Paradigm Academic Press
Innovation in Science and Technology
ISSN 2788-7030
SEP. 2026 VOL.5, NO.3

Research on Government Web Accessibility Automated Auditing Technology Based on Static Analysis and DOM Rule Engine

Xincheng Liu[1]

[1] The University of Sydney, NSW 2006, Australia

Correspondence: Xincheng Liu, The University of Sydney, NSW 2006, Australia.

doi:10.63593/IST.2788-7030.2026.09.006

Abstract

Aiming at the industry pain-points of high labour costs for manual compliance auditing of domestic government Web accessibility, dynamic DOM omission detection, incomplete static source-code parsing, poor adaptability of general-purpose rules to specific scenarios, and high false-positive rates of mainstream detection tools, this paper proposes an automated auditing framework for government Web accessibility that integrates static code analysis and a DOM rule engine. This paper constructs a layered rule ontology library compatible with the WCAG 2.1-AA international standard and domestic government accessibility specifications, designs a DOM rule engine with multi-node association constraint reasoning capability, and builds a multi-source static-analysis pipeline covering HTML, CSS and JavaScript. A defect-entity alignment algorithm and a weighted confidence fusion model are adopted to eliminate detection conflicts between the two modules, so as to output graded and standardised auditing results. System verification is performed relying on a ground-truth dataset of 427 manually annotated government webpages (Fischer T, Lundell B & Gamalielsson J., 2025). The experimental results show that compared with mainstream tools such as axe-core and Lighthouse, the precision, recall and F1-score of the proposed method reach 87.2%, 84.5% and 85.8% respectively. Its F1-score is 9.7% higher than that of axe-core, and its false-positive rate decreases by 11.3% (Gunderson J & Schepers D., 2021). The proposed method can accurately identify accessibility defects of government-specific components such as government forms, pop-up windows and data tables, and provides reliable technical support for compliance construction and inclusive transformation of government-website accessibility.

Keywords: web accessibility, static code analysis, DOM rule engine, government websites, automated auditing, WCAG 2.1, defect fusion reasoning, ontology rule library, government digital inclusiveness

1. Introduction

1.1 Research Background

Digital government affairs serve as a critical carrier of public inclusive services. The accessibility of government websites directly safeguards the rights and interests of people with disabilities and elderly groups to equally obtain government information and conduct online administrative services. Although the accessibility construction of Chinese government webpages has been steadily advanced, practical dilemmas remain for large-scale compliance evaluation. Manual auditing delivers reliable results; nevertheless, a complete evaluation for a single website requires 12-18 working hours (Cooper M, Kirkpatrick A, O'Connor J, et al., 2018), leading to exorbitant labour costs for batch inspection of massive websites. Existing open-source detection tools suffer from dynamic DOM omission detection, insufficient adaptation to government-oriented scenarios and high false-positive rates, which render them incompatible with the requirements of government compliance auditing. Hence, it is urgent to develop a special-purpose automated-detection technical system for government Web platforms.

1.2 Domestic and Overseas Research Status

Overseas research on Web accessibility detection started relatively early, giving rise to engineered detection frameworks including axe-core and Lighthouse. These frameworks can realise automatic verification for WCAG 2.1-AA clauses, yet they are mostly oriented toward commercial webpages and pay insufficient attention to business components such as government forms, pop-up windows and complex tables. Most domestic studies reuse overseas open-source engines for empirical investigations. Customised rule-ontology modelling targeting government-oriented scenarios remains scarce, and research on integration schemes combining static source-code analysis and runtime DOM detection is inadequate. Moreover, high-quality manually annotated ground-truth datasets for government webpages are lacking, which restricts the parameter tuning and comparative verification of algorithms.

1.3 Existing Problems

Combined with the above-mentioned domestic and overseas research status, several key technical weaknesses persist in current automated-detection practices for government Web accessibility:

1) Pure static analysis fails to recognise DOM nodes dynamically generated by JavaScript, resulting in prominent omission-detection phenomena for frequently-used government components such as asynchronous pop-up windows and dynamically-rendered forms.
2) General-purpose accessibility rule libraries are not optimised and adapted for government-specific business components, yielding high false-positive rates in government-oriented scenarios.
3) The two systems of static analysis and runtime DOM detection operate independently, lacking mechanisms for defect-entity alignment and conflict resolution, and detection confidence cannot be quantified.
4) Publicly available manually-annotated ground-truth datasets for government-webpage accessibility are scarce, which hinders algorithm parameter tuning and comparative verification.

1.4 Research Innovations

Targeting the above-mentioned technical defects, this study conducts research from four dimensions: rule-library construction, detection engine, result fusion and dataset construction. The primary innovations are summarised as follows:

1) A layered rule-ontology library adapted to WCAG 2.1-AA and domestic government specifications is constructed, with 28 additional special-purpose detection rules for government businesses to supplement detection capabilities for components such as forms, pop-up windows and government tables.
2) A DOM rule engine with multi-node association-constraint reasoning is designed. Combined with the multi-source static-analysis pipeline, bidirectional feature extraction at both the source-code layer and runtime layer is achieved.
3) A defect-entity alignment algorithm and a weighted confidence fusion model are proposed to resolve conflicts in detection results from dual modules and output standardised auditing results graded by confidence levels.
4) A manually annotated dataset containing 427 government-webpage samples is built. Comparative experiments and ablation experiments are carried out to verify the detection performance of the proposed framework in government-oriented scenarios.

2. Relevant Theoretical Foundations and Key Technologies

2.1 Web Accessibility Standards and Defect-Classification System

Core Specifications of WCAG 2.1-AA: Centring on four principles — Perceivable, Operable, Understandable and Robust — the standard establishes detection constraints covering more than 50 detectable sub-clauses, serving as the universal technical benchmark for global Web-accessibility evaluation. On the basis of WCAG 2.1-AA, domestic technical specifications for government-website accessibility supplement localised constraint clauses for business components such as government announcements, administrative forms, data tables and file-download controls. Explicit clause-mapping relationships exist between the two sets of specifications.

Accessibility defects of government webpages are divided into three major categories: tag-semantics defects, style-interaction defects, and business-component-specific defects for government affairs. Statistical results of defect distribution for public government defect samples are shown in Table 1.

Table 1. Statistics on Defect-Type Distribution of Government-Webpage Accessibility Defects

Defect Category	Number of Sample Defects	Proportion	Typical Defect Examples	Risk Level

Tag-semantics Defects	312	42.7%	Missing alt text for images, missing form label tags	High Risk
Style-interaction Defects	246	33.7%	Insufficient colour contrast ratio, inaccessible keyboard focus	Medium-to-High Risk
Government-business-specific Defects	172	23.6%	Pop-up-window focus hijacking, missing table headers for government tables	Medium Risk

Government-business-specific defects rarely occur on ordinary Internet webpages and constitute the primary source of missed detections by general-purpose detection tools.

2.2 Principles of Web Static-Code Analysis Technology

HTML/CSS Static Parsing: Lexical and syntactic parsing generates an Abstract Syntax Tree (AST) for webpages. Tag information, attributes and style declarations inside source code are directly read without browser-rendering workflows. The average parsing time for a single webpage is merely 18-42 ms (Ara J & Sik-Lanyi C., 2025). The inherent limitation of static analysis lies in JavaScript dynamic logic: DOM nodes generated through interfaces such as createElement and innerHTML do not exist in the initial source-code AST. Relying purely on static source code cannot obtain the real-time node status after webpage execution.

Boundary Constraints for JS Static Analysis: JS static analysis can identify risky code fragments, yet it cannot simulate complete execution logic or judge whether business branches are actually triggered. It can only output potential-risk markers instead of directly judging real-page defects.

2.3 Basic Theory of DOM Rule Engines

Differences between Static DOM and Runtime DOM: Static DOM corresponds to the parse tree derived from original HTML source code; runtime DOM is generated after JavaScript execution and style computation, reflecting the real browsing status perceived by end-users. A rule engine consists of a rule-definition module, a pattern-matching module and a reasoning-output module. Open-source projects including axe-core and ARIA-rules adopt chain-matching mechanisms, which excel at single-node condition verification but exhibit weak multi-node association-constraint reasoning capabilities. Their matching performance degrades for scenarios such as label-input binding in forms and complex nested government tables.

The fundamental logical-expression definition for rule matching is given below:

$$R=\left\{C_i/_{i=1\ldots m},D_j/_{j=1\ldots n}\right\}$$

Where: R denotes the complete rule triggering a defect; C_i represents self-attribute constraint conditions of nodes; D_j denotes constraint conditions of associated nodes; m is the number of constraints for the current node; n is the number of constraints for associated nodes. An accessibility defect is judged to exist for the DOM node only when all constraint conditions are satisfied simultaneously.

2.4 Horizontal Comparison of Technical Indicators for Mainstream Detection Tools

Three mainstream automated accessibility-detection tools, namely axe-core, Lighthouse and WAVE, are selected for horizontal comparison in terms of technical mechanisms, quantity of built-in rules and adaptability for government-oriented scenarios, as presented in Table 2.

Table 2. Technical Comparison of Mainstream Web-Accessibility Detection Tools

Tool Name	**Detection Technical Mechanism**	**Number of Built-in Rules**	**Customised Rules for Government Components**	**False-positive Rate in Government-oriented Scenarios**
axe-core	Runtime DOM Rule Matching	132	0	21.7%
Lighthouse	Static + DOM Hybrid Detection	94	0	24.1%
WAVE	DOM Rendering	107	0	20.3%

	Detection			

As indicated by Table 2, none of the existing tools are equipped with special-purpose rules for government business components, yielding false-positive rates generally higher than 20 % for government webpages.

3. Construction of Layered Rule Ontology Library for Government Web Accessibility

3.1 Rule-Demand Analysis for Government Webpages

Characteristics of Government-webpage Components: Government websites frequently feature business components including administrative-application forms, announcement pop-up windows, statistical-data tables, paginated navigation and attachment-download controls. Such components contain abundant JavaScript dynamic-rendering logic, which cannot be covered by general-purpose detection rules. Detection rules are divided into three categories: source-code static-verifiable rules, runtime-DOM verification rules and manual-review prompt rules. Static-verifiable rules can be directly judged via HTML/CSS source code; DOM-verification rules require access to post-rendering nodes and computed styles of webpages; manual-review prompt rules address subjective-perception-oriented issues, for which automated technology only outputs prompt markers requiring secondary manual confirmation.

Multi-dimensional Constraints for Rules: A complete detection rule incorporates six dimensions of constraints: tag-semantics constraints, attribute constraints, style-threshold constraints, keyboard-interaction constraints, ARIA-semantics constraints and government-business-scenario constraints.

3.2 Ontology Modelling for Accessibility Rules

The ontology model contains seven core entities: defect category, detection object, constraint condition, standard source, scenario tag, risk level and baseline weight. Relationships among entities are categorised as mapping, inheritance and association. Mapping is completed from all verifiable sub-clauses of WCAG 2.1-AA and domestic government-accessibility-specification clauses to ontology-rule instances. The rule baseline weight w_{base} is defined: high-risk defects are assigned weights within the interval $[0.75,1.0]$, medium-risk defects within $[0.45,0.74]$, and low-risk defects within $[0.20,0.44]$.

3.3 Rule-Description Language and Rule Grading Implementation

Design of Rule-description Language: JSON structured format is adopted to describe rules, supporting logical combinations of AND, OR and NOT, as well as configurable threshold parameters such as contrast ratio and element dimensions, which facilitates subsequent rule iteration and expansion. Twenty-eight customised rules for government-oriented scenarios are supplemented, covering special-purpose business scenarios including pop-up-window focus management, header association for complex government tables and dynamic-verification prompts for administrative forms.

Rules are graded into four levels: high-risk defects, medium-risk defects, low-risk defects and prompt-type issues. High-risk defects directly block users with disabilities from accessing business functions; medium-risk defects significantly degrade accessibility experience; low-risk defects denote optimisation-oriented problems; prompt-type issues only serve as clues for manual review.

3.4 Mechanism for Rule-Conflict and Redundancy Elimination

Condition overlap and contradictory judgements may emerge between multiple rules, which easily trigger duplicate alarms. A rule-conflict-detection algorithm is designed to traverse the complete rule set and compare intersections of rule-constraint-condition sets. Two rules are marked as conflicting if they target identical input objects yet yield opposite judgement results; a rule is marked as redundant if its constraint set constitutes a subset of that of another rule. In total, 176 rules are verified in the constructed rule library: 8 redundant rules are eliminated and 3 groups of conflicting rules are resolved, leaving 165 valid rules. Statistics on the composition of the rule library are provided in Table 3.

Table 3. Composition Statistics of Government-Accessibility Rule Ontology Library

Rule Type	**Quantity of Rules**	**Source Description**
Rules Mapped from WCAG 2.1-AA	117	Converted from international-standard clauses
Rules Mapped from Domestic Government Specifications	20	Converted from domestic government-accessibility-specification clauses

Customised Rules for Government-oriented Businesses	28	Customised rules for pop-up windows, government forms, tables and other components

4. Design and Implementation of the DOM Rule Engine

4.1 Overall Architecture of the DOM Rule Engine

The DOM rule engine adopts a five-layer hierarchical architecture: DOM input layer, node pre-processing layer, rule-matching scheduling layer, conditional reasoning layer and defect-metadata output layer. After browser rendering, the complete runtime-DOM tree of a webpage is fed into the engine. Invalid nodes are filtered and cleaned in pre-processing. Rule-library scheduling executes multi-constraint-condition reasoning and outputs standardised defect results.

4.2 DOM Pre-Processing Module

DOM-tree Cleaning and Filtering: Comment nodes, script nodes and completely hidden nodes with the display: none attribute are filtered out, retaining visual business-component nodes to reduce computational overhead for subsequent matching. For DOM fragments locally generated by JavaScript, sub-tree slicing and extraction are implemented. Instead of re-rendering the complete webpage, rule verification is only executed for partial DOM sub-trees, reducing the average detection time for local fragments by 34.6%.

Multi-dimensional feature extraction for DOM nodes is completed. The feature set includes tag names, attribute sets, computed CSS styles, ARIA attributes, as well as parent-child and sibling-node association relationships. Node features are output in a structured format to provide feature input for subsequent rule matching.

4.3 Multi-Condition-Combined Rule-Matching and Reasoning Algorithm

Three-tier matching logic is implemented on the basis of the rule-logical-expression formula:

1) **Single-node attribute matching**: On the basis of the rule-logical-expression formula, attribute-condition matching and verification are performed for individual DOM nodes on webpages.
2) **Multi-node association-constraint matching**: Cross-node verification is implemented to handle association-constraint scenarios such as input-label binding and table-header-cell association.
3) **Logical-condition reasoning**: Logical reasoning operations are carried out to judge composite conditions including AND, OR and NOT.

A rule-priority scheduling strategy is introduced. High-weight high-risk rules are executed first. Pruning operations are performed for irrelevant sub-branches upon successful matching to avoid unnecessary traversal. For government webpages containing fewer than 1 200 nodes, the pruning strategy reduces engine-matching time by **21%-29%**.

4.4 Defect Localisation and Metadata Output

The output includes the DOM path of defects, defect descriptions, corresponding standard-clause numbers, repair suggestions, risk levels and initial engine confidence scores. CSS selectors are adopted to record DOM paths, enabling precise localisation of defective nodes on webpages and supplying matching evidence for source-code traceback in the static-analysis module.

4.5 Engine-Performance Optimisation Test

Five groups of government webpages with varying node scales are selected for engine-performance testing. Test results are listed in Table 4.

Table 4. Detection-time Test of the DOM Rule Engine for Webpages

Total Quantity of DOM Nodes	Webpage Type	Engine Detection Time (ms)
326	Government Announcement Page	76
614	Government List Page	124
957	Administrative-form Page	187
1243	Comprehensive Government Portal Page	241
1682	Multi-component Nested Page	316

Detection time increases approximately linearly with rising node quantities, satisfying time-overhead requirements for automated auditing of conventional government webpages.

5. Multi-Source Information Static-Analysis Pipeline Technology

1) Overall Framework of the Static-analysis Pipeline

The static-analysis pipeline consists of webpage source-code acquisition, multi-file parsing, AST construction, source-code feature extraction, static-rule matching and static-defect-output modules. It accepts original HTML, CSS and JavaScript source code of webpages as input, constructs Abstract Syntax Trees for respective file types, and mines source-code-level potential accessibility risks without browser-rendering execution. Structured defect data are output in a standardised format whose fields are fully aligned with those output by the DOM rule-engine module.

2) HTML Source-code Static-analysis Module

Lexical and syntactic parsing is performed for HTML source code to generate HTML-AST. AST nodes are traversed to extract tags, attributes and metadata. Defects determinable at the static stage such as missing alt attributes, missing form tags and misused semantic tags can be directly identified. A key merit of this module lies in its capability to record source-code line numbers and achieve precise localisation of defective code lines.

3) CSS Static-analysis Module

CSS stylesheets are parsed to generate CSS-AST. Colour values, font parameters, display, visibility and other style attributes are extracted. Colour-contrast features are pre-computed at the static stage. Nevertheless, such computations are only based on original source-code styles and cannot capture styles dynamically modified by JavaScript. Hence static contrast-ratio results serve merely as reference features; final judgements adopt runtime contrast ratios computed by the DOM rule engine.

4) JavaScript Source-code Static-analysis Module

JavaScript Abstract Syntax Trees are constructed to identify invocation positions of dynamic-DOM-generation interfaces such as createElement, innerHTML and appendChild. High-risk code fragments including focus hijacking, keyboard-event interception and dynamic pop-up-window generation are detected. It should be clarified that static analysis only recognises code patterns and cannot simulate business-execution branches. Accordingly, all outputs from the JavaScript module are marked as potential risks with confidence values within the range 0.4-0.6. Real defects must be jointly judged in combination with runtime-detection results from the DOM rule engine.

5.1 Standardised Output of Static-Analysis Results

A unified defect-output structure is defined with fields covering defect type, risk level, source-code line number, DOM-path reference, static-module confidence score and standard-clause number. These fields are consistent with outputs from the DOM rule engine, laying a unified data foundation for defect-entity alignment and fusion computation described in Chapter 6.

6. Dual-Module Fusion Auditing Strategy for Static Analysis and DOM Rule Engine

6.1 Capability Comparison Between Dual Detection Modules

The static-analysis module can localise defects to source-code lines and identify potential code-level risks in advance. However, it cannot acquire real-time webpage status after JavaScript execution and suffers notable omission-detection for dynamically-generated DOM. The DOM rule engine performs detection on the basis of browser-rendered runtime-DOM trees, consistent with actual end-user access effects. Yet it struggles to trace back to original source-code fragments and may produce false alarms under certain scenarios. Three types of situations arise for results output by the two modules: the same defect is detected by both modules; a defect is detected solely by one single module; conflicting judgements are returned by the two modules. Therefore, a fusion strategy must be constructed to normalise detection results.

Table 5. Capability Comparison between Static-code-analysis Module and DOM Rule-engine Module

Comparison Dimension	Static Code-analysis Module	DOM Rule-engine Module
Detection Basis	AST parsed from original HTML/CSS/JavaScript source code of webpages	Runtime-DOM tree after browser rendering
Capability for Dynamic-DOM Identification	Weak; cannot parse nodes generated by JavaScript	Strong; can capture asynchronously-rendered components
Defect-localisation	Can localise down to source-code line	Only outputs DOM selector paths

Granularity	numbers	
Primary Advantages	Early mining of source-code risks, fast execution speed	Consistent with real-user access status
Primary Limitations	Cannot simulate script-execution logic	Difficult to trace back to original code fragments
Typical Output Defect Types	Missing tag attributes, static-style anomalies	Pop-up-window focus issues, form binding anomalies, contrast-ratio anomalies

6.2 Defect-Entity Alignment Algorithm

Defects corresponding to one real defect instance may differ in textual descriptions and field formats when output by different modules; accordingly defect-entity matching and alignment must be accomplished. Weighted multi-feature similarity computation is adopted for defect-entity matching. The similarity-computation formula is defined as:

$$S=\alpha S_{path}+\beta S_{type}+\gamma S_{clause}$$

Where: S denotes the total similarity of defect entities; S_{path} represents DOM-path similarity; S_{type} denotes defect-type similarity; S_{clause} is the similarity of corresponding standard clauses; α, β and γ are weight coefficients satisfying $\alpha+\beta+\gamma=1$. In this paper, $\alpha=0.45$, $\beta=0.35$, $\gamma=0.20$. A similarity threshold $T=0.72$ is set. When $S\geq T$, two defects output by different modules are judged to belong to the same real-defect instance and merged.

6.3 Conflict Resolution and Weighted Confidence-Computation Model

The fusion module integrates the static-module confidence score $Conf_s$, DOM-rule-engine confidence score $Conf_d$, rule baseline weight w_{base} and government-scenario coefficient λ to compute the final output confidence $Conf_{final}$:

$$Conf_{final}=w_{base}\cdot\lambda\cdot(\delta Conf_s+(1-\delta)Conf_d)$$

Where δ is the module-evidence weight coefficient; the government-scenario coefficient satisfies $\lambda\in[0.85,1.15]$. Values greater than 1 are adopted for government-specific-rule scenarios, and $\lambda=1$ for general-webpage scenarios. If a defect is detected by only one module and no corresponding result is returned by the other module, the score of the missing module is set to zero. When conflicting judgements are returned by the two modules, the final confidence is forcibly constrained within the interval 0.4-0.6 and marked as a sample requiring manual review.

Confidence-grading thresholds are defined as follows: $Conf_{final}\geq 0.70$: marked as **confirmed defect**; $0.40\leq Conf_{final}<0.70$: marked as **suspected defect**, requiring manual review; $Conf_{final}<0.40$: judged as **non-defect**.

6.4 Complete Automated-Auditing Workflow

System input: webpage URL or local webpage source code.

1) On one hand, input is sent to the static-analysis pipeline to obtain a set of static defects.
2) On the other hand, page rendering is triggered, runtime DOM is extracted and fed into the DOM rule engine to obtain a set of DOM defects.
   - Defect-entity alignment is executed to merge duplicate defects.
   - Conflict resolution and confidence computation are performed via the confidence model.
   - Confirmed defects, suspected defects and defect-free items are output in accordance with risk levels, generating structured JSON auditing reports and visualised reports.

7. Prototype-System Implementation and Experimental Verification

7.1 Prototype-System Implementation

The back-end of the prototype system is developed with mixed Python + NodeJS technologies. The webpage-rendering module is built upon Playwright. The static-analysis module invokes HTML-parser, CSS-AST and ESTree for multi-source-code parsing. The DOM rule engine is independently developed following the architecture described in Chapter 4 without reliance on third-party rule components such as axe-core. The system supports three input modes: single-URL detection, batch-website detection and local-webpage-source-code detection. It outputs JSON-formatted structured reports as well as HTML visualised auditing reports. The system consists of five sub-modules: webpage-collection sub-module, static-analysis sub-module, DOM-rule-engine

sub-module, fusion-processing sub-module and report-generation sub-module.

7.2 Experimental Dataset and Evaluation Metrics

The dataset contains 427 government-webpage samples collected from multi-level domestic government websites, manually annotated by two Web-accessibility-specialist researchers to form Ground-Truth ground-truth labels. Among them, 183 samples contain government-specific business components, and the total number of defects in dataset samples reaches 839. The dataset is partitioned at a ratio of 7:3 into a training-and-parameter-tuning subset and a test subset. The parameter-tuning subset determines hyper-parameters including similarity thresholds and confidence thresholds. All comparative experiments are conducted on the test subset to avoid parameter over-fitting.

Adopted evaluation metrics include Precision, Recall, F1-score, false-positive rate, false-negative rate and average single-page detection time.

7.3 Comparative Experiments

Comparative experiments are performed for the proposed method alongside axe-core, Lighthouse and WAVE on the identical test subset. Experimental outcomes are presented in Table 6.

Table 6. Performance-comparison Results for Multiple Detection Schemes

Detection Scheme	**Precision / %**	**Recall / %**	**F1-score / %**	**False-positive Rate / %**
axe-core	77.4	75.1	76.1	21.7
Lighthouse	74.8	72.3	73.5	24.1
WAVE	76.2	73.6	74.8	20.3
Proposed Method	87.2	84.5	85.8	10.4

Comparative-experiment results demonstrate that the F1-score of the proposed method is 9.7 percentage points higher than axe-core (the optimal baseline), while its false-positive rate drops by 11.3 percentage points. For government-specific defects such as government pop-up windows and complex government tables, the recall rates of other tools range merely from 62.1 % to 68.4 %, whereas the recall rate of the proposed method attains 81.2 %, manifesting substantially enhanced detection capability for government-business components. General-purpose tools lack customised rules for government-oriented businesses, yielding insufficient recognition for business-specific defects and generating numerous invalid alarms, which push up false-positive metrics.

7.4 Ablation Experiments

Three ablation-experiment groups are set up to verify the contributions of the two core modules and the fusion strategy: Group A: static-analysis pipeline only; Group B: DOM rule engine only; Group C: complete fusion scheme of this paper. Ablation-experiment results are summarised in Table 7.

Table 7. Ablation-experiment Results

Experiment Group	**Precision / %**	**Recall / %**	**F1-score / %**
A Static Analysis Only	71.3	64.7	67.8
B DOM Rule Engine Only	80.1	76.4	78.2
C Complete Fusion Scheme	87.2	84.5	85.8

The static-analysis-only group delivers the lowest recall rate owing to massive missed detections for dynamic-DOM defects. The DOM-rule-engine-only group exhibits insufficient precision because of false alarms originating from the absence of source-code-layer evidence constraints. After dual-module fusion, precision, recall and F1-score all achieve prominent improvements, proving complementary capabilities between static analysis and the DOM rule engine and validating the practical performance gain brought by the fusion strategy.

7.5 Empirical Testing on Real-World Government Websites

Batch empirical auditing is performed for 12 domestic government-portal and administrative-service webpages.

In total, 342 confirmed defects and 117 suspected defects are detected (Ara J, Sik-Lanyi C & Kelemen A., 2024). Statistics of high-frequency defects are as follows: insufficient colour contrast ratio accounts for 27.2 %, missing alt text for images accounts for 21.3 %, abnormal header association for government tables accounts for 14.0 %, and abnormal pop-up-window focus management accounts for 10.2 %. Empirical-test results reveal that business-component-specific accessibility defects constitute a considerable proportion in government webpages and deserve priority attention for subsequent website-transformation work.

8. Research Limitations and Future Prospects

8.1 Research Limitations

Highly subjective semantic accessibility defects still cannot be fully automatically judged. Such issues include textual-reading logic and rationality of voice-prompt semantics, which are related to subjective-experience dimensions and hard to distinguish by automated algorithms. Only review markers can be output, mandating manual intervention by specialists. For government-website pages with extremely complex asynchronous fragment rendering, a small number of boundary-scenario missed detections emerge due to constraints of the DOM-fragment-extraction mechanism. Furthermore, the current rule library mainly covers general-purpose government business components, leaving detection-rule gaps for some niche custom-built business components. Subsequent work will continuously expand samples and rules to further enhance framework adaptability.

8.2 Future-Work Prospects

Four directions for further research are proposed to improve the technical system for government-Web-accessibility auditing:

1) **Intelligent Defect Reasoning and Optimised Repair**: Introduce large-language models to perform semantic reasoning and judgement for complex suspected defects and overcome difficulties in automatic identification of semantic-oriented defects. Generate standardised repair schemes tailored to government-webpage business characteristics to boost implementation efficiency for accessibility transformation.
2) **Performance Optimisation for Batch Detection**: Optimise concurrent scheduling and resource allocation for large-scale websites, streamline redundant computations in DOM parsing and static analysis, and cut time and hardware costs for massive-page detection to satisfy requirements for regular batch inspection of government platforms.
3) **Expansion to Multi-terminal Detection Scenarios**: On the basis of PC-side detection capabilities, add rendering-adaptation and accessibility-detection rules for mobile webpages to build a cross-terminal full-scenario automated-auditing system for government webpages.
4) **Open-source Construction of Standardised Datasets**: Expand manually-annotated samples and component-defect data for government-webpage accessibility, and build large-scale high-coverage open standard datasets to supply benchmarks for algorithm iteration and scientific research within this field.

References

Ara J, Sik-Lanyi C, Kelemen A. (2024). Accessibility engineering in web evaluation process: a systematic literature review. *Universal Access in the Information Society, 23*(2), 653-686. https://doi.org/10.1007/s10209-023-00967-2.

Ara J, Sik-Lanyi C. (2025). Automated evaluation of accessibility issues of webpage content: tool and evaluation. *Scientific Reports, 15*(1), 1-14. https://doi.org/10.1038/s41598-025-92192-5.

Cooper M, Kirkpatrick A, O'Connor J, et al. (2018). Web Content Accessibility Guidelines (WCAG) 2.1. World Wide Web Consortium(W3C), 2018-06-05[2026-05-10]. https://www.w3.org/TR/WCAG21/.

Fischer T, Lundell B, Gamalielsson J. (2025). Coverage of web accessibility guidelines provided by automated checking tools. *Universal Access in the Information Society, 24*(4), 3615-3637. https://doi.org/10.1007/s10209-025-01263-x.

Gunderson J, Schepers D. (2021). Automated web accessibility evaluation: a systematic review. *ACM Transactions on Accessible Computing, 14*(3), 1-32. https://doi.org/10.1145/3468874.

Copyrights

Copyright for this article is retained by the author(s), with first publication rights granted to the journal.

This is an open-access article distributed under the terms and conditions of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/4.0/).