

DRUPALCON EUROPE 2026 · ROTTERDAM

Can AI Fix the Web?

The promises and perils of automated remediation



Junaid Masoodi

Country Manager – Bitbase India



BEFORE YOU START

This was a talk. Here is the argument.

Delivered at DrupalCon Europe 2026 in Rotterdam on 29 September. This version is written to be read on your own, every claim that was spoken from the stage is now on the slide that carries it.

The question

Can AI fix web accessibility? Vendors say yes. The data says something more specific, and more useful.

The short answer

Not from inside your user's browser at runtime — that is architecturally impossible. But it helps a developer a great deal at authoring time.

Who it is for

Front-end developers and architects, and the agency owners who decide what gets sold to clients.

Every figure in this deck is sourced on the slide that uses it, and the full reference list is at the end. Where a number is disputed in the field, the deck says so rather than picking the flattering one.

THE SCOREBOARD

Three years of AI promises. Here is what happened to the web.

95.9%

of the top 1,000,000 home pages fail an
automated WCAG scan

up from 94.8% in 2025

56.1

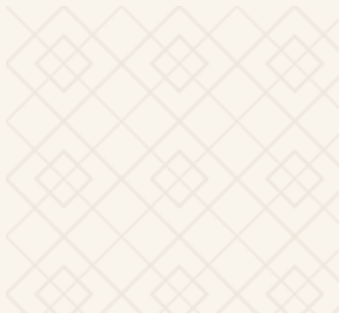
detectable errors on the average home page

up 10.1% year on year

6,000+

US digital accessibility lawsuits projected
for 2026

~20% above 2025



THE UNCOMFORTABLE PART

**96% of those errors are in six categories.
A machine can find every one of them.**

Low contrast text	83.9%
Missing image alt text	53.1%
Missing form input labels	51.0%
Empty links	46.3%
Empty buttons	30.6%
Missing document language	13.5%

So here is the question.

We are getting worse at the problems
automation can already see.

**Why would we believe it is about to
solve the ones it can't?**





The false promise of the one-click fix

What the marketing says. What the architecture does.



You have all seen this advert.

“

Add one line of JavaScript.

Pay monthly.

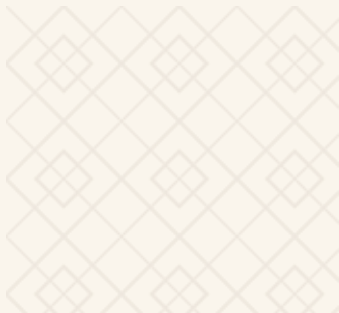
Your entire legacy platform is WCAG 2.2 AA compliant by tomorrow.

No developers. No audit. No remediation backlog.

It is the best-sounding proposition in our industry.

- It removes a budget line nobody wants to defend
- It removes a conversation with the client about scope
- It removes an audit that would embarrass the last five years of delivery

And it does not work.



THE CEILING

Forget the percentage. Look at the list.

You will hear that automation catches 30% of WCAG issues. You will also hear Deque's 2025 study putting axe-core at 57% - one counts success criteria, the other counts issue volume. I do not want to argue about the denominator.

Nobody argues about this list:

Is the alt text meaningful?

A machine confirms the attribute exists. Never that it describes the right thing.

Is the reading order logical?

DOM order can be perfectly valid and the narrative still incoherent.

Can the flow be completed?

Checkout, booking, benefits appeal end to end, with a screen reader.

Is the error recoverable?

Detected, announced, understood, and fixable by the person who hit it.

Does focus land somewhere sensible?

After a modal, a route change, a validation failure, an injected region.

Does the name match the label?

2.5.3 — partly automatable, and routinely wrong in practice.

THE ARCHITECTURAL FLAW

An overlay is a race condition you pay for monthly.

1

Your HTML ships broken

Missing landmarks, unlabelled inputs, div-based controls.

2

The browser builds the a11y tree

The screen reader has already begun reading it.

3

The overlay JavaScript wakes up

And starts rewriting the DOM underneath both of them.

4

The framework re-renders

React, Angular and Vue change state independently. The repair is gone.

You have not fixed the accessibility tree. You have added a second writer to it, one that loses.

WHO SAYS SO

1,031 signatories. Including the people who build your screen readers.

Signed by accessibility engineers at

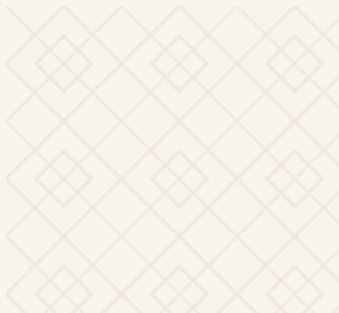
**Google · Microsoft · Apple · Shopify
ServiceNow · NBC**

Researchers at MIT, Carnegie Mellon and Gallaudet. Contributors to the WAI-ARIA specification. Disability rights counsel including Lainey Feingold.

And contributors to JAWS and NVDA themselves.

What they state overlays cannot repair

- Text alternatives for images, not reliably
- Field labels, error handling, focus control
- Keyboard navigation repairs
- Anything inside React, Angular or Vue state changes
- PDF, Canvas, SVG and media content





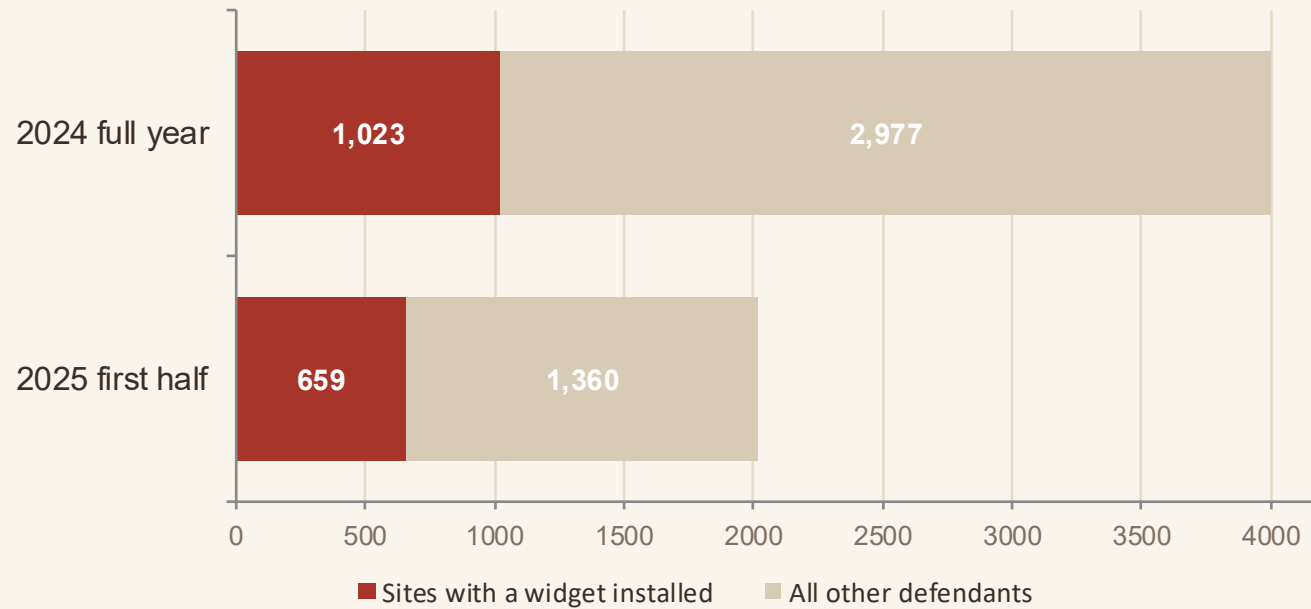
The legal reality

Where the overlay stops being bad engineering and starts being liability.



THE FILING DATA

The widget is not a shield. It is a flag.



1 in 4

of all 2024 US digital accessibility suits hit a site that had already bought a widget.

1 in 3

in the first half of 2025.

The ratio is moving in the wrong direction.

THE REGULATOR ARRIVES

FTC v. accessiBe

3 Jan 2025

Complaint filed

The FTC alleges accessWidget could not do what its marketing claimed.

22 Apr 2025

Final order approved

Commission vote 3 to 0. US\$1 million monetary judgment.

Now

The order stands

Binding on how automated remediation may be sold.

What the order actually bars

Claiming an automated product can make any website WCAG-compliant or can ensure continued compliance over time without competent and reliable evidence.

There was a second allegation.

accessiBe published third-party reviews of its own product as independent opinion, while paying, or being commercially connected to, the reviewers, without disclosing it.

Why this matters more than the million dollars

The evidence base was manufactured

If you researched overlays in 2022 and came away reassured, some of what reassured you was paid for.

The order covers endorsements too

It bars misrepresenting endorsers as independent, ordinary users or objective organisations.

Apply the test to AI a11y tooling now

The same playbook is running again, with 'AI-powered' in place of 'automated'. Ask who paid for the review.

AND IN EUROPE

The European Accessibility Act has been in force since 28 June 2025.

Who it reaches

Private-sector digital services - e-commerce, banking, transport, e-books, telecoms.

The standard

EN 301 549, which incorporates WCAG 2.1 level AA.

The penalties

Set nationally. Italy: up to 5% of turnover.
Germany: up to €100,000. France: €50,000.

Under the ADA, the overlay is a bad defence. Under the EAA, someone at your client has to sign an accessibility statement - resting on a product the FTC has already said cannot substantiate a compliance claim.

A DIGRESSION, AND THEN THE POINT

A ceiling built with no nails and no glue.

This is khatamband - a Kashmiri ceiling craft. Hundreds of small pieces of walnut and deodar, cut so that they interlock and hold by geometry alone.

No nails. No adhesive. Any single panel can be lifted out and replaced without disturbing the ones around it. There are ceilings in Srinagar built this way three hundred years ago that are still being repaired, panel by panel, today.

That is a design system.

An overlay is a sheet nailed over the top of one.



Overlays rent accessibility. Source code owns it.

RENT

- Subscription never ends and never amortises
- Nothing accrues to your codebase
- Risk stays with your client, and with you
- You inherit the vendor's regulatory exposure

OWN

- Fixed once, in the component, in the repo
- Every future project starts from a better baseline
- Defensible under audit, litigation and the EAA
- The fix survives the next redesign



Sheeting, or joinery. That is the whole business case - and it is not really about accessibility.



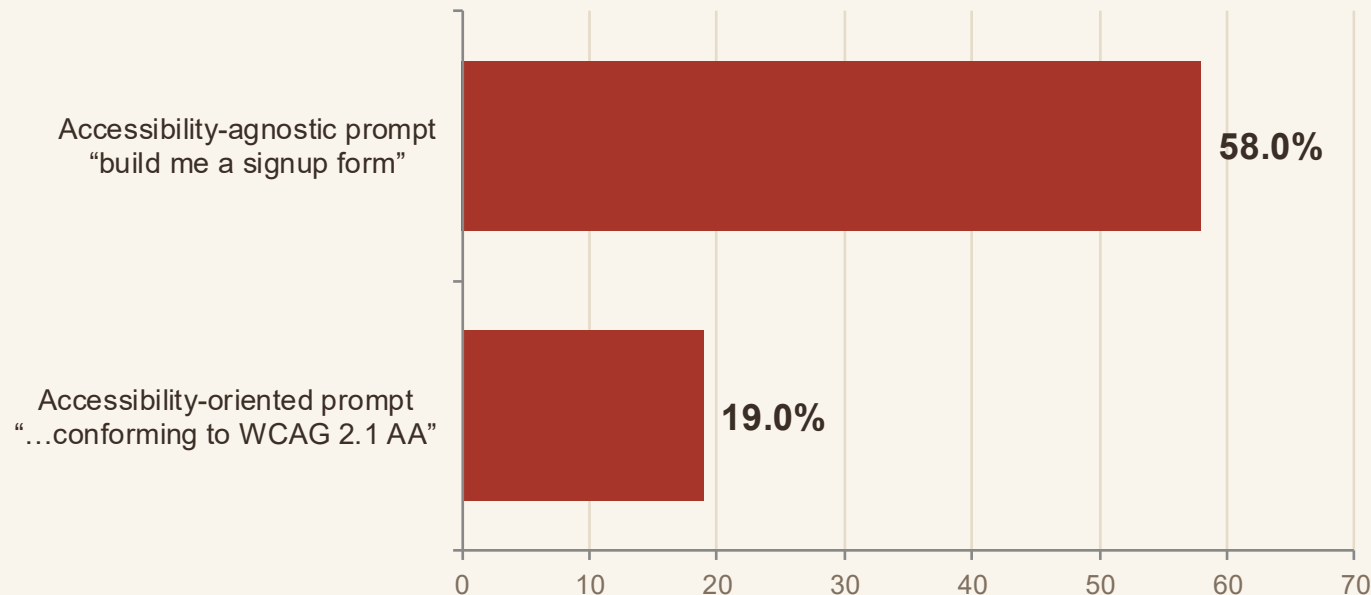
Where AI fails: hallucinated empathy

The model is not lying to you. It is doing statistics on markup.



PEER-REVIEWED, NOT ANECDOTE

The variable is not the model. It is the prompt.



80 generated interfaces

Claude 3.5 Haiku and GPT-4-turbo, expert-evaluated against WCAG.

Both halves of the argument, in one table.

AI writes inaccessible code by default and much better code when someone who knows WCAG is driving.

And it still fails one time in five.

ASK IT PLAINLY, GET THIS

What fails when nobody mentions accessibility



of generated interfaces contained a violation of that criterion

Every one of these is a criterion your users feel and your linter cannot see.

SYNTACTICALLY CORRECT, SEMANTICALLY BROKEN

This passes your linter.

TYPICAL AI OUTPUT

```
<div class="custom-btn"
  role="button"
  aria-label="Click here to submit form"
  tabindex="0">
  Submit
</div>
```

WHAT IT SHOULD BE

```
<button type="submit">
  Submit
</button>
```

What the validator cannot see

- No keydown handler - Enter and Space do nothing
- Not in the form's submit path
- aria-label overrides the visible text (fails 2.5.3)
- No native disabled, focus ring or form association

Four lines of ARIA to poorly re-implement an element the platform already gave you correctly, and for free.

RECORDED DEMO · NVDA + FIREFOX

The same button, as a person hears it.

AI-GENERATED MARKUP

“Click here to submit form, button”

[user presses Enter]

— *silence* —

[user presses Space]

“Page scrolled”

NATIVE BUTTON

“Submit, button”

[user presses Enter]

“Booking confirmed”

[focus moves to the confirmation]

“Your reference is 4471”

axe-core reports zero violations on the left-hand version.

THE ALT-TEXT TRAP

Vision models describe. Accessibility names.

	WHAT IS IN THE IMAGE	WHAT THE IMAGE IS FOR
A search icon	“Black magnifying glass on a grey circular background”	“Search”
A team headshot in an “about us” grid	“A smiling man with brown hair wearing a blue blazer standing in front of a window”	“Junaid Masoodi, Country Manager”
A revenue chart	“A bar chart with five blue vertical bars increasing left to right”	“Revenue by quarter; full figures in the table below”

The model cannot know the purpose, because the purpose is not in the image. It is in the page. And in 100% of accessibility-agnostic generations, this criterion failed.



Where AI actually excels

Move it out of the browser and into the pipeline.



THE PIVOT

Not a runtime patch for the user. A design-time assistant for the developer.

Overlay

Runs in the user's browser. Fixes nothing. Costs monthly, forever. Fails on the next re-render.

Assistant

Runs in the IDE and the pipeline. Fixes the source. Costs once. Survives every future render.

AXE-CORE FAILS THE BUILD · THE MODEL DRAFTS THE PATCH · A HUMAN REVIEWS IT

```
- name: Accessibility audit
  run: npx axe-cli https://staging.example.com --json > ally-results.json

- name: Draft remediation for review      # opens a PR. Never merges.
```

SAFE TO DEPLOY ON MONDAY

Four places this genuinely pays

1 Pattern analysis across a portfolio

Ten thousand audit rows across forty client sites, clustered into the twelve component defects that actually cause them. The use case nobody talks about, and the highest ROI of the four.

2 Draft patches in the pull request

The model reads the failing DOM snapshot and proposes a diff. A developer reviews it like any other PR. Nothing merges unreviewed.

3 Baseline alt text at scale

Twelve thousand legacy media items. AI drafts, an editor corrects. Ninety percent of the typing disappears; the judgement stays human.

4 Inline authoring feedback

WCAG-tuned prompts in the IDE flagging a missing label or focus state before the commit, when the fix costs minutes rather than a sprint.

Every one of these has a human at the end of it. That is not a limitation of the approach. That is the approach.



The human in the loop

The loop fails at the human step. Design for that.



Four steps. Only one of them is the AI.

1

DETECT

Automated

axe-core and Lighthouse flag the objective failures - contrast, missing attributes, structural violations.

2

PROPOSE

AI-assisted

The model drafts a source-code patch or an ARIA refactor. It opens a pull request. It does not merge.

3

VERIFY

Human, always

Keyboard only: Tab, Shift+Tab, Space, Enter, Esc. Then a real screen reader - VoiceOver, NVDA or JAWS.

4

COMMIT

Source control

The fix lands in the component, in the repository, permanently. Not in a runtime script that reapplies it forever.

Step 3 is not optional, and it is not a code review. Somebody puts their hands on a keyboard and their ears on a screen reader.

When AI-assisted accessibility fails, it fails at the human step.

1

Nobody asked

Developers did not prompt the assistant for accessibility at all. It was never part of the request.

2

Nobody finished

Placeholder attributes the model emitted were never replaced. The scaffold shipped as the implementation.

3

Nobody could check

No practical way to verify whether the result actually conformed. So it was assumed.

Three failure modes. Not one of them is the model's fault. Which is exactly why a protocol beats a better tool.

IF YOU DO ONE THING

Add one line to your pull request template.

Keyboard tested: yes / no

Why this one, and not a tool

It costs nothing

No procurement, no budget line, no vendor evaluation. You can add it this afternoon.

It catches what tools cannot

Focus order, operability, whether the flow can actually be completed - none of which a scanner or a diff can see.

It makes the gap visible

A 'no' in that field is not a failure. It is a question somebody now has to answer out loud.

SO — CAN AI FIX THE WEB?

No. But it can help you fix it.

- Accessibility is an architectural requirement, not an operational afterthought.
- Automation is a detector, not a remediator. Know which one you bought.
- The model is not the variable — the person driving it is.
- Overlays rent accessibility. Source code owns it.

Build the ceiling so that any panel can be lifted out and replaced. That is the craft and no machine has learned it yet.



Where to start, in order.

1

Scan for the six

Run axe-core across your portfolio and count only the six WebAIM categories. They are 96% of detectable errors and they are the cheapest thing you will ever fix.

2

Cluster the findings

Feed the audit export to a model and ask which component generates the most findings. On a shared theme, thousands of findings usually collapse into about a dozen defects.

3

Fix the component, not the page

Repair each defect once, in the component, in the repository. That is the difference between joinery and sheeting.

4

Add the keyboard line

One field in the pull request template. It is the only step here that changes behaviour rather than code.

5

Then, and only then, automate drafting

Let the model propose patches in CI. It opens a pull request. It never merges.

Steps 1 to 3 are a week of work on most portfolios. Step 4 is an afternoon. Step 5 is the only one that needs a budget.

IF YOU WANT TO CHECK MY WORKING

The four studies this argument rests on

Guriță & Vatavu, W4A '25

80 generated interfaces across two frontier models, expert-evaluated against WCAG. Violation rate 58% on plain prompts, 19% when the prompt named WCAG 2.1 AA.

Supports: The 39-point swing, and the residual one-in-five.

Abu Doush & Kassem, 2025

Eleven component patterns benchmarked across GPT-4o, Copilot Pro, Claude 3.7 Sonnet and Grok 3. All produced semantically valid code that failed accessibility without follow-up prompting.

Supports: That the div-with-role pattern is the category default, not one bad generation.

Mowar et al., CodeA11y, CHI '25

Two studies, 36 developers using AI assistants. Three failure modes: nobody prompted for accessibility, nobody replaced placeholder attributes, nobody could verify the result.

Supports: That the loop fails at the human step, not the model.

WebAIM Million, 2026

Automated scan of the top one million home pages. 95.9% with detected WCAG failures, 56.1 errors per page, both worse than 2025. Six categories account for 96% of errors.

Supports: The scoreboard, and the fact that we are losing ground on the easy half.

The litigation counts come from UsableNet's annual reports, the regulatory facts from the FTC's own press releases, and the signatory count from overlayfactsheet.com.

THANK YOU

Sources, and where to argue with me

WebAIM Million 2026

webaim.org/projects/million

Overlay Fact Sheet — 1,031 signatories

overlayfactsheet.com

FTC v. accessiBe, final order April 2025

ftc.gov

UsableNet ADA lawsuit reports

usablenet.com

Guriță & Vatavu, W4A '25 — LLM accessibility barriers

doi.org/10.1145/3744257.3744266

Mowar et al., CodeA11y, CHI '25

arxiv.org/abs/2502.10884

Abu Doush & Kassem 2025 — GAI accessible HTML benchmark

doi.org/10.1007/s10209-025-01250-2

Deque Automated Accessibility Coverage Report

deque.com

Junaid Masoodi

www.junaidmasoodi.com — iam@junaidmasoodi.com

