaningful, or contextually appropriate. Automated tools can only identify where the label may be present.	Yes, although it can be subjective.	**Not directly and correction only (x2):** **Paper:** *Fostering websites accessibility: A case study on the use of the Large Language Models ChatGPT for automatic remediation* [29] **Approach:** ChatGPT was prompted to generate form labels based on the form's HTML structure and input types. **Paper:** *ACCESS: PROMPT ENGINEERING FOR AUTOMATED WEB ACCESSIBILITY VIOLATION CORRECTIONS* [13] **Approach:** Using Chain of Thought (CoT) prompting, GPT-3.5 sequentially identified and addressed missing or incorrect form labels.
3.3.3 Error Suggestion	Assess whether the provided guidance is sufficient, helpful, and/or not too vague, like just ""Error"".	Yes/Partially, can be subjective to evaluate if the provided error message is helpful or not.	No
3.3.4 Error Prevention (Legal, Financial, Data)	Detect forms or processes involving legal, financial, or sensitive data.	No/Partially, hard and costly to automate the test for all the conditions this guideline imposes.	No
3.3.7 Redundant Entry	Assess whether redundant fields serve a valid purpose by analyzing the workflow and instructions.	Partially, can prove to be very subjective.	No

Success Criteria	How could a LLM enhance the evaluation?	Can the test be automated? (Yes, No, Partially[Why?])	Is there any LLM application for its evaluation?
3.3.8 Accessible Authentication (Minimum)	Analyze authentication flows to detect elements that might violate the guideline.	No/Partially, LLMs can easily throw false positives when identifying cognitive function tests. SC has many exceptions, and it can prove to be costly to automate a test that considers all of them.	No
4.1.2 Name, Role, Value	Validate that the programmatically determinable name, role, and value match the visual or functional intent of the component. Analyze the relationship between ARIA attributes, component behavior, and design intent to identify mismatches. Evaluate complex, scripted components to ensure that custom behaviors are accessible.	Yes/Partially, process can be very costly, specially in complex pages	**Not directly and correction only (x2):** **Paper:** *Fostering websites accessibility: A case study on the use of the Large Language Models ChatGPT for automatic remediation* [29] **Approach:** ChatGPT identified and remediated ARIA attribute issues, ensuring proper accessibility roles and values. **Paper:** *ACCESS: PROMPT ENGINEERING FOR AUTOMATED WEB ACCESSIBILITY VIOLATION CORRECTIONS* [13] **Approach:** GPT-3.5 added missing ARIA attributes to the HTML DOM to ensure compliance with the guideline.
4.1.3 Status Messages	Evaluate whether the content and context of status messages are clear and convey correct information.	Partially, can be tough to automate the testing of the timing of a status shows up, as it can be subjective to evaluate if it disrupts the workflow or not.	No

Note: The column “Is there any LLM application for its evaluation?” was filled based on research available up to 12 January 2025. Publications after this date are not included.