Targeted Interaction Agents for Web Accessibility Evaluation

Mingyuan Zhong
Computer Science & Engineering
University of Washington
Seattle, Washington, USA
myzhong@cs.washington.edu

Ajit Mallavarapu
The Information School
University of Washington
Seattle, Washington, USA
ajitm@uw.edu

Mengqi Shi
University of Washington
Seattle, Washington, USA
shi21@uw.edu

Davin Win Kyi
Computer Science & Engineering
University of Washington
Seattle, Washington, USA
davin123@cs.washington.edu

James Fogarty
Computer Science & Engineering
University of Washington
Seattle, Washington, USA
jfogarty@cs.washington.edu

Jacob O. Wobbrock
The Information School
University of Washington
Seattle, Washington, USA
wobbrock@uw.edu

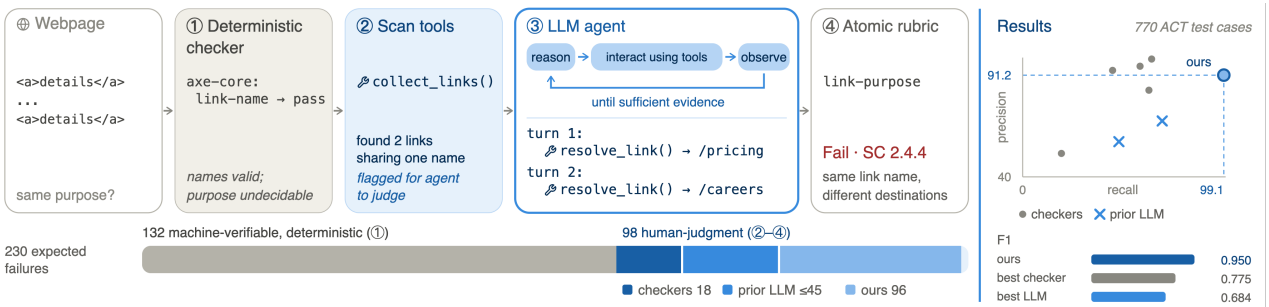

Figure 1: (Left) Our system builds on top of existing deterministic checkers (1) and contributes *interactive tools* and *atomic rubrics* for LLM agent assessment (2)–(4). Of 230 expected ACT failures, our system detects 96 of the 98 that require human judgment. (Right) Our system achieves >99% recall, >90% precision, and the highest F1.

Abstract

Large Language Models (LLMs) have shown potential for expanding the coverage of web accessibility evaluation. However, our experiments reveal that recent LLM-driven systems yield both high false positives and high false negatives on the Accessibility Conformance Testing (ACT) rules developed by the W3C. Our error analysis identifies three recurring causes: missing perception or interaction evidence, missing criterion support, and semantic judgments without sufficient grounding. We propose *interactive tools* and *atomic rubrics* for LLM agents to reliably assess accessibility. The interactive tools inspect or operate a page deterministically to collect evidence, such as focus appearances and error messages. Each rubric covers a testable aspect of a WCAG Success Criterion. They guide agents on how to use tools and interpret evidence. Our preliminary results on a subset of ACT test cases show substantially higher recall (99.1% vs. 68.7%) and precision (91.2% vs. 68.1%) than the strongest evaluated LLM-driven baseline.

CCS Concepts

• **Human-centered computing** → **Accessibility systems and tools**; • **Computing methodologies** → **Intelligent agents**.

UIST Adjunct '26, Detroit, MI, USA

ACM ISBN 979-8-4007-2855-6/26/11
https://doi.org/10.1145/3830397.3841898

ACM Reference Format:
Mingyuan Zhong, Ajit Mallavarapu, Mengqi Shi, Davin Win Kyi, James Fogarty, and Jacob O. Wobbrock. 2026. Targeted Interaction Agents for Web Accessibility Evaluation. In *The 39th Annual ACM Symposium on User Interface Software and Technology (UIST Adjunct '26), November 02–05, 2026, Detroit, MI, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3830397.3841898

1 Introduction

Web accessibility evaluation requires substantial expertise [13] and manual efforts [4] for accurate assessment, because current rule-based checkers provide limited coverage of guidelines [2, 18]. LLM-driven checkers have been proposed to address this coverage gap, especially for assessing errors that require semantic understanding of webpage structure and content [5, 6, 9, 11]. However, as we will show through our experiments, these existing checkers perform poorly when assessed on the Accessibility Conformance Testing (ACT) rules [17] developed by the W3C. We found their high error rates were mainly due to three causes: missing perception and interaction evidence, missing criteria support, and semantic judgments made without sufficient grounding.

To improve LLM's performance in web accessibility evaluation, motivated by prior work where LLM agents are used to identify accessibility errors through interaction [19], we introduce *interactive tools* and *atomic rubrics* that support an agentic system. The tools render and interact with a page programmatically, collecting evidence of markups and metadata, rendered appearance, and how it responds to keyboard, focus, and screen reader use. The

rubrics support an LLM agent to interpret this evidence and run additional tools in multiple turns to determine whether an aspect of a WCAG Success Criterion is met. Over a 324-case subset of ACT cases requiring human judgment, our system detects 98% of expected failures, compared to 18–46% with existing checkers.

2 ACT Dataset and Evaluation

The ACT rules are developed by the W3C to "test conformance with Web Content Accessibility Guidelines (WCAG) 2.2, ... and other accessibility practices" [17]. Although passing these rules does not guarantee WCAG compliance, the rules represent an industry consensus on how WCAG should be implemented [16].

Dataset. To understand the current state of accessibility checkers, we selected a subset of 770 ACT test cases, covering 20 WCAG success criteria (SC) [1]. The 20 SC correspond to 49 ACT rules, the test suites of which contain 230 expected failures and 540 passed or inapplicable cases (see Appendix A). The ACT rules registry [17] records 48 of the 49 rules as having at least one semi-automated (human-in-the-loop) implementation and 10 as also having manual testing methods.

Checkers. We include seven open-source checkers: five deterministic (rule-based): Deque axe-core [2], IBM Equal Access [7], HTML_CodeSniffer [15], Siteimprove Alfa [14], and QualWeb [8]; and two LLM-driven checkers from prior work: GenA11y [6], which extracts page content per SC and judges accessibility by an LLM, and AccessGuru [5], which combines axe with an LLM-based semantic detector. We include these two because of their broad coverage, public availability, and their complementary approaches.

Data Partition. We identify 28 of the 49 rules as *machine-verifiable*, where at least one of the five deterministic checkers achieves perfect recall with zero false positives (448 cases, 132 failures). The other 21 are *human-judgment* rules (324 cases, 98 failures). The two sets overlap by two test cases shared across rules. Appendix B provides a breakdown.

Results. We found deterministic checkers to perform well on the machine-verifiable set and several checkers retain high precision (> 90%) over *all* test cases. However, on the human-judgement set, no deterministic checker exceeds 18.4% recall. LLM-driven checkers improve recall to 42–46% on these rules, but with substantially lower precision across all cases. Thus, their overall F1 (0.53–0.68) is lower than several deterministic checkers (F1 > 0.70), limiting practical utility. Our review of all errors shows current checkers miss failures in three areas: (1) *Meaning*, where semantic judgment is needed, e.g., whether a label is descriptive. (2) *Perception*, where specific detection is needed, e.g., whether there is text inside an image. (3) *Interaction*, where interactive checking is needed, e.g., keyboard traps, error messages. LLM-driven checkers cover meaning only partially and judge perception and interaction without sufficient evidence, resulting in unsupported judgments.

3 System Design and Preliminary Results

Based on these findings, we adopt deterministic checkers where they are accurate (see Appendix C.1). For cases where checkers lack coverage, we create targeted *interactive tools* that collect evidence in meaning, perception, and interaction. We also authored detailed atomic *rubrics* according to WCAG to support LLM assessment.

Tool-supported agentic analysis. As our analysis revealed, the perception and interaction failures require probing tools that existing LLM-driven checkers lack. We therefore built two types of interactive tools: scan tools that collect visual and behavioral evidence, and agentic tools that an LLM can invoke according to a rubric. Detailed lists of these are avialable in Appendix C.2.

Scan tools run once per page to record page behavior and enumerate candidate issues. They construct keyboard-navigation graphs, perform screen-reader traversals, gather screenshots, etc. Potential issues identified in a scan are referred to an LLM agent.

Agentic tools help an LLM agent gather necessary context before making a decision. The tools can resolve links, run OCR, resize pages, and interact with an element while observing any changes.

Rubrics guide an LLM agent in its analysis of a candidate issue. The agent follows the rubric to call necessary tools, reason about all available evidence, and make a decision. Each rubric is atomic: it decides one aspect of one SC, specifying evidence required and decision method based on WCAG's understanding documents. For example, SC 2.4.4 (Link Purpose) is decided by two rubrics: `link-purpose` (purpose clear in context) and `link-name-equivalence` (links with identical names serve equivalent purpose). We authored 21 such rubrics for 14 SC.

Results. On the full corpus, our system detects 99.1% of expected failures, compared to 63.5% for the best deterministic checker and 68.7% for the best LLM-driven baseline (Table 1). Our overall precision of 91.2% is close to the best deterministic checkers (93–99%), surpassing existing LLM-driven ones (58–68%). On the human-judgment rules, where no deterministic checker exceeds 18.4% recall and prior LLM checkers reach at most 0.50 F1, our system detects 98.0% of failures at 86.5% precision, with F1 of 0.92. We include visualizations of these results in Appendix D.

Table 1: Checker performance on the ACT test cases.

Checker	Recall %	Precision %	F1
Full corpus: 770 cases, 230 failures			
QualWeb	63.5	**99.3**	0.775
axe-core	57.8	95.7	0.721
Alfa	62.2	83.6	0.713
IBM Equal Access	44.3	93.6	0.602
HTML_CodeSniffer	19.1	51.8	0.279
AccessGuru	68.7	68.1	0.684
GenA11y	47.4	57.7	0.520
Ours	**99.1**	91.2	**0.950**
Human-judgment rules: 324 cases, 98 failures			
QualWeb (best deterministic)	18.4	**94.7**	0.308
AccessGuru	42.9	60.0	0.500
GenA11y	45.9	55.6	0.503
Ours	**98.0**	86.5	**0.919**
Machine-verifiable rules: 448 cases, 132 failures			
QualWeb (best deterministic)	97.0	**100.0**	**0.985**
AccessGuru	87.9	71.6	0.789
GenA11y	48.5	59.3	0.533
Ours	**100.0**	95.0	0.974

4 Discussion

Our results indicate that reliable LLM-based accessibility evaluation requires a separation between evidence collection (via tools) and interpretation of SC (via rubrics). Interactive tools expose properties that are unavailable from static page representations, while atomic

rubrics limit each LLM judgment to a specific requirement and specify the evidence needed to support it. Instead of relying on LLM's reasoning capabilities for all assessments, we use it as a decision component guided by evidence that complements deterministic checkers, thus limiting false positives.

Our evaluation remains preliminary. The tools and rubrics were developed through analysis of errors on the same selected ACT corpus used for evaluation, and ACT test cases do not capture the complexity of real websites. Future work should evaluate system components on unseen cases and real pages, separately measure candidate generation, rubric routing, tool use, and final judgment. We also note that WCAG does not cover all access needs [10, 12]. Therefore, future work should look into how these uncovered aspects can be reliably supported in accessibility evaluation.

Acknowledgments

This work was supported in part by a Google research award and by Google.org. Any opinions, findings, conclusions or recommendations expressed in our work are those of the authors and do not necessarily reflect those of any supporter.

References

[1] Alastair Campbell, Chuck Adams, Rachael Bradley Montgomery, Michael Cooper, and Andrew Kirkpatrick. 2024. Web Content Accessibility Guidelines (WCAG) 2.2. *W3C* (2024). https://www.w3.org/TR/WCAG22/

[2] Deque. 2021. axe: Accessibility Testing Tools and Software. https://www.deque.com/axe/

[3] Deque. 2024. The Automated Accessibility Coverage Report. https://accessibility.deque.com/hubfs/Accessibility-Coverage-Report.pdf

[4] Shadi Abou-Zahra Eric Velleman. 2014. Website Accessibility Conformance Evaluation Methodology (WCAG-EM) 1.0. *W3C* (2014). https://www.w3.org/TR/WCAG-EM/

[5] Nadeen Fathallah, Daniel Hernández, and Steffen Staab. 2025. AccessGuru: Leveraging LLMs to Detect and Correct Web Accessibility Violations in HTML Code. In *Proceedings of the 27th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '25)*. Association for Computing Machinery, New York, NY, USA, Article 88, 22 pages. doi:10.1145/3663547.3746360

[6] Ziyao He, Syed Fatiul Huq, and Sam Malek. 2025. Enhancing Web Accessibility: Automated Detection of Issues with Generative AI. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE101 (June 2025), 24 pages. doi:10.1145/3729371

[7] IBM Equal Access. 2025. W3C ACT Implementation. https://github.com/IBMa/equal-access/wiki/W3C-ACT-Implementation. Accessed: 2026-07-03.

[8] LASIGE, Faculdade de Ciências da Universidade de Lisboa. [n. d.]. QualWeb: an automatic accessibility evaluator. http://qualweb.di.fc.ul.pt/evaluator/. Source code: https://github.com/qualweb/core. Accessed: July 9, 2026.

[9] Juan-Miguel López-Gil and Juanan Pereira. 2025. Turning manual web accessibility success criteria into automatic: an LLM-based approach. *Universal Access in the Information Society* 24 (2025), 837–852. doi:10.1007/s10209-024-01108-z

[10] Delvani Antônio Mateus, Carlos Alberto Silva, Marcelo Medeiros Eler, and André Pimenta Freire. 2020. Accessibility of mobile applications: evaluation by users with visual impairment and by automated tools. In *Proceedings of the 19th Brazilian Symposium on Human Factors in Computing Systems* (Diamantina, Brazil) *(IHC '20)*. Association for Computing Machinery, New York, NY, USA, Article 4, 10 pages. doi:10.1145/3424953.3426633

[11] Fabio Paternò, Manuela Vinci, Marco Manca, and Nicola Iannuzzi. 2025. How an LLM Can Improve Automatic Web Accessibility Validation ?. In *Proceedings of the 16th Biannual Conference of the Italian SIGCHI Chapter (CHItaly '25)*. Association for Computing Machinery, New York, NY, USA, Article 23, 8 pages. doi:10.1145/3750069.3750310

[12] Dagfinn RØmen and Dag Svanæs. 2012. Validating WCAG versions 1.0 and 2.0 through usability testing with disabled users. *Univers. Access Inf. Soc.* 11, 4 (nov 2012), 375–385. doi:10.1007/s10209-011-0259-3

[13] Letícia Seixas Pereira and Carlos Duarte. 2025. Evaluating and monitoring digital accessibility: practitioners' perspectives on challenges and opportunities. *Universal Access in the Information Society* (2025). doi:10.1007/s10209-025-01210-w

[14] Siteimprove. [n. d.]. Alfa: a suite of open-source libraries for automated accessibility conformance testing. https://alfa.siteimprove.com/. Source code: https://github.com/Siteimprove/alfa. Accessed: July 9, 2026.

[15] Squiz. [n. d.]. HTML_CodeSniffer: a client-side script that checks HTML source code and detects violations of a defined coding standard. https://squizlabs.github.io/HTML_CodeSniffer/. Source code: https://github.com/squizlabs/HTML_CodeSniffer. Accessed: July 9, 2026.

[16] W3C Web Accessibility Initiative. 2026. About ACT Rules. https://www.w3.org/WAI/standards-guidelines/act/rules/about/. Accessed: 2026-07-03.

[17] W3C Web Accessibility Initiative. 2026. Accessibility Conformance Testing (ACT) Overview. https://www.w3.org/WAI/standards-guidelines/act/. Accessed: 2026-07-03.

[18] WebAIM. 2024. WAVE Web Accessibility Evaluation Tools. https://wave.webaim.org/

[19] Mingyuan Zhong, Xia Chen, Davin Win Kyi, Chen Li, James Fogarty, and Jacob O. Wobbrock. 2026. TaskAudit: Detecting Functiona11ity Errors in Mobile Apps via Agentic Task Execution. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (CHI '26)*. Association for Computing Machinery, New York, NY, USA, Article 1200, 20 pages. doi:10.1145/3772318.3791415

A WCAG Success Criteria Selection

We selected WCAG SC to include in our dataset by the number of manual issues detected in Deque's large-scale industry audit dataset [3] and selecting SC whose occurrence is over 0.12%. This resulted in an initial set of 29 SC (Table 2). We then excluded nine SC: seven without any published ACT rules (1.3.2 Meaningful Sequence, 1.4.1 Use of Color, 1.4.11 Non-text Contrast, 1.4.13 Content on Hover or Focus, 2.4.3 Focus order, 3.3.2 Labels or Instructions, 4.1.3 Status Messages), one (4.1.1 Parsing) removed in WCAG 2.2, and one (1.4.10 Reflow) whose test cases are covered by another rule.

Table 2: WCAG 2.1 A/AA success criteria (SC) ranked by manual issue count in the Deque coverage report (294,958 total issues) [3], with corresponding ACT rules and test cases. SC above a 0.12% occurrence threshold (≈354 issues) are selected when corresponding ACT rules exist. Shaded rows: every ACT rule of the criterion is fully covered by at least one deterministic checker (Table 3).

SC	Success criterion	Manual	Auto	Total	Selected	ACT rules	Test cases	Notes
4.1.2	Name, Role, Value	22,011	26,276	48,287	✓	14	211	
1.3.1	Info and Relationships	19,950	16,432	36,382	✓	4	74	
1.4.3	Contrast (Minimum)	14,981	73,733	88,714	✓	1	34	
2.4.3	Focus Order	9,553	0	9,553	—			no published ACT rules
2.1.1	Keyboard	9,178	234	9,412	✓	2	25	
1.1.1	Non-Text Content	7,687	16,014	23,701	✓	6	94	
2.4.7	Focus Visible	7,312	0	7,312	✓	1	9	
1.4.11	Non-text Contrast	4,539	0	4,539	—			no published ACT rules
4.1.1	Parsing	3,351	31,137	34,488	—			obsolete in WCAG 2.2
1.3.2	Meaningful Sequence	3,313	0	3,313	—			no published ACT rules
1.4.1	Use of Color	3,261	452	3,713	—			no published ACT rules
1.4.4	Resize Text	2,099	0	2,099	✓	2	30	
3.3.2	Labels or Instructions	2,019	518	2,537	—			no published ACT rules
2.4.2	Page Titled	1,962	249	2,211	✓	2	20	
1.4.5	Images of Text	1,778	10	1,778	✓	1	15	
2.4.4	Link Purpose (In Context)	1,376	0	1,376	✓	3	70	
4.1.3	Status Message	1,337	0	1,337	—			no published ACT rules
2.4.6	Headings and Labels	1,228	0	1,228	✓	2	30	
1.4.10	Reflow	1,181	0	1,181	—			rule maps to 1.4.4; included there
1.3.5	Identify Input Purpose	730	132	862	✓	1	30	
1.4.13	Content on Hover or Focus	685	0	685	—			no published ACT rules
3.3.1	Error Identification	668	0	668	✓	1	9	
1.4.12	Text Spacing	657	15	672	✓	3	62	
1.3.3	Sensory Characteristics	570	0	570	✓	1	21	
2.2.2	Pause, Stop, Hide	560	0	560	✓	1	11	
2.4.1	Bypass Blocks	532	2,001	2,533	✓	3	34	
2.5.3	Label in Name	495	32	527	✓	1	15	
2.2.1	Timing Adjustable	381	22	403	✓	1	15	also maps to 2.2.4, 3.2.5
2.1.2	No Keyboard Trap	377	0	377	✓	1	16	

0.12% occurrence threshold: 20 additional SC (with 317 issues or fewer each) are excluded

B Data Partition

We partition the dataset at the ACT rule level to distinguish rules that current deterministic checkers can fully decide from those that still require judgment. We classify a rule as *machine-verifiable* when at least one of the five benchmarked checkers achieves perfect performance across the rule's full test suite: 1.0 recall, zero false positives, and no execution errors.

This process yields 28 machine-verifiable rules with 448 cases and 132 expected failures (Table 3), and 21 human-judgment rules with 324 cases and 98 expected failures. Two test pages appear in both partitions because each is used by one rule of each type. The partition is based only on the observed behavior of the five benchmarked checkers and does not depend on our system's outputs.

The human-judgement rules contain 42.6% of all expected failures, yet no deterministic checker exceeds 18.4% recall on this subset (Table 1).

Table 3: The 28 machine-verifiable ACT rules of the 49-rule dataset against the five deterministic checkers. ● = full coverage (recall 1.0, zero false positives, zero errors on the rule's test suite).

SC	ACT rule	Cases (fail)	axe	IBM	HTML_CS‡	alfa‡	QualWeb
1.1.1	`23a2a8` image-name	18 (5)	●			●	●
	`7d6734` svg-name	10 (4)	●			●	●
	`8fc3b6` object-name	18 (6)	●	●		●	●
+4.1.2	`59796f` image-button-name	12 (3)	●	●		●	●
1.3.1	`a25f45` headers-attr	19 (4)	●	●			●
	`ff89c9` required-context-role	15 (4)	●	●		●	●
	`d0f69e` th-assigned-cells	16 (3)					●
1.3.5	`73f2c2` autocomplete-valid	30 (10)	●			●	●
1.4.4	`b4f0c3` meta-viewport	16 (7)				●	●
1.4.12	`24afc2` letter-spacing	19 (4)		●		●	●
	`9e45ec` word-spacing	19 (4)		●		●	●
	`78fd32` line-height	24 (6)		●			●
2.1.1	`akn7bn` iframe-tab-order	10 (1)					●
	`0ssw9k` scrollable-focusable	15 (2)	●				●
2.2.1[a]	`bc659a` meta-refresh	15 (4)	●				●
2.4.2	`2779a5` page-title-exists	13 (6)					●
+4.1.2	`c487ae` link-name (2.4.4)	28 (11)	●	●			●
2.5.3	`2ee8b8` label-in-name	15 (5)		●			●
4.1.2	`674b10` role-valid	11 (2)	●	●			●
	`97a4e1` button-name	17 (5)	●	●		●	●
	`e086e5` form-field-name	22 (9)	●				
	`m6b1q3` menuitem-name	8 (2)	●			●	●
	`307n5z` presentational-children	12 (5)	●			●	●
	`6cfa84` aria-hidden-focusable	15 (6)					●
	`4e8ab6` required-states-props	14 (5)	●	●		●	●
	`5c01ea` aria-prop-permitted	17 (2)	●	●			●
	`2t702h` summary-name	12 (3)				●	
	`cae760` iframe-name	11 (4)	●			●	●
Rules fully covered (of 28)			18	13	0	15	26

‡ alfa and HTML_CodeSniffer expose no ACT identifiers and are scored by SC overlap.
[a] `bc659a` also maps to SC 2.2.4 and 3.2.5.

C System Implementation

C.1 Deterministic Component

We combine three deterministic checkers: QualWeb, axe-core, and IBM Equal Access. We adopt a checker's verdict only for the specific requirement it evaluates and only when its output can be attributed at the required level.

We selected these three from the five deterministic checkers in our benchmark based on two requirements. First, a decision must be attributable to the requirement under evaluation. Alfa and HTML_CodeSniffer do not report ACT rule identifiers. Their findings can therefore be matched only through Success Criterion overlap and cannot be safely restricted to the specific rule under test. Second, any false positive produced by a checker is propagated directly to the final output. Checkers must therefore have very low false-positive rates. On our testing, QualWeb produces 1 false positive, axe-core produces 6, and IBM Equal Access produces 7. In comparison, Alfa produces 28 and HTML_CodeSniffer produces 41. In our experiments, adding Alfa and HTML_CodeSniffer as barriers increased false positives without deciding any additional rule.

The three selected checkers also provide complementary coverage. Of the 28 machine-verifiable rules, QualWeb fully decides 26 and axe-core decides 18, including one rule not covered by QualWeb. IBM Equal Access decides 13, all of which are covered by the other two checkers, but provides additional corroborating evidence. Together, the three checkers cover 27 of the 28 rules, deciding 53.0% of expected failures at 98.4% precision. The remaining rule, `summary-name`, is handled by the LLM-based pathway, along with other candidate cases.

C.2 Tools and Rubrics

Table 4 details 29 scan and agentic tools we developed in our system: 15 *scan tools* that run before any LLM involvement, and 14 *agentic tools* that the agent may call during analysis. The deterministic component also provide supporting findings to several rubrics. Every tool in the table is used by at least one rubric. The reported line counts include each tool's dedicated implementation. Three branch-probing tools share a small dispatcher, listed in the footnote.

Table 5 details the 21 rubrics used for agentic analysis. These rubrics cover 14 of the 20 Success Criteria in the corpus. The remaining six criteria (1.3.5, 1.4.4, 1.4.12, 2.1.1, 2.2.1, and 2.4.1) are handled entirely by the deterministic component and do not require rubric-based agent analysis.

Table 4: Interactive tools that collect evidence in our system, referenced by rubrics in Table 5.

#	Tool	LOC	Purpose
Scan			
S1	Look alike character finder	60	Finds characters that look like ordinary letters but are actually different symbols.
S2	List markup collector	69	Gathers how list content is marked up on the page.
S3	Table markup collector	60	Gathers how table rows, columns, and header cells are wired together.
S4	Keyboard travel recorder	387	Moves through the page with the keyboard to record the tab order, any places where focus gets stuck, and whether focus stays where expected.
S5	Solid background contrast probe	282	Measures the contrast of text when it sits over a single solid background colour.
S6	Picture of text probe	20*	Checks whether a small image is in fact rendering words rather than a picture.
S7	Long description presence probe	19*	Checks whether a complex image offers a longer description somewhere.
S8	Form error visibility probe	106	Submits a form with bad input and checks whether an error is actually shown to the reader.
S9	Name and state probe	69	Checks an element's spoken name and its required on or off states.
S10	Grouped field probe	15*	Checks whether fields that belong together, such as a split phone number, are grouped as one.
S11	Focus indicator probe	124	Checks whether a visible focus marker appears when the keyboard moves to an element.
S12	Keyboard trap escape probe	80	Tries documented ways to move focus out of a place where it appears to be stuck.
S13	Arrow key trap probe	87	Drives the arrow keys inside a composite control to see whether focus can leave it.
S14	Visible label inside name probe	40	Checks whether the words a reader sees on a control are contained in the name a screen reader speaks.
S15	Moving content pause probe	60	Checks whether content that moves or updates on its own can be paused, stopped, or hidden.
Agentic			
A1	Text over image contrast measurer	70	Measures the worst contrast underneath every letter when text sits over a picture or a colour blend.
A2	Accessibility node inspector	162	Reads a chosen element's computed screen reader role, where its name comes from, its required states, and, for a table cell, the header text tied to it.
A3	Single activation observer	132	Activates one control and reports what text newly appears, whether it lands in a region that announces updates, and whether focus moves.
A4	Screen reader announcement capture	77	After a control is activated, captures the exact words a screen reader would speak.
A5	Interaction sequence driver	222	Performs a short sequence of real actions such as typing, clicking, hovering, dragging, or pressing keys on a fresh copy of the page and reports what changed.
A6	State screenshot capture	94	Forces an element into a state such as focus, hover, checked, or open and returns before and after pictures together with the style change.
A7	High magnification crop	24	Renders a small element again at higher magnification so faint text or a tiny mark becomes readable.
A8	Image text reader	38	Reads text that is baked into an image so it can be compared against the written description.
A9	Image region colour comparer	83	Checks whether two named parts of an image are noticeably different in colour.
A10	Two colour contrast calculator	92	Computes the exact contrast between two solid colours, such as text against its background.
A11	Rendered pixel colour reader	75	Reads the true rendered colour of a small part or of the exact background beneath letters.
A12	Link destination follower	174	Follows a link to where it actually lands after any redirect and compares the real destinations of links that share a name.
A13	Embedded frame content comparer	31	Opens two embedded frames that share a name and compares what each one actually contains.
A14	Whole page screenshot	43	Captures the whole page beyond the visible area to confirm where an element such as a heading sits.

*plus a 33-line dispatcher shared by S6, S7, and S10.

Table 5: Rubrics in the study scope and the tools each rubric uses, indexed according to Table 4.

SC	Rubric	Scan	Agentic	SC	Rubric	Scan	Agentic
1.1.1	Alt text adequacy	S1 S6	A2 A7 A8 A9	2.2.2	Motion control	S15	—
	Long description completeness	S7	A2 A7 A8 A9	2.4.2	Page title	—	A14
	Decorative image verification	S6	A2 A7 A8 A9	2.4.4	Link purpose	—	A2 A12
	CAPTCHA alternative	—	A7 A8 A9		Link name equivalence	—	A2 A12
1.3.1	Information and relationships	S2 S3	A2 A14	2.4.6	Heading descriptive	—	A2 A14
	Field programmatic association	S2 S3	A2 A14	2.4.7	Focus visible and clear	S11	A6
1.3.3	Sensory characteristics	—	A14	2.5.3	Label inside name	S14	A2
1.4.3	Contrast over complex backdrop	S5	A5 A6 A10 A1 A11	3.3.1	Error identification	S8	A3 A4 A5
1.4.5	Images of text	—	A5 A7 A8	4.1.2	Accessible name adequacy	S9 S10	A2 A13
2.1.2	Keyboard trap	S4 S12 S13	A3 A5		Duplicate name equivalence	S9 S10	A2 A13
					Automatic update notification	S9 S10	A2 A3 A4 A13

D Additional Results

We evaluate 770 unique ACT test cases, including 230 expected failures, from 49 rules spanning 20 WCAG Success Criteria. All LLM models were evaluated with Claude Sonnet 4.6. All metrics are computed per test case. Figure 2 shows precision–recall plot for all checkers. Arrows indicate the change introduced by adding the LLM component for AccessGuru and our system. Adding AccessGuru's LLM component increases recall but substantially reduces precision. In our system, the LLM agent increases recall from 53.0% to 99.1%, while precision decreases from 98.4% to 91.2%. We also plot checker capability by SC (Table 6), where our system achieves full coverage for 18 of the 20 SCs.

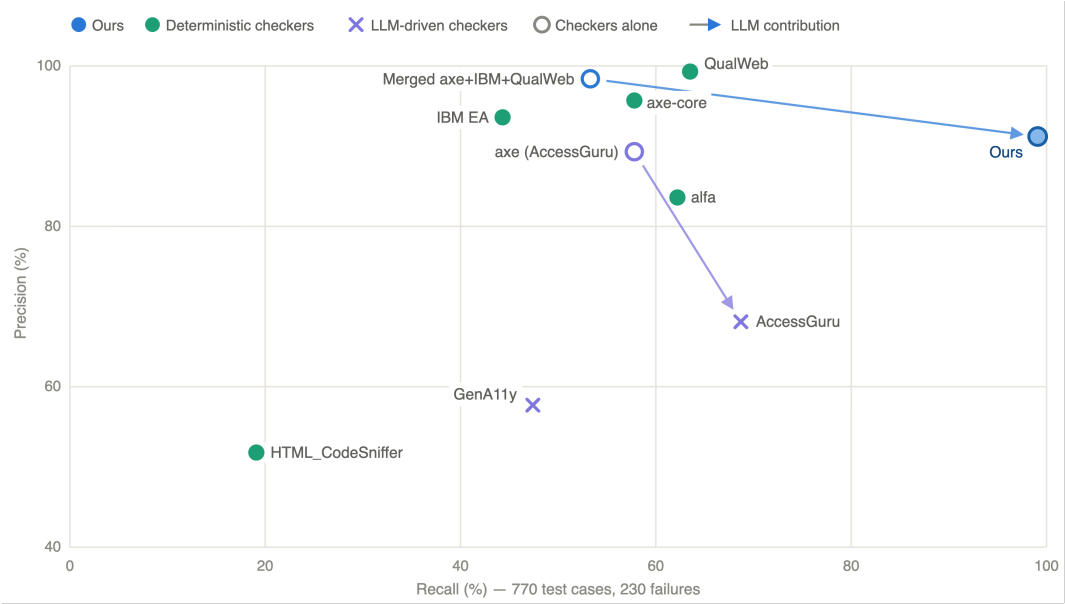

Figure 2: Recall and precision of the evaluated accessibility checkers. Open circles show AccessGuru and our system's deterministic component, and arrows show the change after adding its LLM component.

Table 6: Observed ACT-rule coverage by WCAG Success Criterion. ● = full, ◐ = partial, – = none.

SC	Success criterion	Cases (fail)	axe [2]	IBM [7]	HTML_CS [15]	alfa [14]	QualWeb [8]	AccessGuru [5]	GenA11y [6]	Ours
1.1.1	Non-Text Content	94 (26)	◐	◐	◐	◐	◐	●	●	●
1.3.1	Info and Relationships	74 (21)	◐	◐	◐	◐	◐	●	●	●
1.3.3	Sensory Characteristics	21 (4)	–	–	–	–	–	–	–	●
1.3.5	Identify Input Purpose	30 (10)	●	◐	–	●	●	●	–	●
1.4.3	Contrast (Minimum)	34 (11)	◐	–	◐	◐	◐	●	●	●
1.4.4	Resize Text	30 (12)	◐	–	–	◐	◐	–	–	●
1.4.5	Images of Text	15 (5)	–	–	–	–	–	–	●	◐
1.4.12	Text Spacing	62 (14)	◐	●	–	◐	●	●	–	●
2.1.1	Keyboard	25 (3)	◐	–	–	◐	●	●	–	●
2.1.2	No Keyboard Trap	16 (5)	–	–	–	–	–	–	–	●
2.2.1	Timing Adjustable	15 (4)	●	–	◐	◐	●	●	–	●
2.2.2	Pause, Stop, Hide	11 (1)	–	–	–	–	–	–	–	●
2.4.1	Bypass Blocks	34 (7)	–	–	◐	–	–	◐	–	●
2.4.2	Page Titled	20 (9)	◐	◐	◐	◐	◐	●	●	●
2.4.4	Link Purpose (In Context)	70 (25)	◐	◐	◐	◐	◐	●	●	◐
2.4.6	Headings and Labels	30 (10)	–	–	–	–	–	●	●	●
2.4.7	Focus Visible	9 (1)	–	–	–	–	–	–	–	●
2.5.3	Label in Name	15 (5)	–	●	–	◐	●	◐	–	●
3.3.1	Error Identification	9 (5)	–	–	–	–	–	–	●	●
4.1.2	Name, Role, Value	211 (66)	◐	◐	◐	◐	◐	●	●	●
Full + partial (of 20)			2+8	2+6	0+8	1+11	5+7	11+2	9+0	18+2

Full = every ACT rule of the SC fully covered; partial = some rules covered or some detections; none = zero detections.