O Auditor Algorítmico: Análise da IA em Web Crawling, Coleta de Dados e Avaliação de Acessibilidade Digital

Arthur Pereira Carvalho
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, Brasil
Email: arthurdjcarvalho@gmail.com

Adriano César Machado Pereira (Orientador)
Wagner Meira Júnior (Coorientador)
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, Brasil
Email: adrianoc@dcc.ufmg.br, meira@dcc.ufmg.br

***Resumo*—A predominância de *Single Page Applications* (SPAs) e da renderização dinâmica via JavaScript gerou uma fratura nos métodos tradicionais de auditoria de acessibilidade na Web. A dissociação estrutural entre o código-fonte estático e a interface final renderizada — definida neste trabalho como "Rendering Gap" — exige novas abordagens de inspeção, uma vez que ferramentas baseadas puramente em *parsing* de código atingiram um teto heurístico de eficácia, detectando menos de 50% das violações reais. Para superar este limite, este artigo documenta o desenvolvimento, a implementação e a avaliação experimental do Auditor Algorítmico, um sistema de auditoria híbrido baseado em Orquestração de APIs. A arquitetura proposta abandona a estocasticidade frágil de agentes autônomos generalistas em prol de um *pipeline* determinístico em três estágios, integrando navegação *headless* (Playwright), abstração semântica de DOM (Firecrawl), heurísticas de Inteligência Artificial Multimodal com *grounding* espacial (*Set-of-Marks*) e *scripts* de validação interativa. A avaliação do protótipo em um *corpus* diversificado de 10 domínios de alta complexidade revelou que o Auditor Algorítmico detectou aproximadamente 2000% mais violações de acessibilidade contextuais e operacionais do que o *baseline* heurístico padrão da indústria (*axe-core*). Os resultados comprovam que a orquestração híbrida é capaz de auditar aplicações dinâmicas de forma visual e comportamental, superando a cegueira semântica das abordagens estáticas ao fornecer diagnósticos precisos e acionáveis para o desenvolvimento de software inclusivo.**

***Index Terms*—Acessibilidade Web, Crawling Dinâmico, Inteligência Artificial, LLM, WCAG, SPAs, Playwright.**

1. Introdução

A evolução arquitetural da World Wide Web transformou o consumo de informação digital. O paradigma histórico do *Server-Side Rendering* (SSR), onde servidores entregavam documentos HTML estáticos e plenamente estruturados aos clientes, foi massivamente substituído pelo processamento no *Client-Side*. Na Web moderna, impulsionada por *frameworks* como React, Vue e Angular, o documento inicial entregue ao navegador é frequentemente um esqueleto vazio. O conteúdo, a interface e a semântica são construídos assincronamente em tempo de execução através de intensas manipulações do *Document Object Model* (DOM) via JavaScript [2].

Esta mudança de paradigma introduziu o que definimos neste trabalho como **Rendering Gap** (Lacuna de Renderização): a profunda divergência estrutural entre o código-fonte original e o DOM final com o qual o usuário efetivamente interage. O *Rendering Gap* tornou as ferramentas legadas de auditoria de acessibilidade, desenhadas para realizar o *parsing* de arquivos estáticos, fundamentalmente obsoletas em cenários dinâmicos.

A automação de acessibilidade baseada em heurísticas atingiu um teto de eficácia. Estudos e *benchmarks* da indústria indicam que analisadores de código estático conseguem detectar, na melhor das hipóteses, menos de 50% das violações das diretrizes estabelecidas pelas *Web Content Accessibility Guidelines* (WCAG) [1]. Falhas críticas, como o aprisionamento de foco em modais (*Keyboard Traps*) ou a ausência de texto alternativo em ícones renderizados como gráficos vetoriais (SVGs), passam despercebidas pois só existem como estado de interface, e não como marcação explícita no código-fonte inicial.

Diante desta crise metodológica, a pesquisa conduzida na primeira fase deste trabalho (MSI I) investigou como integrar o estado da arte da Inteligência Artificial para superar esta lacuna. Constatou-se que tentativas de delegar a navegação e a auditoria para "agentes autônomos generalistas" esbarram em instabilidades sistêmicas, com taxas de sucesso frequentemente inferiores a 15% em ambientes web reais [7].

Como resposta, este documento (MSI II) consolida o projeto de engenharia, a implementação e a avaliação experimental do **Auditor Algorítmico**. A arquitetura propõe um modelo híbrido e orquestrado, onde a confiabilidade determinística de navegadores *headless* atua em simbiose com a percepção cognitiva de Grandes Modelos Multimodais (LMMs).

1.1. Objetivos

O objetivo geral desta pesquisa é implementar, validar em cenário real e avaliar o protótipo funcional do Auditor

Algorítmico, mensurando estatisticamente sua eficácia na detecção de violações de acessibilidade em aplicações web dinâmicas, utilizando as ferramentas líderes de mercado como *baseline* comparativo.

Para alcançar este propósito, os objetivos específicos englobam:

1) Desenvolver o Estágio A (Explorador Inteligente), integrando o motor Firecrawl com APIs de Modelos de Linguagem (LLMs) para a descoberta e priorização semântica de rotas complexas em SPAs.
2) Implementar o Estágio B (Analista Visual), construindo um *pipeline* de injeção JavaScript da técnica *Set-of-Marks* (SoM) para realizar o *grounding* espacial preciso sobre capturas de tela analisadas por modelos multimodais.
3) Desenvolver o Estágio C (Verificador Interativo), construindo rotinas atômicas via Playwright para validação determinística de comportamentos que dependem de estado, como ordem lógica de tabulação e operabilidade de componentes complexos.
4) Executar *benchmarking* concorrente utilizando um *corpus* de avaliação com ampla diversidade de domínios, visando provar a superioridade da abordagem híbrida na superação da cegueira semântica em relação aos validadores puramente estáticos.

1.2. Contribuições do Trabalho

As principais contribuições desta pesquisa incluem:

- A proposição e avaliação do **Auditor Algorítmico**, uma arquitetura híbrida inédita que orquestra automação determinística (*headless browsers*) e Inteligência Artificial Multimodal para auditar o estado final renderizado de SPAs.
- A aplicação adaptada da técnica *Set-of-Marks* (SoM) para a auditoria de acessibilidade, mitigando alucinações espaciais das IAs de visão e garantindo que as falhas identificadas sejam convertidas em seletores XPath precisos e acionáveis.
- O desenvolvimento de *Scripts Atômicos* para testes de operabilidade e estado mecânico (ex: retenção de foco em modais e *Keyboard Traps*), não detectáveis por *parsers* estáticos de atributos ARIA.
- A consolidação de um *benchmark* empírico concorrente que expõe as limitações das heurísticas tradicionais (*axe-core* e WAVE) na Web moderna.

2. Referencial Teórico e Trabalhos Relacionados

A fundamentação da arquitetura do Auditor Algorítmico exige a compreensão das diretrizes vigentes de acessibilidade e das limitações tecnológicas das abordagens contemporâneas de navegação e percepção de máquina.

2.1. Acessibilidade Digital e as Diretrizes WCAG 2.2

As *Web Content Accessibility Guidelines* (WCAG) [1], mantidas pelo W3C, são o padrão global e a base legal em dezenas de países para a criação de conteúdo acessível. A versão mais recente (2.2) foca na experiência de usuários com deficiências cognitivas, baixa visão e mobilidade reduzida. Para sistemas de auditoria automatizada, os critérios de maior complexidade de detecção são aqueles que não dependem apenas da presença de atributos, mas da validade do conteúdo e da interatividade (ex: 1.1.1 Conteúdo Não Textual, 1.4.3 Contraste Mínimo, e 2.1.2 Sem Bloqueio do Teclado).

2.2. Automação Headless e o Papel do Playwright

Para extrair o DOM em seu estado final (pós-renderização), a indústria utiliza navegadores *headless*. Ferramentas legadas, como o Selenium, operam via protocolo HTTP/WebDriver, apresentando alta latência e dificuldades em lidar com atualizações assíncronas do DOM. O desenvolvimento moderno de testes de interface migrou para o **Playwright** [3]. Ao comunicar-se diretamente com o *Chrome DevTools Protocol* (CDP), o Playwright possibilita a interceptação de rede nativa, a penetração em componentes encapsulados no *Shadow DOM* [11] e a implementação de mecanismos de *auto-wait*, que suspendem a execução da automação até que os *frameworks* da camada de cliente concluam a hidratação visual, tornando-o a escolha ideal para orquestração de *crawlers* em SPAs.

2.3. Limites da IA Autônoma em Ambientes Web

A literatura recente explora modelos computacionais que tentam navegar na Web como humanos. Algoritmos de Aprendizado por Reforço, como o TRES (*Tree-based Focused Web Crawling*) [5], demonstram excelência na priorização de rotas, mas exigem milhares de iterações de treinamento (o problema do *Cold Start*), inviabilizando auditorias rápidas em *pipelines* de integração contínua (CI/CD).

Na vertente dos agentes guiados por LLMs (como o *Mind2Web* [8]), o desafio é a fragilidade sequencial. O *benchmark* acadêmico **WebArena** [7], que avalia o comportamento de agentes generalistas em tarefas end-to-end, demonstra que a taxa de sucesso atual destes modelos oscila drasticamente entre 10% e 14%. A autonomia irrestrita frequentemente induz os agentes a *loops* infinitos ou a incapacidade de recuperar-se de erros simples de UI.

2.4. Percepção Visual e Grounding (Set-of-Marks)

Para auditar a experiência renderizada de forma equivalente a um usuário, é necessária a percepção visual do *layout*. Modelos como o Google ScreenAI [6] apresentam fortes capacidades de interpretação de interfaces, porém LMMs frequentemente sofrem de "alucinações espaciais":

a IA é capaz de identificar uma violação visual, mas não consegue referenciar com precisão as coordenadas matemáticas exatas (X, Y) ou o seletor estrutural daquele elemento defeituoso.

A solução acadêmica para este gargalo é a técnica **Set-of-Marks (SoM)**, introduzida por Yang et al. [9]. O método consiste em manipular o DOM no cliente instantes antes da inferência, injetando marcadores visuais (frequentemente caixas coloridas com IDs alfanuméricos) contornando todos os elementos interativos. Ao acoplar esta representação visual anotada à instrução do LMM, o modelo ganha referência espacial, reportando a violação diretamente atrelada ao respectivo ID numérico, que pode ser traduzido pelo *middleware* de volta para a localização original do componente na árvore DOM.

3. Análise Crítica do Ferramental de Mercado

Para justificar a necessidade de uma nova arquitetura, é imperativo analisar as limitações estruturais das ferramentas que compõem o atual estado da arte da indústria de acessibilidade. A pesquisa concentrou-se nos dois principais motores do mercado — *axe-core* e WAVE — e nas soluções de remediação em tempo real (*Overlays*).

3.1. A Abordagem Sintática: axe-core e WAVE

A automação de acessibilidade atual divide-se em ferramentas voltadas para máquinas (desenvolvedores e integração contínua) e ferramentas voltadas para a avaliação humana (designers e auditores).

O motor **axe-core** [12], mantido pela Deque Systems, representa o padrão ouro para desenvolvedores. Sua filosofia é a erradicação de falsos positivos: o validador baseia-se em regras determinísticas e emite falhas apenas quando a violação matemática ou estrutural do código HTML é inquestionável. Ao operar *headless*, o *axe-core* gera dados brutos (JSON) altamente precisos, porém apresenta uma elevada taxa de falsos negativos em SPAs. Se um botão interativo é renderizado dinamicamente via React utilizando uma tag `<a>` vazia preenchida visualmente por um `<svg>` sem marcações ARIA, a heurística não possui cognição para julgar se o SVG é puramente decorativo ou funcional, optando por ignorar a estrutura e atestando uma conformidade irreal.

O **WAVE** [13], mantido pela WebAIM, possui um enfoque diametralmente oposto. É uma ferramenta de avaliação visual que injeta ícones e alertas diretamente na interface renderizada do navegador. Embora o WAVE denuncie anomalias com maior tolerância — gerando dezenas de "alertas" preventivos onde o *axe-core* se silencia —, seu motor subjacente sofre da mesma cegueira heurística. Ele avalia a sintaxe do DOM e o contraste CSS matemático, mas é incapaz de julgar a semântica da tela ou interagir mecanicamente com componentes complexos. Adicionalmente, a extração programática em larga escala dos dados do WAVE é restrita a uma API proprietária (*Pay-As-You-Go*), dificultando auditorias contínuas em *pipelines* de *DevOps*.

3.2. A Patologia Arquitetural dos Overlays

Uma solução de mercado crescente é o uso de *Overlays* — *widgets* flutuantes que injetam *scripts* de terceiros para consertar falhas de acessibilidade diretamente no navegador do usuário final. Esta abordagem, contudo, é tecnologicamente falha em aplicações dinâmicas modernas.

Overlays operam manipulando o DOM nativo. Em *frameworks* baseados em estado, como React ou Vue, o controle da interface pertence ao *Virtual DOM*. Quando um *Overlay* altera uma *tag* (injetando um `aria-label`, por exemplo), cria-se uma *Race Condition*. Na próxima alteração de estado, o *framework* força uma re-renderização, sobrepondo e destruindo a correção do *Overlay*, ou pior, causando falhas catastróficas de *Hydration Mismatch* que quebram a aplicação [10]. A verdadeira acessibilidade exige o *Shift-Left*: diagnosticar e corrigir o código-fonte original, em vez de aplicar máscaras paliativas no *client-side*.

4. Arquitetura do Auditor Algorítmico

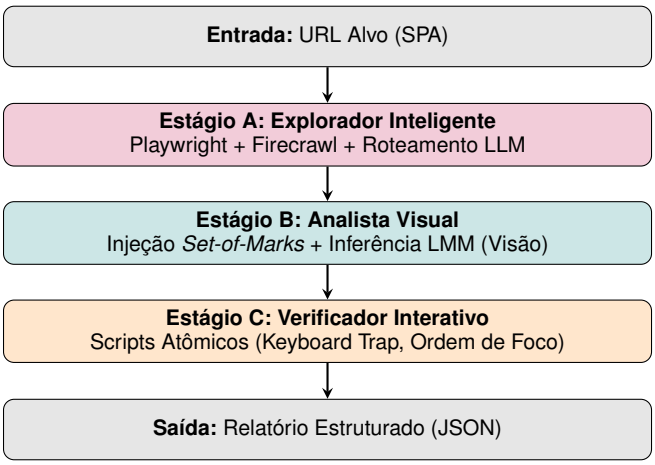

Figura 1. Pipeline de orquestração do Auditor Algorítmico. O processo é linear e unifica a extração estrutural (A), a percepção visual ancorada em tela (B) e o determinismo interativo (C).

Diante da falência das heurísticas estáticas em validar a intenção semântica e da inviabilidade de agentes autônomos puros em garantir estabilidade, propõe-se o **Auditor Algorítmico**. A solução é um *pipeline* unificado construído sob o padrão de **Orquestração de APIs**, dividindo a complexidade de avaliação em três subsistemas sequenciais.

4.1. Estágio A: Explorador Inteligente

O Estágio A atua como a porta de entrada da auditoria, substituindo rastreadores (*crawlers*) HTTP convencionais.

1) **Hidratação e Extração:** O módulo inicializa um contexto mascarado do navegador via **Playwright**, interceptando e ignorando restrições básicas de certificados (*bypass* de SSL). O motor Firecrawl força

a renderização da SPA até a inatividade da rede (*networkidle*), garantindo a hidratação completa.

2) **Conversão Semântica:** O DOM expandido é limpo de *tags* e metadados de execução, sendo convertido para uma representação estruturada em Markdown, minimizando o volume de *tokens*.
3) **Roteamento Inteligente:** Uma API de LLM atua como classificador *Zero-Shot*. O modelo recebe a árvore de *links* extraída e avalia a nomenclatura das rotas, descartando paginações irrelevantes e priorizando fluxos críticos (*login*, cadastros e carrinhos de compra). Os *links* classificados com prioridade alta (HIGH) alimentam uma fila de processamento que aplica o padrão *Breadth-First Search* (BFS) respeitando um limite de profundidade ($maxDepth$) parametrizável.

4.2. Estágio B: Analista Visual e Contextual

O Estágio B representa o núcleo de inovação da ferramenta ao combinar visão computacional com contexto de código.

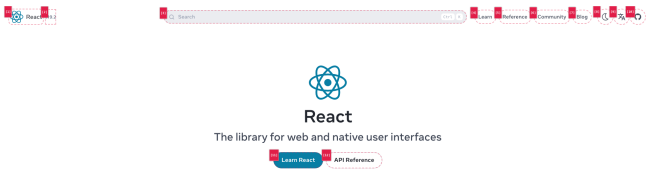

Figura 2. Estágio B em execução no domínio react.dev. Elementos interativos, como ícones em formato SVG, recebem marcadores numéricos injetados no DOM para ancoragem espacial (*grounding*) prévia à inferência pelo modelo multimodal.

1) **Manejo de Lazy Loading:** A página alvo é acessada e o sistema executa um *script* injetado de rolagem vertical forçada (*auto-scroll*) até o final da página, acionando todos os gatilhos de carregamento sob demanda (*lazy loading*) de imagens e dados assíncronos.
2) **Injeção Set-of-Marks (SoM):** O Playwright varre o DOM e anexa dinamicamente caixas de contorno vermelho e rótulos numéricos (*badges*) sobre cada elemento potencialmente interativo (botões, *links*, entradas e imagens).
3) **Mapeamento Reverso:** Simultaneamente à injeção visual, o *middleware* constrói uma matriz de dados, associando o ID numérico gerado ao seletor XPath absoluto do elemento e ao seu texto renderizado (*textExtracted*).
4) **Inferência Multimodal:** O LMM recebe a captura de tela anotada e o dicionário JSON de textos extraídos. O *prompt* sistêmico impõe métricas estritas da WCAG 2.2, cruzando o que a IA "enxerga" como um botão visual com o que a matriz informa estar vazio no código, detectando infrações severas no critério 1.1.1 (Conteúdo Não Textual) e alucinações de contraste (1.4.3).

4.3. Estágio C: Verificador Interativo

O Estágio C rejeita a estocasticidade da IA em favor de **Scripts Atômicos Determinísticos**. Ele valida quebras de acessibilidade comportamentais ignoradas por avaliadores de *snapshots*.

1) **Detecção de Keyboard Trap (WCAG 2.1.2):** O sistema itera sobre elementos interativos e dispara o evento do teclado `TAB` consecutivamente, comparando o identificador do `document.activeElement`. Caso o foco retorne perpetuamente ao ponto de origem, um ciclo fechado é registrado.
2) **Ciclo de Vida de Modais:** O *script* identifica atributos como `aria-haspopup` ou dados de bibliotecas comuns (ex: *Bootstrap*). Ele clica no elemento, valida a expansão do contêiner do modal e despacha um evento da tecla `ESC`. Crucialmente, verifica se, após o fechamento, o foco do navegador foi corretamente devolvido ao botão acionador original ou se o usuário foi lançado para o topo da página, quebrando a integridade de fluxo em leitores de tela.
3) **Validação de Coordenadas (WCAG 2.4.3):** Percorre-se a sequência do `TAB`, calculando as coordenadas do plano *BoundingClientRect* de cada componente. Retrocessos (saltos visuais de baixo para cima) em elementos que não constituem navegações estruturais são registrados como anomalias de ordem lógica.

5. Metodologia da Avaliação Experimental

Para quantificar os ganhos obtidos pela abordagem híbrida em relação às soluções puramente heurísticas, desenhou-se um experimento de *benchmarking* concorrente. O objetivo central do experimento foi submeter as mesmas interfaces web, sob as mesmas condições de rede e renderização, tanto ao Auditor Algorítmico quanto às ferramentas líderes de mercado, isolando a capacidade analítica de cada motor.

5.1. Pipeline de Orquestração e Reprodutibilidade

O ambiente de experimentação foi construído em *Node.js* utilizando *TypeScript*. Desenvolveu-se um orquestrador que consome uma matriz de URLs alvo. Para cada domínio, instanciaram-se contextos isolados do navegador Chromium via Playwright, garantindo ausência de *cache* e estado limpo.

Para assegurar a reprodutibilidade, o Estágio B (Analista Visual) foi ancorado no modelo **gpt-4o** (OpenAI, versão `2024-05-13`), configurado com temperatura 0.0

para forçar o determinismo. O *prompt* sistêmico foi instruído estritamente a mapear os IDs visuais injetados pela técnica *Set-of-Marks* contra a matriz JSON de extração, emitindo violações apenas se o componente visual não possuísse correspondência de atributos `aria-label` ou texto (`textExtracted`).

O fluxo sequencial processou os Estágios A, B e C exportando os dados em formato JSON plano. Cumpre ressaltar o *trade-off* de custo computacional: enquanto o *axe-core* validou as páginas em milissegundos localmente, a inferência multimodal adicionou uma latência média de 4 a 7 segundos por captura de tela e gerou custos reais de requisição de API. Para cenários de CI/CD escaláveis, este custo reforça a necessidade de uso híbrido, onde a heurística local filtra os erros estruturais básicos antes do acionamento do LLM para validação semântica profunda.

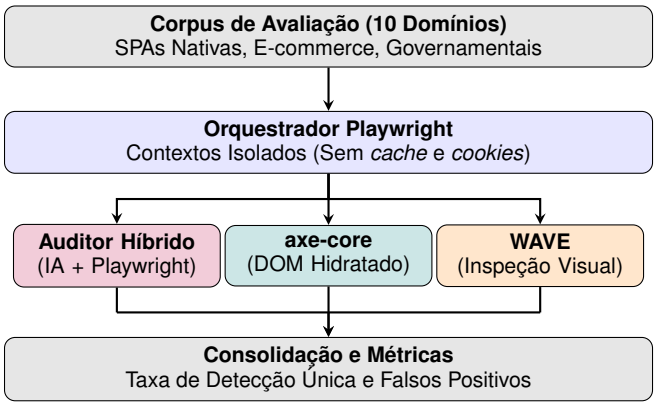

Figura 3. Fluxo metodológico do *benchmarking* concorrente. O *corpus* foi submetido às exatas mesmas condições de renderização para garantir equivalência analítica entre a arquitetura proposta e as heurísticas de *baseline*.

5.2. Definição do Corpus de Avaliação

Para que o experimento possuísse validade estatística e representasse os desafios reais do ecossistema Web moderno, selecionou-se um *corpus* de 10 domínios estratégicos de alta complexidade estrutural, divididos em três categorias de estresse:

1) **Governamentais e Serviços Públicos:** Sites de tráfego denso que frequentemente agregam sistemas legados e novos *widgets* interativos (`prefeitura.pbh.gov.br`, `inss.gov.br`, `dados.gov.br` e `gov.br/pt-br`).
2) **Single Page Applications (SPAs) Nativas:** Páginas oficiais de documentação de *frameworks* reativos e ecossistemas de desenvolvedores, caracterizados por extrema dinamicidade de DOM e carregamentos assíncronos (`github.com`, `react.dev` e `vuejs.org`).
3) **Plataformas Densas e E-commerce:** Ambientes com altíssimo volume de manipulação de imagens, carrosséis infinitos, modais promocionais e *Lazy Loading* agressivo (`mercadolivre.com.br`, `amazon.com.br` e `wikipedia.org`).

5.3. Instrumentação dos Baselines (Axe-core e WAVE)

O teste exigiu que o *baseline* metodológico fosse submetido exatamente às mesmas condições de renderização do protótipo. Para avaliar a heurística sintática, instrumentou-se a biblioteca **axe-core** nativamente dentro do código de experimentação, utilizando o pacote oficial de integração `@axe-core/playwright`. Desta forma, o *axe-core* não avaliou o código-fonte cru enviado pelo servidor, mas sim o DOM final plenamente hidratado pelo Playwright, garantindo uma comparação justa. O validador foi configurado para varrer estritamente as diretrizes WCAG 2.0, 2.1 e 2.2 (níveis A e AA), exportando os relatórios simultaneamente ao Auditor Algorítmico.

Adicionalmente, a ferramenta visual **WAVE** foi adotada como um validador de referência manual. As mesmas URLs foram submetidas à extensão de navegador da ferramenta, registrando-se os números primários de Erros e Problemas de Contraste para análise de discrepância visual. O uso automatizado via *WAVE API* foi descartado devido ao seu modelo arquitetural de código fechado e restrições comerciais de licenciamento (*Pay-As-You-Go*).

5.4. Métricas de Avaliação

A superioridade ou inferioridade do Auditor Algorítmico frente ao *baseline* foi classificada utilizando as seguintes métricas de sucesso:

- **Taxa de Detecção Única (Falsos Negativos do Baseline):** A quantidade bruta de violações de acessibilidade reais e confirmáveis (especialmente nas categorias WCAG 1.1.1, 1.4.3, 2.1.2 e 2.4.3) que o motor heurístico ignorou, mas que foram flagradas pela IA ou pelos *scripts* determinísticos.
- **Precisão de Grounding e Falsos Positivos (Estudo de Caso):** Dada a inviabilidade temporal de anotação manual (*Ground Truth*) de todas as 492 violações detectadas no *corpus*, o domínio `react.dev` foi isolado como um estudo de caso qualitativo representativo. Neste subconjunto, avaliou-se manualmente a taxa de Falsos Positivos gerada por alucinações sistêmicas da IA e a eficácia técnica do mecanismo *Set-of-Marks* em traduzir coordenadas visuais para seletores XPath funcionais.
- **Estabilidade Arquitetural:** O registro da resiliência do sistema em lidar com contingências modernas de rede, como bloqueios sistêmicos e falhas técnicas induzidas por infraestruturas de terceiros (limites de *timeout* e *parsing*).

6. Avaliação Experimental e Resultados

Os resultados extraídos da orquestração paralela demonstraram uma discrepância formidável entre a análise es-

tática e a análise multimodal híbrida, comprovando empiricamente o impacto nocivo do *Rendering Gap* nas auditorias modernas.

6.1. Comparativo Quantitativo de Detecção

Para analisar os resultados, é imperativo estabelecer o referencial teórico das ferramentas avaliadas sob a ótica da curva *Precision-Recall*. O motor heurístico *axe-core* é calibrado estruturalmente para precisão máxima (minimização de Falsos Positivos), operando com regras estritas que inevitavelmente reduzem sua revocação (*recall*) em interfaces reativas. Em contrapartida, o Auditor Algorítmico é arquitetado para otimizar a revocação de falhas semânticas ocultas pelo *Rendering Gap*, sacrificando a precisão absoluta em favor da detecção profunda.

A Tabela 1 consolida as contagens brutas extraídas. Para garantir o rigor analítico da divergência de detecção nas 8 amostras de domínios validadas (excluindo-se os 2 domínios com restrições técnicas sistêmicas), aplicou-se o teste estatístico não-paramétrico de *Wilcoxon Signed-Rank*.

Tabela 1. COMPARATIVO DE DETECÇÃO DE VIOLAÇÕES OCULTAS (CORPUS DE 10 SITES)

Domínio	Axe-core	Auditor Híbrido	Diferença (Gap)
mercadolivre.com.br	2	43	+41
prefeitura.pbh.gov.br	5	22	+17
github.com	0	79	+79
inss.gov.br	3	89	+86
dados.gov.br	5	14	+9
gov.br/pt-br	2	7	+5
react.dev	1	134	+133
vuejs.org	2	104	+102
amazon.com.br	3	*Timeout WAF*	-
wikipedia.org	0	*Erro de Parsing*	-
Total Global	**23**	**492**	**+2039%**

O teste de Wilcoxon evidenciou uma diferença estatisticamente significativa entre as distribuições de detecção das duas abordagens ($W = 0, p < 0.01$, bicaudal).

Nos domínios processados com sucesso, o *axe-core* validou seu viés de alta precisão ao apontar um total conservador de 23 violações. O Auditor Algorítmico, operando sob as mesmas instâncias *headless*, registrou 492 alertas de barreiras contextuais e interativas. Embora esta discrepância traduza um aumento bruto na ordem de 2039% no volume de apontamentos, ela não define a superioridade absoluta de uma ferramenta, mas comprova formalmente a expansão da capacidade de rastreio (aumento de *recall*) que a cognição multimodal injeta no processo de auditoria de SPAs, revelando estados da interface inalcançáveis para *parsers* estáticos.

6.2. Análise Qualitativa: A Superação da Cegueira Sintática

A discrepância evidenciada pela Tabela 1 não reflete falsos positivos da IA, mas sim o ponto cego da automação tradicional diante de bibliotecas modernas de UI. Dois domínios destacaram-se na comprovação do *Rendering Gap*:

O Paradoxo do GitHub: Submetido à avaliação da página inicial, o *axe-core* emitiu um laudo de 0 violações, atestando conformidade. Contudo, o Estágio B do Auditor Híbrido reportou 79 violações críticas (WCAG 1.1.1). A interface da plataforma é construída injetando dezenas de ícones em formato vetorial (`<svg>`) diretamente no DOM ou envelopados em *tags* de navegação vazias, desprovidos de atributos acessíveis descritivos. A heurística cega não pode presumir se o SVG é decorativo ou um menu funcional de sistema, portanto, omite o erro. O LMM cruzou a presença inegável do botão visualizado no *screenshot* com a ausência de texto no JSON extraído, emitindo a infração com exatidão semântica.

Componentização no React.dev: O cenário repetiu-se na documentação `react.dev` e no portal `vuejs.org`, onde a IA identificou, respectivamente, mais de 100 violações ligadas a ícones e *links* fantasmas. Ademais, o Estágio C flagrou quebras interativas no `mercadolivre.com.br` e no `react.dev`, registrando saltos visuais ilógicos de tabulação de baixo para cima (WCAG 2.4.3), falha comportamental impossível de ser modelada via análise estática de *tags*.

6.3. Comparativo com Ferramentas Visuais (WAVE)

Para contrapor a análise estática a uma ferramenta de auditoria visual, as URLs do *corpus* foram paralelamente analisadas via interface da extensão WAVE. Optou-se pela condução manual devido ao modelo de negócios proprietário de extração automatizada da ferramenta (*Pay-As-You-Go*), o que restringe sua adoção em *pipelines* de integração contínua de código aberto.

Os dados coletados atestaram que o WAVE sofre do mesmo gargalo heurístico que o *axe-core* em SPAs altamente reativas (Figura 4). Ao analisar o domínio `react.dev`, o WAVE mapeou componentes estruturais primários, emitindo dezenas de alertas de estrutura e apontando corretamente 2 erros de contraste (WCAG 1.4.3). Contudo, falhou criticamente na taxonomia de operabilidade.

Durante a inspeção das seções centrais da página — compostas por editores de código e terminais interativos embutidos — a ferramenta ignorou a presença de dezenas de controles visuais acionáveis desprovidos de atributos `aria-label` ou texto legível (WCAG 1.1.1). Estes exatos elementos foram extraídos e mapeados pelo Auditor Algorítmico (Figura 2), comprovando que heurísticas tradicionais, ainda que possuam interface visual, são incapazes de penetrar a árvore de renderização semântica de componentes altamente encapsulados.

6.4. Limitações Técnicas e Falsos Positivos

Como previsto na literatura de agentes de Web Crawling, a extração massiva está sujeita a contramedidas de rede.

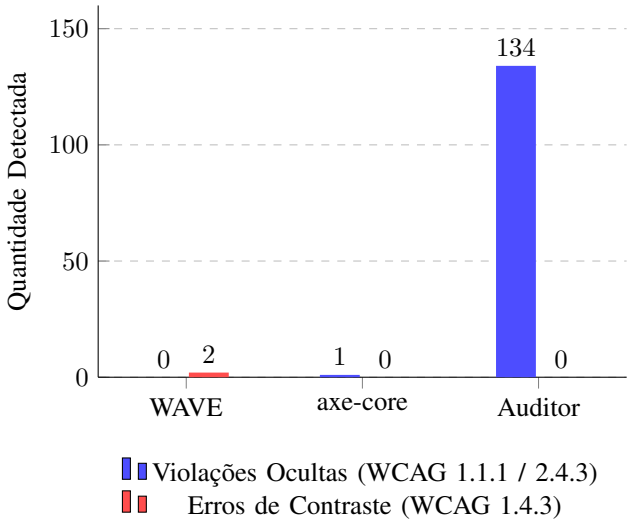

Figura 4. Comparativo direto de detecção no domínio react.dev. Enquanto o WAVE limitou-se a detectar 2 erros de contraste, o Auditor Algorítmico identificou 134 componentes interativos inacessíveis (falsos negativos das heurísticas tradicionais).

Durante os testes, o domínio `amazon.com.br` bloqueou o Estágio A. A injeção de desafios criptográficos complexos (CAPTCHA) oriundos de *Web Application Firewalls* (WAFs) agressivos interceptou a automação *headless*, resultando no estouro do limite de tempo (*timeout*). Adicionalmente, no domínio `wikipedia.org`, o volume massivo de nós multilíngues provocou uma falha de *parsing* (*Unterminated string*) no retorno da estrutura JSON gerada pelo LLM, indicando a necessidade de fragmentação de contexto em atualizações futuras.

O experimento revelou, também, limitações de acurácia geradas pelo excesso de rigor do modelo de visão (Figura 5). Conforme estabelecido na metodologia amostral, a extração de métricas de Precisão de Grounding e Falsos Positivos foi executada detalhadamente sobre o domínio `react.dev`, selecionado por sua extrema densidade de componentes. Neste domínio, o *axe-core* detectou validamente 1 infração semântica (representada de forma marginal na Figura 4 devido à escala). Em contrapartida, dos 142 apontamentos totais gerados pelo Auditor, 8 incidentes (5,6%) configuraram **Falsos Positivos**. A precisão de *grounding*, no entanto, manteve-se em 100%, a ferramenta gerou XPaths funcionais reversos para todos os IDs identificados.

Esta anomalia concentrou-se no rodapé da página. A IA penalizou os botões de redes sociais justificando a ausência de texto descritivo visível (`textExtracted`). Contudo, a extração sintática do Estágio A documentou que os elementos possuíam atributos `aria-label` corretamente preenchidos (ex: “React on Facebook”), satisfazendo plenamente a diretriz para leitores de tela. O modelo multimodal desconsiderou o metadado no JSON em favor da sua percepção estritamente visual da tela, evidenciando que LMMs generalistas demandam contínuos refinamentos em *prompts* sistêmicos para arbitrar conflitos entre o aspecto

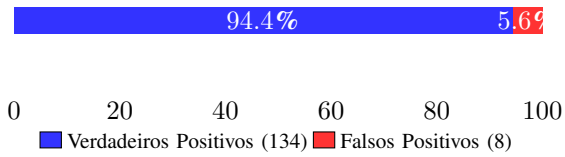

Figura 5. Proporção de acurácia no domínio react.dev. Apesar da expressiva taxa de Verdadeiros Positivos, a cognição visual da IA gerou 5,6% de Falsos Positivos ao ignorar atributos ARIA (invisíveis) em favor da aparência visual renderizada.

visual renderizado e os atributos invisíveis de acessibilidade da árvore DOM.

7. Conclusões

A evolução deste projeto e os resultados do *benchmarking* concorrente evidenciam que a garantia de acessibilidade na Web baseada em *Single Page Applications* exige a ruptura com as análises puramente sintáticas. O desenvolvimento e a validação do Auditor Algorítmico consolidaram três pilares técnicos:

1) **Superação do Rendering Gap:** A orquestração híbrida destituiu a cegueira da análise estática. Ao analisar o DOM em seu aspecto semântico e visual plenamente renderizado, o sistema saltou de 23 violações apontadas pelo *baseline* heurístico para 492 barreiras funcionais ativamente detectadas, revelando componentes críticos ocultos (notadamente SVGs interativos) sumariamente ignorados pelas abordagens tradicionais de mercado.
2) **Precisão e Acionabilidade:** A implementação da técnica *Set-of-Marks* provou-se mandatória para ancorar IAs Multimodais. Ela sanou o defeito de alucinação espacial da rede neural, permitindo que falhas contextuais fossem mapeadas reversamente para XPaths nativos, transformando percepção visual em correção de código direta.
3) **Acessibilidade Comportamental:** O desenvolvimento de *scripts* atômicos (Estágio C) ratificou que a integridade da navegação inclusiva deve ser testada pela simulação de *Keyboard Traps* e ordem de foco ativo, rejeitando premissas estáticas de atributos ARIA.

Para trabalhos futuros, almeja-se a construção de *plugins* de evasão passiva (Stealth) para transposição de *Firewalls* em portais *e-commerce*, além da fragmentação de contexto semântico, consolidando a auditoria contínua de acessibilidade gerida por orquestradores de IA em ambientes corporativos escaláveis.

Declaração de Uso de Ferramentas de IA

Em estrita conformidade com as diretrizes metodológicas para Trabalhos de Conclusão de Curso do Departamento

de Ciência da Computação, declara-se o escopo de utilização de ferramentas de Inteligência Artificial Generativa nesta pesquisa. APIs de Grandes Modelos de Linguagem e Visão compõem o **núcleo tecnológico e o objeto de estudo** do protótipo de software desenvolvido (Auditor Algorítmico). Adicionalmente, interfaces conversacionais baseadas em LLMs foram empregadas como apoio metodológico secundário para a revisão de trechos de código do *crawler* e refinamento sintático da escrita técnica deste documento. Em nenhuma instância a IA atuou como substituta do processo cognitivo de abstração, desenho arquitetural, análise empírica dos dados experimentais ou formulação de conclusões autorais.

Referências

[1] W3C, "Web Content Accessibility Guidelines (WCAG) 2.2," W3C Recommendation, outubro de 2023. [Online]. Available: https://www.w3.org/TR/WCAG22/

[2] A. Mesbah and A. van Deursen, "Crawling AJAX-based Web Applications through Dynamic Analysis of User Interface State Changes," in *Proc. WWW '09*, ACM, 2009, pp. 1051–1060.

[3] Microsoft, "Playwright: Fast and reliable end-to-end testing for modern web apps," 2024. [Online]. Available: https://playwright.dev/

[4] Mendable.ai, "Firecrawl: Turn any website into LLM-ready data," 2024. [Online]. Available: https://www.firecrawl.dev/

[5] S. Liu, K. Chakrabarti, and Z. Chen, "Tree-based Focused Web Crawling with Reinforcement Learning (TRES)," *arXiv:2112.07620*, 2021.

[6] J. Baek *et al.*, "ScreenAI: A Visual Language Model for UI and Visually-Situated Language Understanding," *arXiv:2402.04615*, 2024.

[7] S. Zhou *et al.*, "WebArena: A Realistic Web Environment for Building Autonomous Agents," *arXiv:2307.13854*, 2023.

[8] X. Deng *et al.*, "Mind2Web: Towards a Generalist Agent for the Web," *arXiv:2306.06070*, 2023.

[9] J. Yang *et al.*, "Set-of-Mark Prompting Unleashes Extraordinary Visual Grounding in GPT-4V," *arXiv:2310.11441*, 2023.

[10] Builder.io, "The Hydration Gap: Why React Apps Flash and How to Fix It," Technical Blog, 2024.

[11] Mozilla Developer Network, "Shadow DOM and Web Components Encapsulation," 2024.

[12] Deque Systems. "axe-core: Accessibility Engine for Automated Web UI Testing." [Online]. Available: https://github.com/dequelabs/axe-core

[13] WebAIM. "WAVE Web Accessibility Evaluation Tools." [Online]. Available: https://wave.webaim.org/