ISTANBUL
SOFTWARE
TESTING
CONFERENCE

Lessons Learned from AI-Powered Visual Reasoning

Detecting UI issues, accessibility problems, and design inconsistencies with LLMs

Risko Ruus | ISTC 2026

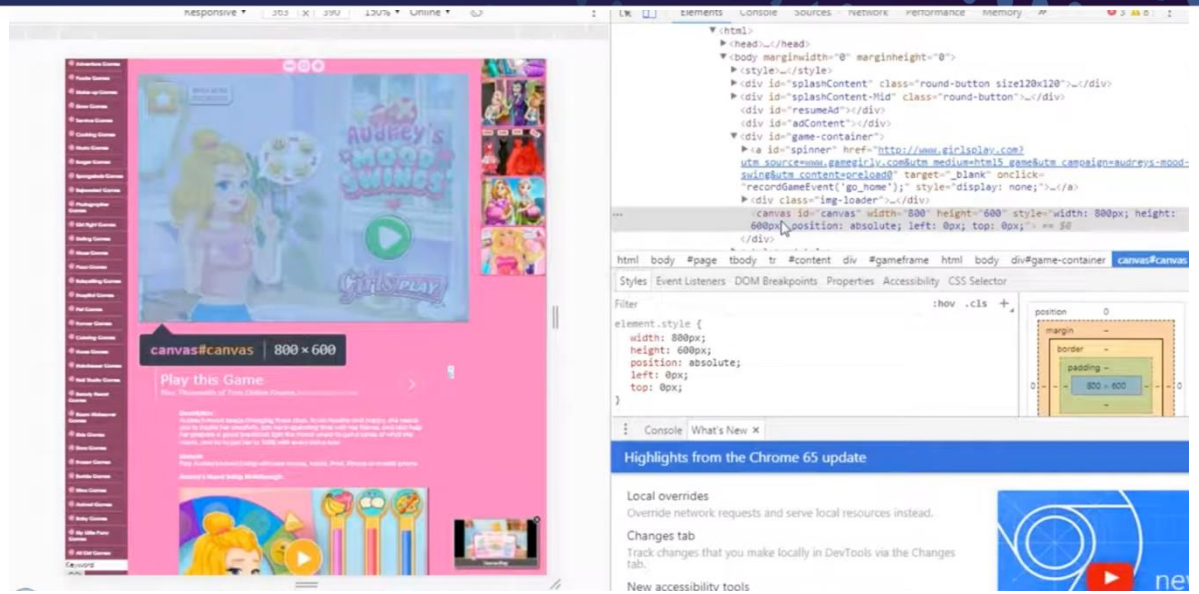
#ISTC2026

2006: Way Back When

Automated Visual Testing
Symbian Operating System

- Minor pixel variations cause assertion failures
- Unique images required for every device
- High maintenance demands

2010: Adobe Flash & Sikuli



Automating Adobe Flash Content with Sikuli (OpenCV)

- Image recognition to identify GUI components
- Flash objects lacked IDs for UI automation
- Integrated it with Selenium WebDriver
- Image maintenance was a burden

2026: Similar Problems Still Today

The Same Problem, New Tech

Web: WebAssembly + Canvas

- App logic runs as opaque WASM bytecode — no DOM, no inspectable state
- UI rendered to a <canvas> — Selenium/Playwright see one element, not buttons or text

Native: Flutter, Rive, game engines, etc.

- App paints its own surface via Skia/Metal/GL — no native view hierarchy
- Appium/Espresso/XCUITest see one container view — widgets inside are invisible

Different stack, same testability gap: no selectors, pixel-based interaction, visual-only validation.

About Me

Risko Ruus

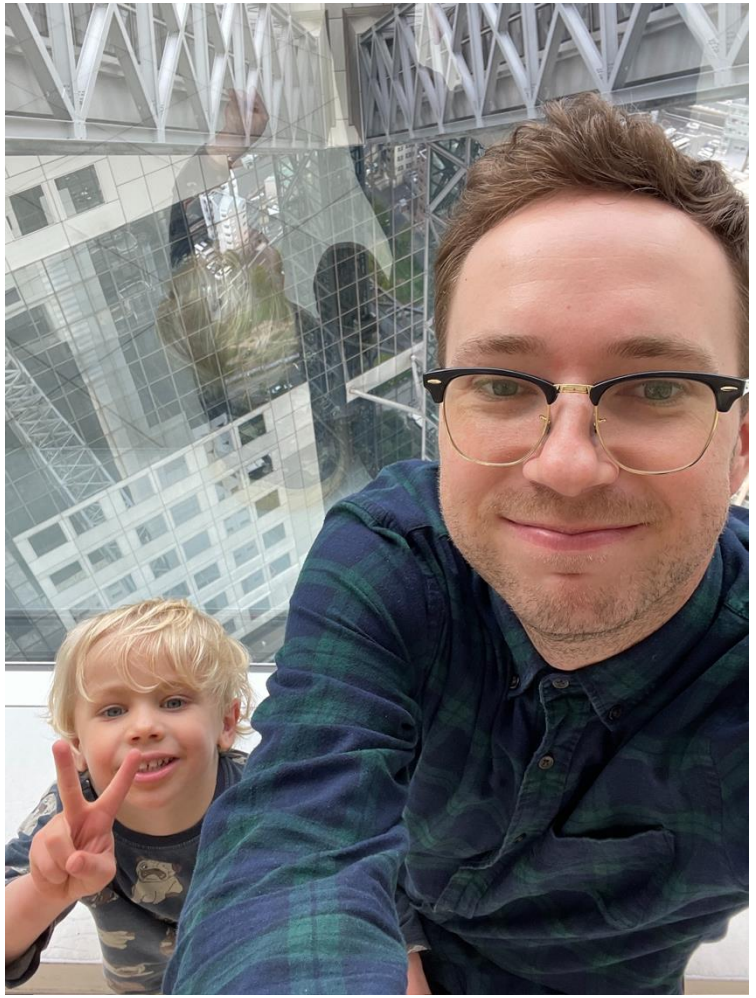
Principal QA Engineer
@Rush Street Interactive

EXPERIENCE

20+ years in QA — from early-stage startups to products at scale

HOBBIES

- Backcountry hiking
- Learning Japanese with my son
- Tinkering with AI



Agenda

01 Traditional UI Assertions

02 Visual AI Reasoning Core Concepts & Basic Usage

03 Advanced Analysis Techniques

04 Integrating Visual Reasoning Checks into Workflows

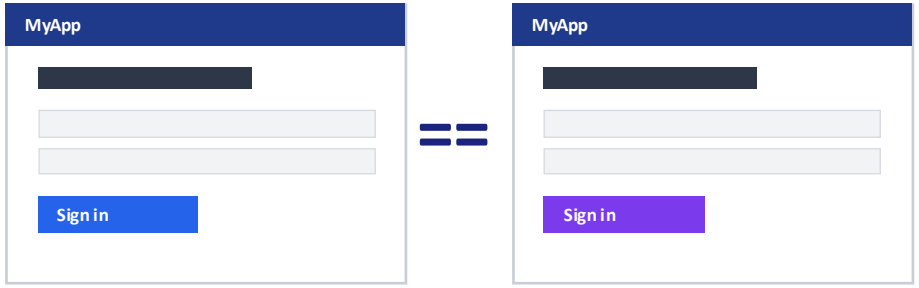
01

Traditional UI Assertions

Traditional UI Assertion Approaches | Limitations

Problems with Traditional Visual Check Approaches

Approach 1: Full Match



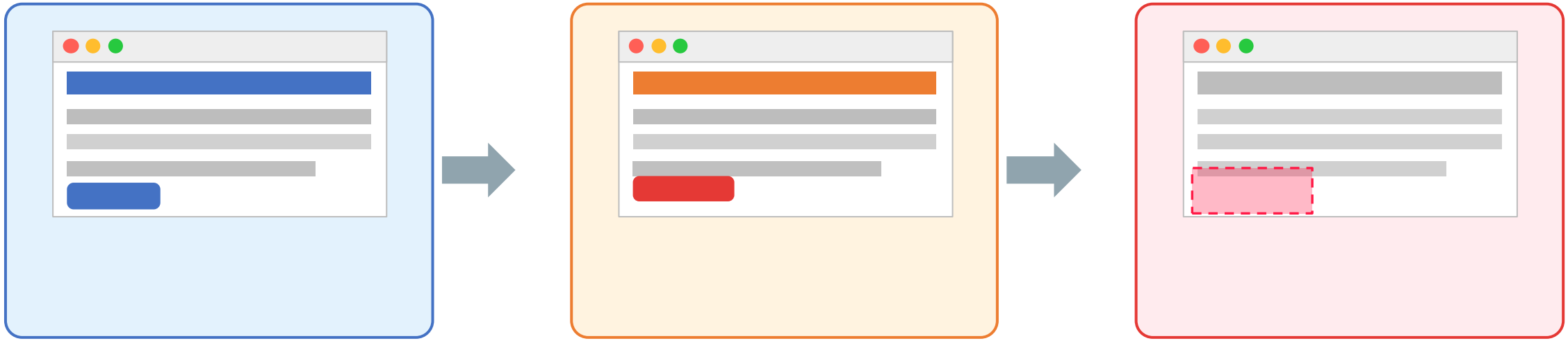
Problems:

- ⚠ Too flaky — fails on minor changes
- ⚠ Font rendering, anti-aliasing cause false failures
- ⚠ Requires constant screenshot updates

High Maintenance | Brittle

How Traditional Pixel Diff Tools Work

Applitools Eyes, Percy, and similar tools



Baseline Screenshot | **Current Screenshot** | **Pixel Diff Output**

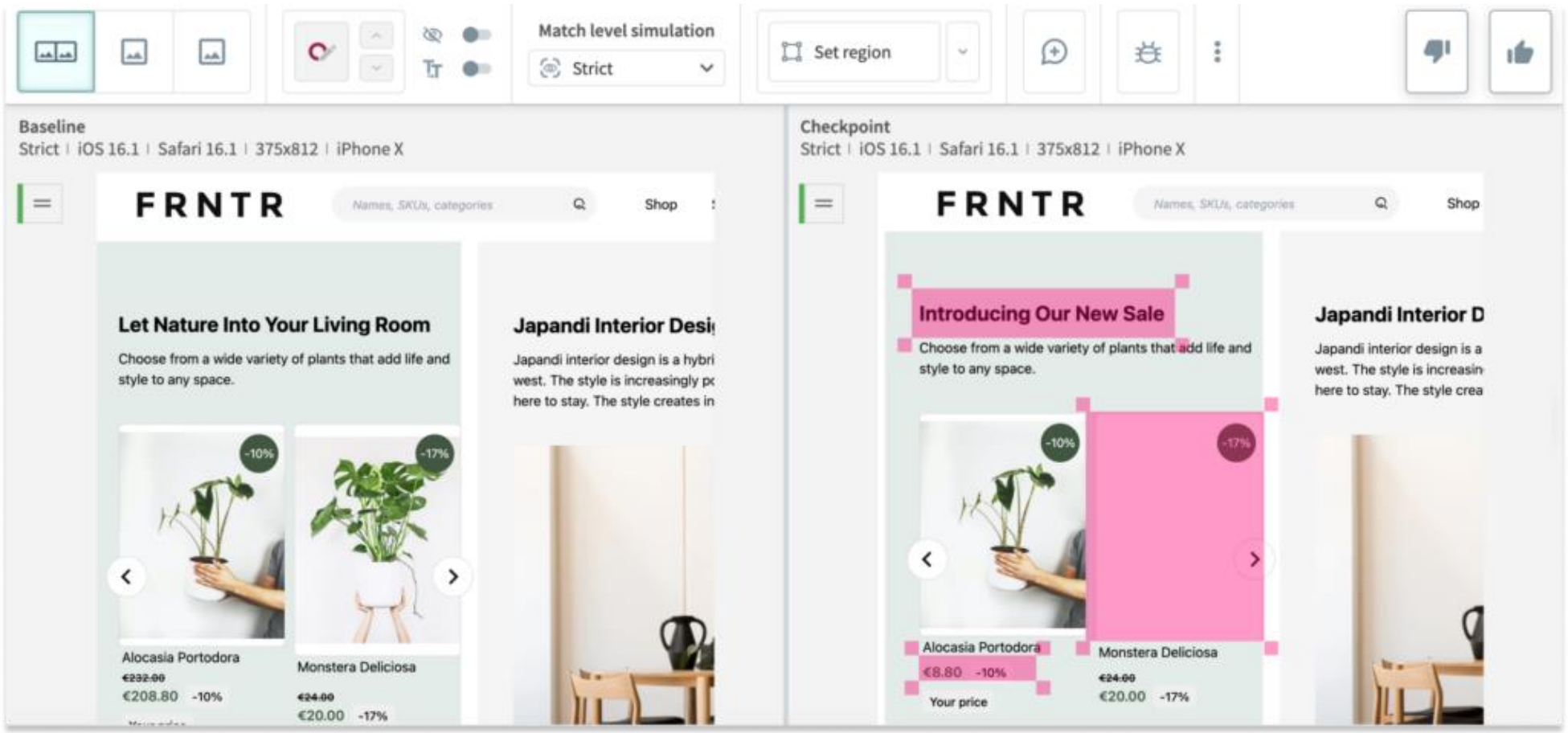
What pixel diff reports:

- "12 pixels changed at region (140, 280)"
- "0.3% visual difference detected"
- "Button color shifted from #44C to #E53"

What it can't tell you:

✕ Is the button still usable?
✕ Does the new color meet contrast standards?
✕ Was this change intentional or a bug?

Example Pixel Diff Output



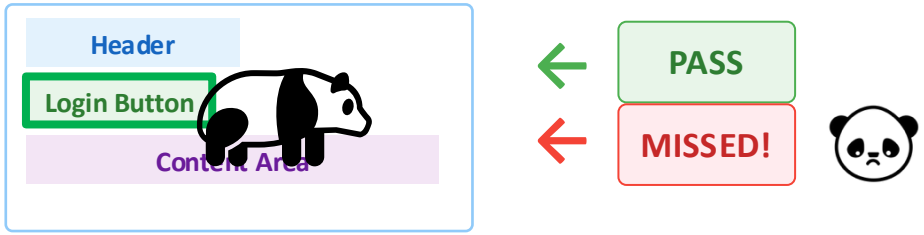
Applitools Eyes Image Comparison

Problems with Traditional Visual Check Approaches

The Giant Panda Problem

Approach 2: Specific Element Check

E.g. verify that a login button is displayed



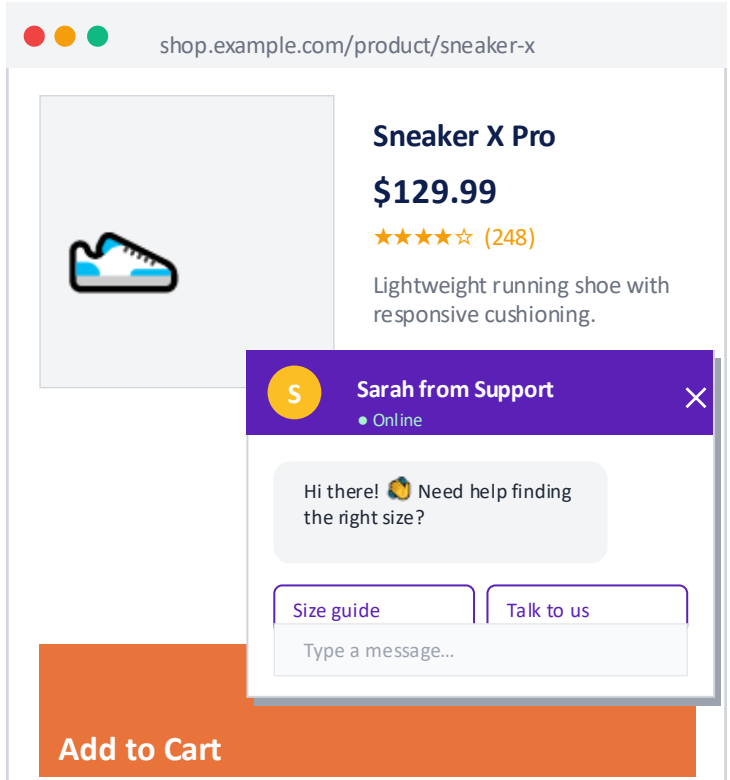
Problems:

- ⚠ Explicit checks miss new erroneous content
- ⚠ Only validates what you coded it to check

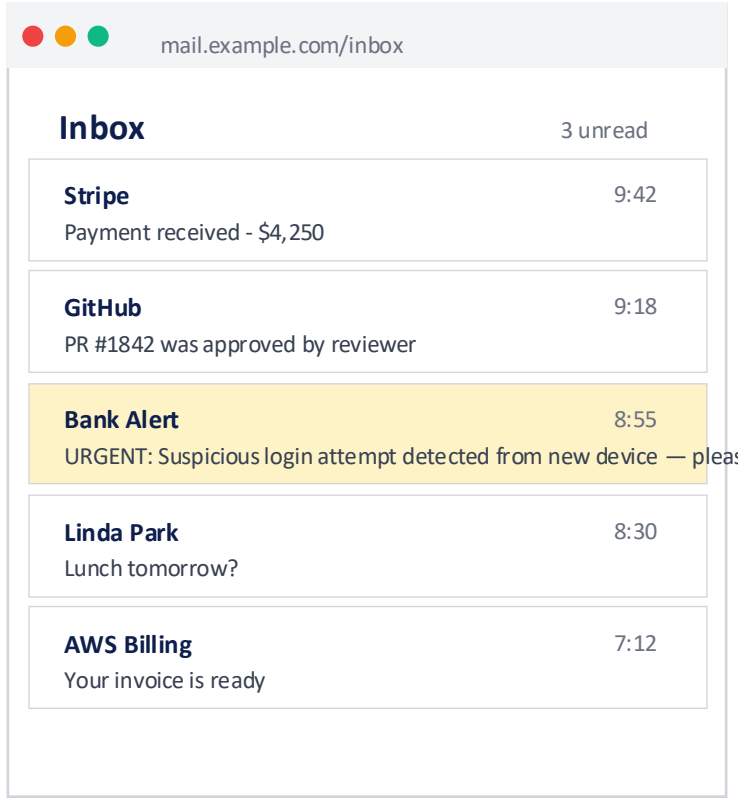
Blind Spots | False Confidence

Real-World Examples: When isDisplayed/isVisible Lies

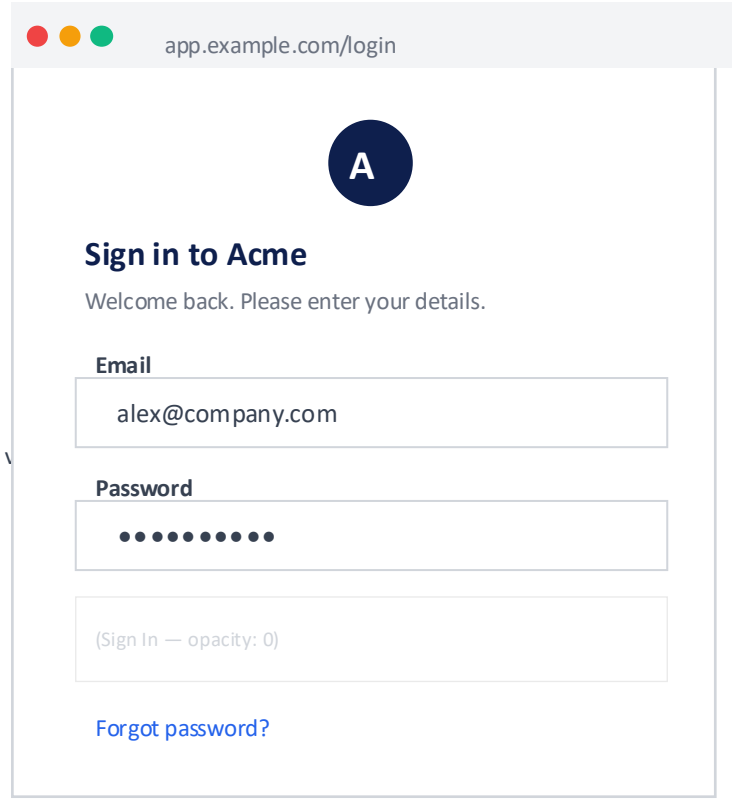
Each page looks fine to a non-vision UI test— but a real user has a poor user experience



Covered by overlay



Clipped by overflow



Opacity: 0

02

Visual AI Reasoning Core Concepts & Basic Usage

Traditional vs AI-powered testing | Analysis types

How the AI Reviews a Screenshot

Three steps from screen image to readable feedback on bugs and design

1. Screenshot + prompt in

Just a screenshot + text
Show the AI a picture with a task of finding issues. No design spec or "before" image needed, but it can be helpful.

2. Visual + semantic check

Looks and understands
The AI reads the image alongside your instructions, spotting visual bugs (overlaps, cut-off text, contrast) and design issues using what it learned during training.

3. Clear feedback out

A list of issues
Returns issues with severity and a short explanation. References shapes or regions by name, not pixel coordinates.

Key insight: *General-purpose LLMs catch many visual and design bugs out of the box — no fine-tuning required.*

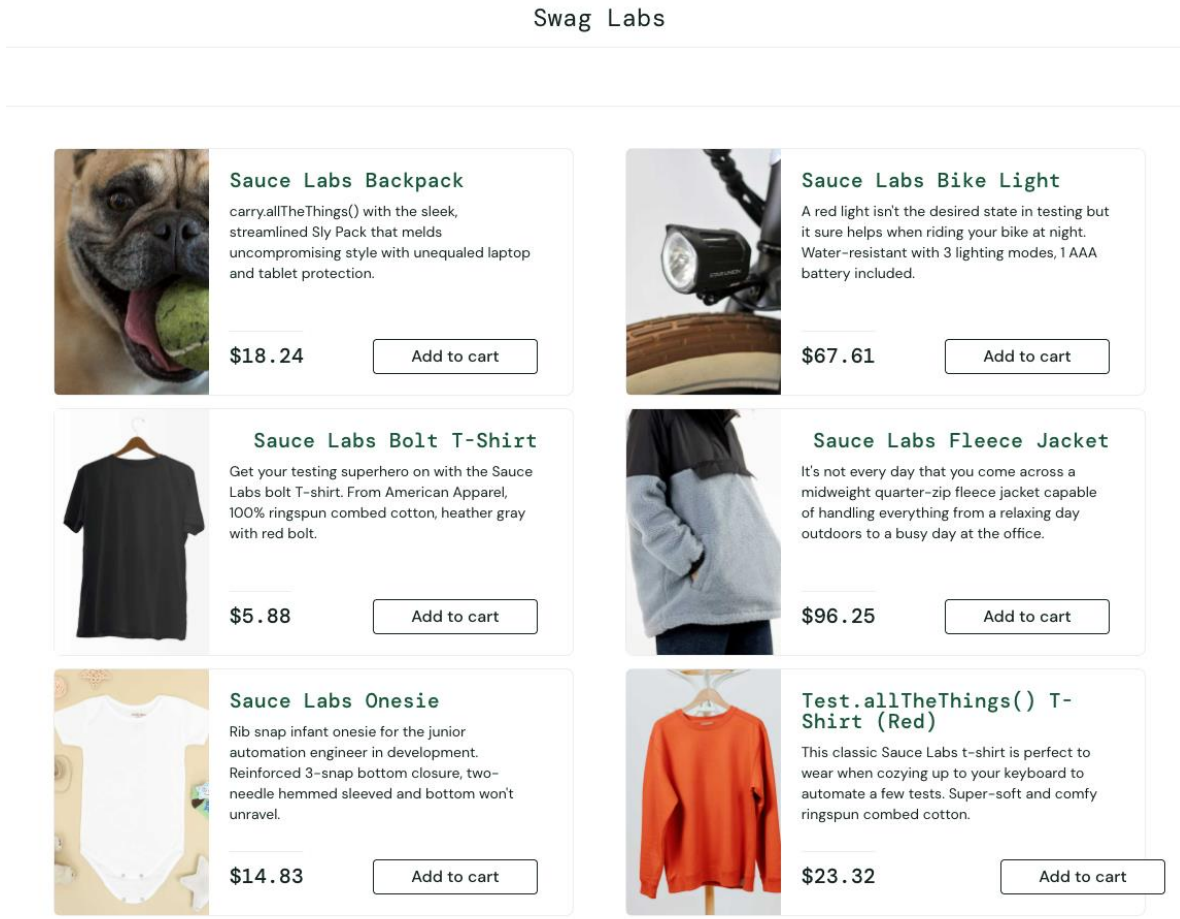
Example Vision LLM Input Image

This public website was developed by Sauce Labs for demo/testing purposes:

https://www.saucedemo.com/inventory.html

It includes deliberate bugs that we expect the AI to identify.

NB! Always check which data you are allowed to send to LLM providers e.g. enterprise / regulated / private data usage



Example Vision LLM Prompt

SYSTEM PROMPT

You are a senior QA engineer. Analyze this e-commerce screenshot and identify all UI, content, and functional issues.

Severity levels:

Critical – Blocks purchasing or damages trust

Major – Clear defect, workaround exists

Minor – Cosmetic, unlikely to affect users

Check for:

Broken/placeholder content, layout issues, wrong images, suspicious pricing, accessibility, staging data leaks.

Output valid JSON using this schema →

JSON SCHEMA

```
{
  "issues": [
    {
      "id": 1,
      "severity": "critical|major|minor",
      "category": "Content, UI/Layout...",
      "element": "Specific element",
      "description": "What is wrong",
      "expected": "Correct state",
      "impact": "User/biz impact"
    }
  ],
  "summary": {
    "critical": 0,
    "major": 0,
    "minor": 0,
    "total": 0,
    "overall_assessment": "Summary..."
  }
}
```

Example Vision LLM Output

JSON OUTPUT

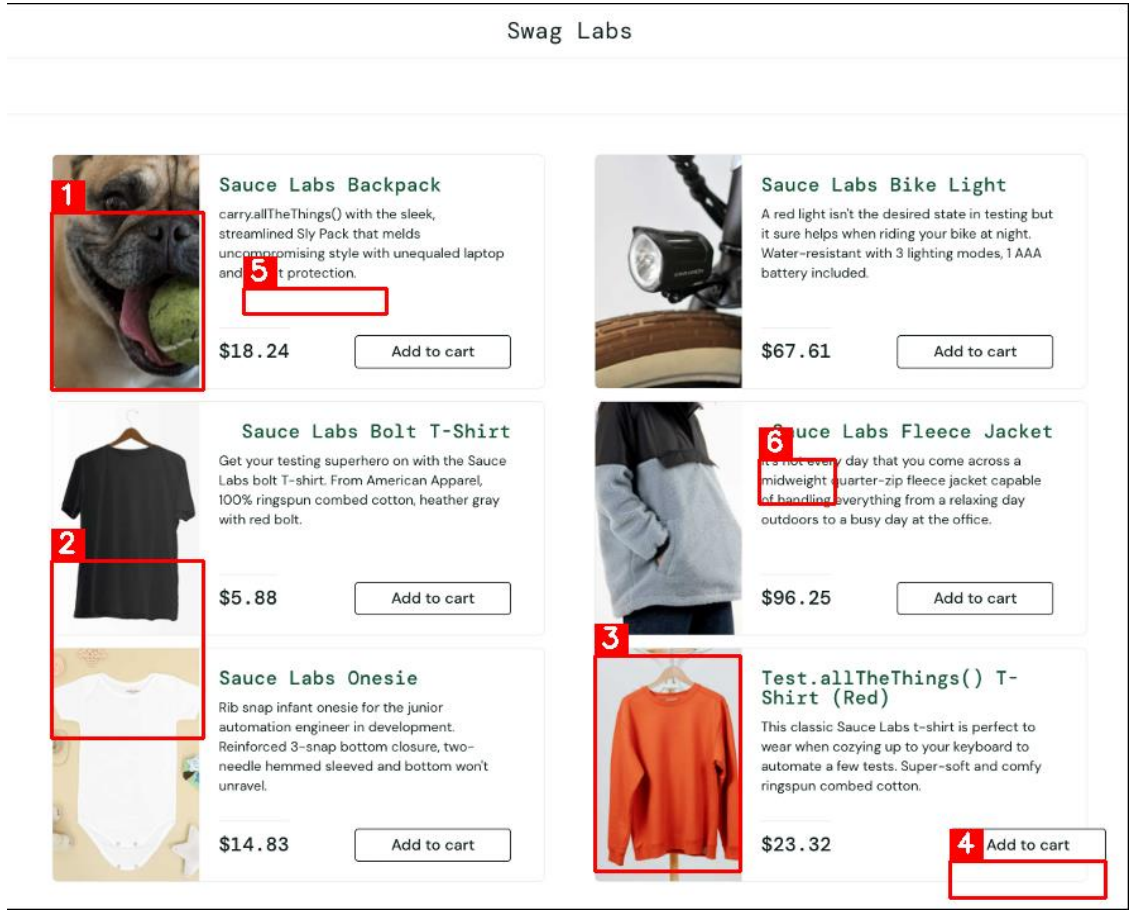

```
{
  "id": 1,
  "severity": "critical",
  "category": "Content/Copy",
  "element": "Page title and all product names",
  "description": "The entire storefront uses 'Sauce Labs'
    branding and product names (e.g., 'Sauce Labs
    Backpack', 'Sauce Labs Bolt T-Shirt'). Sauce
    Labs is a test automation platform; this is
    their demo app leaking test data into what
    should be production.",
  "expected": "Real brand name and real product
    names for an actual production store",
  "impact": "Destroys customer trust entirely;
    no real customer would purchase from a store
    obviously running test data. Indicates staging
    or QA environment exposed publicly."
},
```

Example Vision LLM Output

JSON OUTPUT


```
{
  "id": 2,
  "severity": "critical",
  "category": "Content/Copy",
  "element": "Product name: 'Test.allTheThings() T-Shirt (Red)'",
  "description": "Product name contains a literal test method
    call 'Test.allTheThings()' — unmistakable test/developer
    placeholder data visible to end users.",
  "expected": "A real, customer-facing product name
    (e.g., 'Classic Red T-Shirt')",
  "impact": "Confirms test data is in production;
    severely damages credibility and trust.
    Indicates staging or QA environment exposed."
}
```

Agentic Vision with Annotations Highlighting Detected Issues (Gemini 3 Flash)



ID	Severity	Issue Summary
1	Critical	Wrong image on Backpack (shows dog photo)
2	Major	Bolt T-Shirt missing red bolt graphic
3	Major	T-Shirt shows sweatshirt image
4	Minor	Inconsistent Add to Cart button width
5	Minor	Developer placeholder text in description
6	Major	Suspicious pricing suggests staging data

From Pixel Diffs to LLM Visual Intelligence Comparison

Traditional Pixel Diff		vs	AI-Powered Visual Reasoning
Detects	Pixel-level differences		Semantic UI issues
Reports	Changed vs unchanged pixels		Layout, hierarchy, meaning, accessibility
False Positives	High (any render diff over a threshold)		Medium (contextual reasoning)
Feedback	"Pixels differ at (x,y)"		"Button is hidden by overlay"
Exclusions	Manual exclusion areas per image		Ignore prompts
Speed	Milliseconds (local compute)		Seconds per image (depends on the model reasoning level)
Cost	Near zero (local CPU)		API token fees per invocation
Determinism	Deterministic		**Non-deterministic**

AI visual checks should complement, not replace, deterministic checks!

03

Advanced Analysis Techniques

Model selection | Agent review | Reducing noise and improving accuracy

Model Overview – Current Popular Vision-Capable LLMs*

OpenAI	Anthropic	Google	Open Weighted Models
GPT-5.5 GPT-5.4 Pro GPT-5.4 GPT-5.4 Mini GPT-5.4 Nano	Opus 4.7 Sonnet 4.6 Haiku 4.5	Gemini 3.5 Flash Gemini 3.1 Pro Gemini 3.1 Flash-Lite Gemini 3 Flash	Qwen 3.5 397B A17B Kimi K2.6 Qwen3.5 27B …

Thinking effort is configurable on most of these models

A "thinking effort" or "reasoning budget" setting (e.g. minimal / low / medium / high, or an explicit token cap).

It controls how many internal reasoning tokens the model is allowed to spend before answering.

Higher effort = better accuracy on hard tasks, at the cost of more latency and tokens; lower effort = faster, cheaper responses for simple queries.

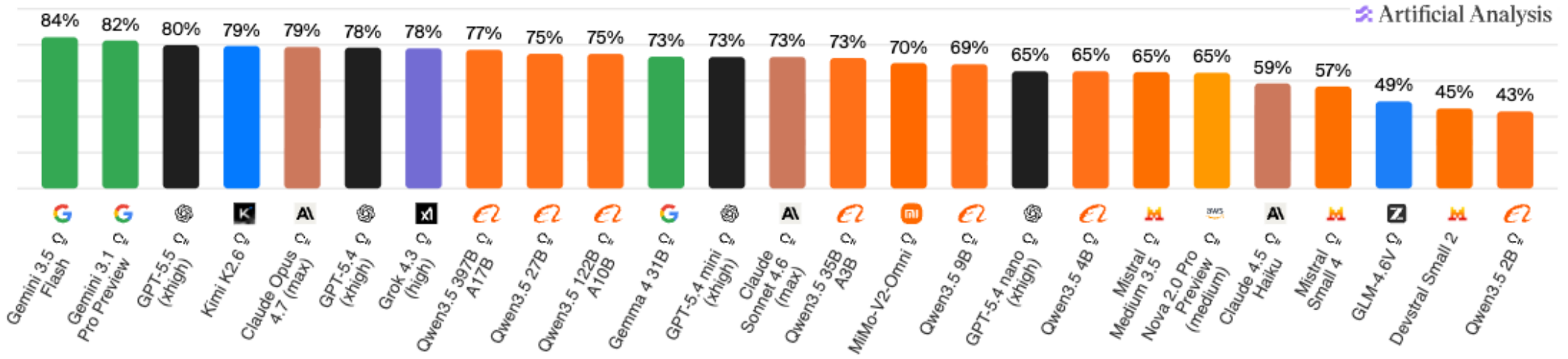
*as of: May 20, 2026

Visual Reasoning Intelligence (MMMU Pro evaluation)

25 of 190 models

0 of 417 model & provider com...

Visual reasoning intelligence: MMMU Pro evaluation



Reasoning models are indicated by a lightbulb icon.

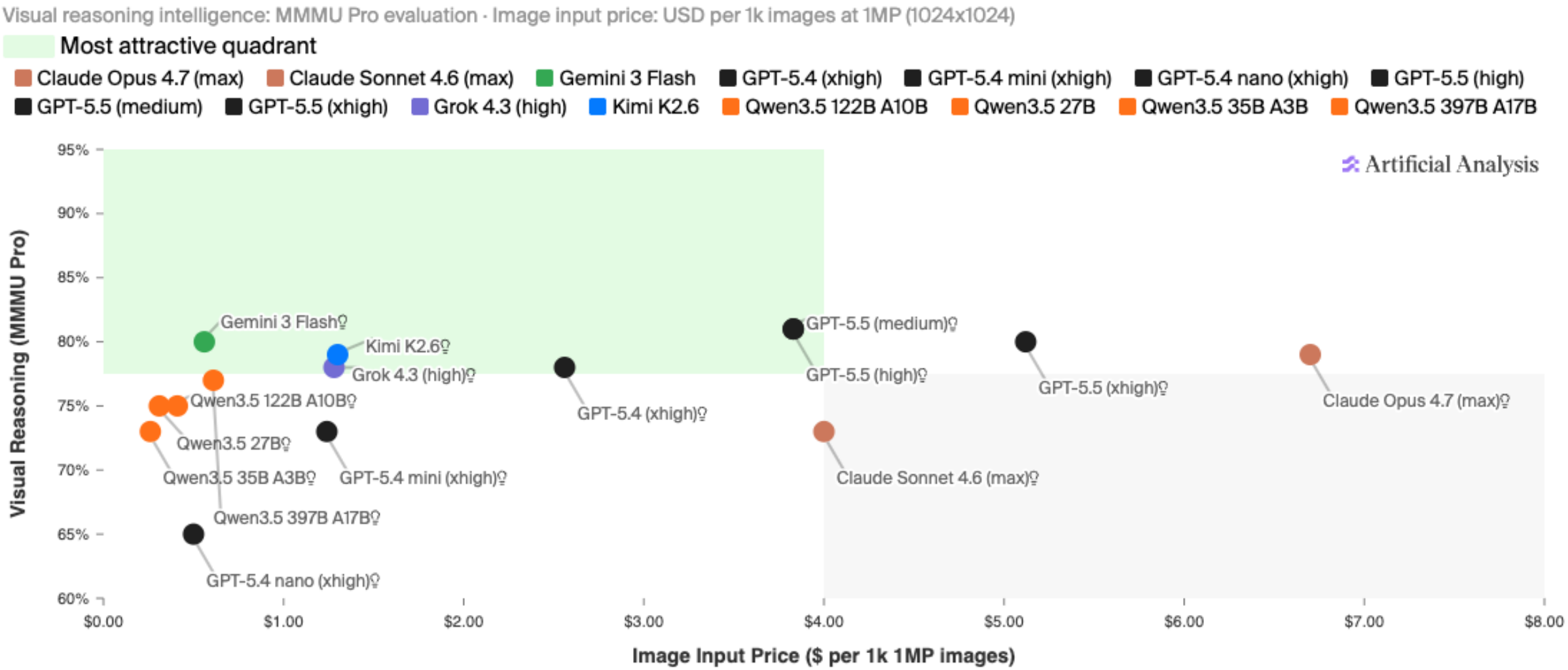
MMMU Pro

Multimodal reasoning quality evaluation based on 1.7k questions which require interpreting and reasoning over images.

*as of: May 20, 2026

https://artificialanalysis.ai/models/multimodal/vision

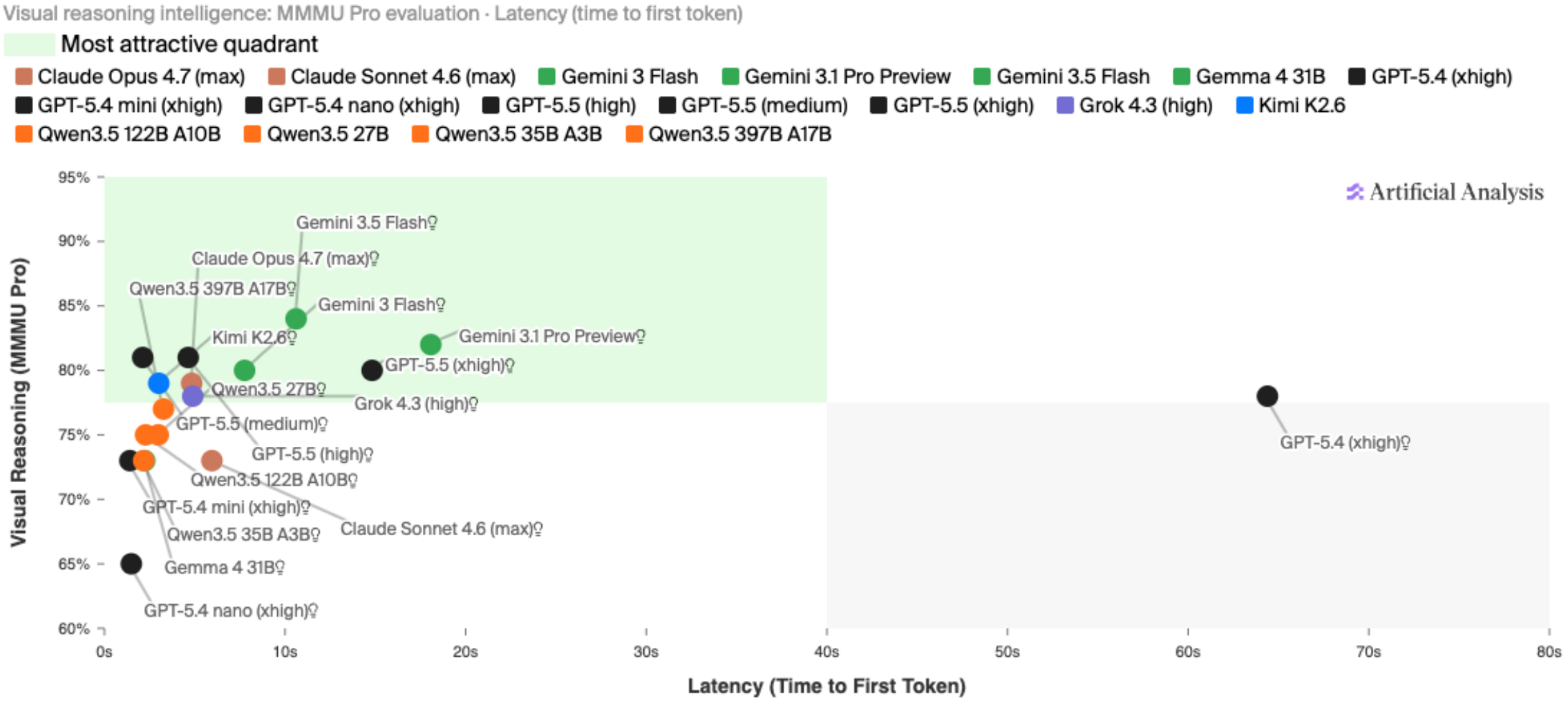
Vision Models Comparison – Visual Reasoning vs. Price*



*as of: May 20, 2026

https://artificialanalysis.ai/models/multimodal/vision

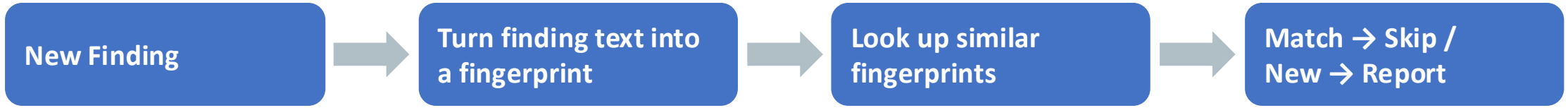
Vision Models Comparison – Visual Reasoning vs. Latency*



*as of: May 20, 2026

https://artificialanalysis.ai/models/multimodal/vision

Building a findings database – Dealing with duplicates



Why Deduplicate?

- LLMs re-flag the same issue with different wording across runs
- Reviewers waste time closing duplicates instead of fixing bugs
- Plain text matching misses rephrased findings
- We need to compare by meaning, not just words

How It Works

Summarize each finding
Convert the issue into a numeric "meaning fingerprint"

Save it to a database
Store fingerprints alongside each issue for fast lookup

Compare new findings
If the new fingerprint closely matches an existing one, treat it as a duplicate and link to the original.

Example: Same Bug, Different Words

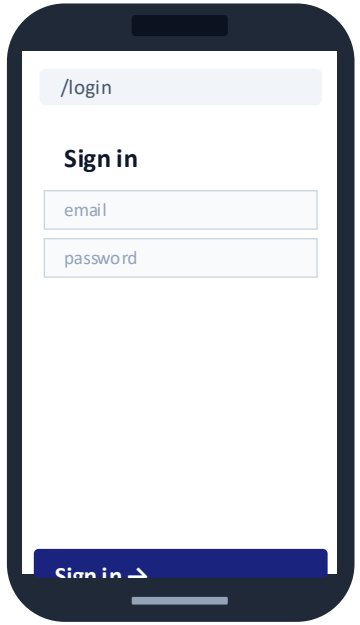
The AI inspects the app twice and describes the same problem in two ways. An embedding model recognizes they're the same and groups them together.

Finding A — Run 1 (build #4821)

"Login button on mobile is cut off at the bottom; users can't tap submit."

Finding B — Run 2 (build #4889)

"Sign-in CTA overflows viewport on iPhone — primary action not clickable."



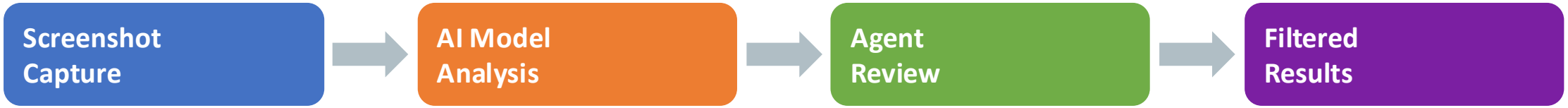
94% match

grouped_issue.json

```
group:        "issue_cluster_a3f9b1"
fingerprint:  "7c4e9b2a…d92ae0"
page:         "/login"
element:      "login_cta"
problem:      "viewport_overflow"
device:       "mobile"
matches:      [A#4821, B#4889]
```

The second finding is similar enough to be linked to the existing group — no duplicate ticket created.

Agent Review & False Positive Filtering



How Agent Review Works

- An AI Agent or multiple agents review the findings
- Filters out noise and low-confidence detections
- Validates issues against domain-specific rules

- Produces a curated, actionable report

When to Enable / Disable

Enable when:

- High false positive rate in initial results
- Domain-specific rules need enforcement

Disable when:

- Speed is critical (adds ~ an additional check)
- Cost becomes too expensive
- Initial model already has high enough accuracy

04

Integrating Visual Reasoning Checks into Workflows

Test automation & CI/CD integration

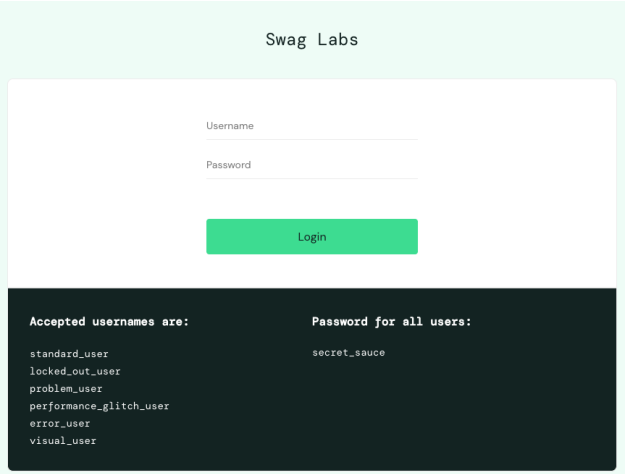
Integrating Visual Reasoning Checks

Selenium / Playwright / Cypress / Appium

Add as a library for existing test projects

test_visual_check.e2e.ts

```
const result = await ai.check(screenshot, [
  "The login button is visible",
  "No error messages are displayed",
  "A banner ad is present at the top of the page"
]);

assertVisualResult(result);
```



OUTPUT

```
2 of 3 checks passed.
Statements:
  PASS  "The login button is visible" — A green 'Login' button is clearly visible. (high)
  PASS  "No error messages are displayed" — No visible error alerts on the page. (high)
  FAIL  "A banner ad is present at the top of the page" — No banner ad visible. (high)
```

https://github.com/nullp2ike/visual-reasoning

Behind the Scenes: System Prompt

How ai.check transforms statements into structured visual QA results

INPUTS

```
screenshot  // Page image capture
```

```
statements[]
"Login button is visible"
"No error messages shown"
"Banner ad at top"
```

SYSTEM PROMPT

Role Assignment

"You are a visual QA assistant"

Evaluation Instructions

1. Evaluate each statement independently
2. Derive pass = AND of all results
3. Summarize reasoning
4. Report issues for failures only

JSON SCHEMA

```
{
  "pass": boolean,
  "reasoning": string,
  "issues": [...],
  "statements": [
    { statement, pass,
      reasoning, confidence }
  ]
}
```

Key Prompt Concepts

Role Definition

Sets the LLM as a visual QA assistant that evaluates screenshots precisely and objectively

Evaluation Instructions

Enforces statements-first evaluation, then derives pass as logical AND of all results

Confidence Scoring

Each statement gets high / medium / low confidence based on visual clarity and ambiguity

Structured Issues

Failed statements generate issues with priority, category, description, and suggestion

https://github.com/nullp2ike/visual-reasoning

Integrating Visual Reasoning Checks

diff.e2e.ts

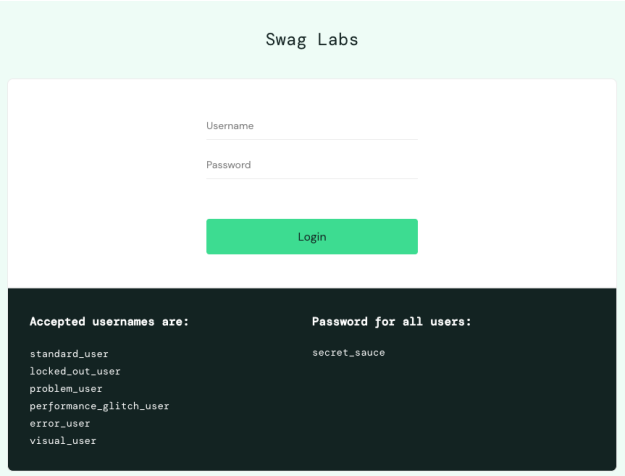
```
const result = await ai.compare(
  image('before.png'), //with login button
  image('after.png') //without the login button
);

expect(result.pass).toBe(true);
```

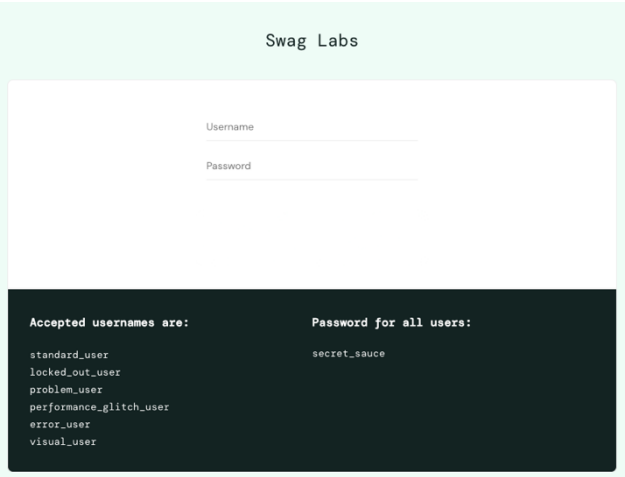
JSON OUTPUT

```
{
  "pass": false,
  "reasoning": "The primary call to action
    (the Login button) is completely
    missing in the current version.",
  "changes": [{
    "description": "The green 'Login'
      button has been removed.",
    "severity": "critical"
  }]
}
```

Visual Regression

Compare baseline vs current with AI





https://github.com/nullp2ike/visual-reasoning

Integrating Visual Reasoning Checks

post-deploy.e2e.ts

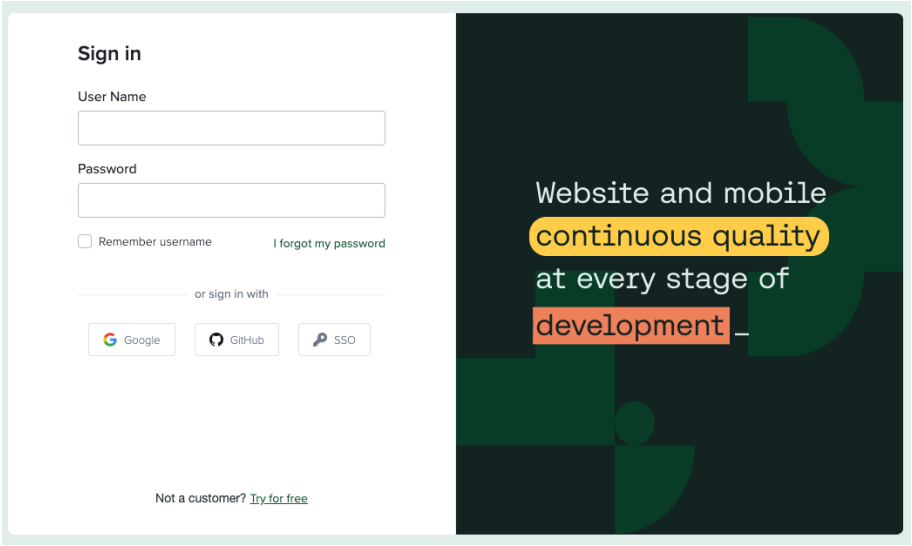
```
const result = await ai.check(screenshot, [
  'The page is displayed without any critical issues'
]);
expect(result.pass).toBe(true);
```

JSON OUTPUT


```
{
  "pass": false,
  "reasoning": "The login screen is missing a primary call-to-
action button (e.g., 'Sign in' or 'Login')...",
  "issues": [{
    "priority": "critical",
    "category": "missing-element",
    "description": "The primary 'Sign in' or 'Login' button is not
visible on the form.",
    "suggestion": "Add a prominent submit button..."
  }]
}
```


In Continuous Delivery Pipelines

Run final visual validation as a post-deploy step



https://github.com/nullp2ike/visual-reasoning

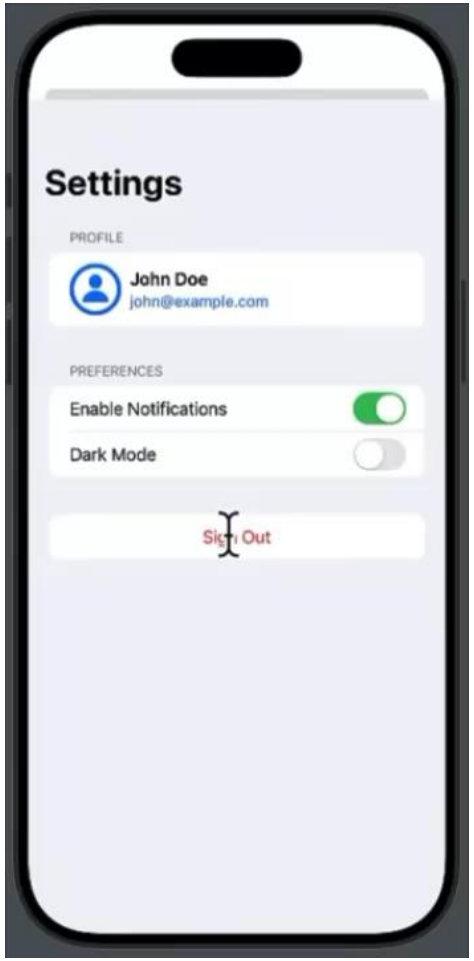
Video Assertion Support

post-deploy.e2e.ts

```
const result = await ai.check(video, [
  'You are signed out.'
]);
expect(result.pass).toBe(true);
```

JSON OUTPUT


```
{
  "pass": true,
  "reasoning": "The requested toast is visible during the sign out flow.",
  "issues": [],
  "statements": [{
    "statement": "The text \"You are signed out.\" is shown in a toast during
sign out.",
    "pass": true,
    "reasoning": "A blue toast displays \"You are signed out.\" at 5.50s,
visible through 6.50s.",
    "confidence": "high",
    "timestampSeconds": 5.5
  }]
}
```

https://github.com/nullp2ike/visual-reasoning

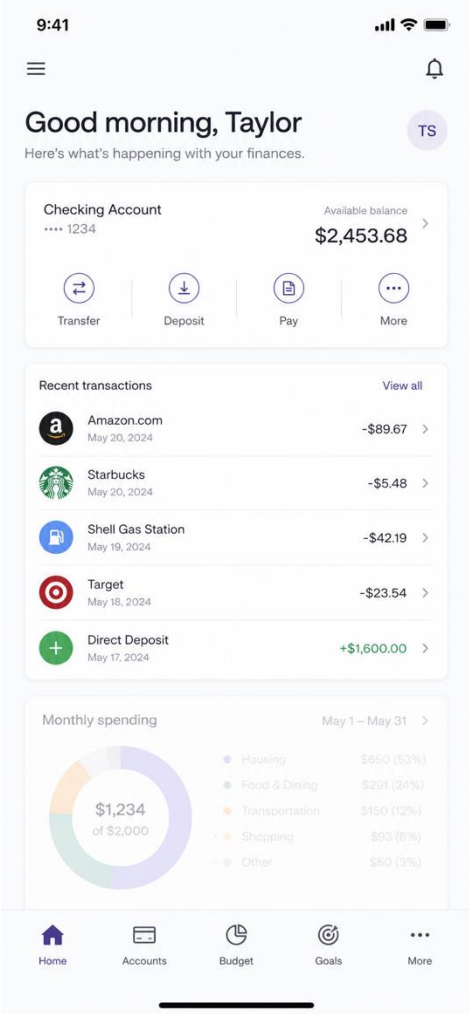
Accessibility Support

post-deploy.e2e.ts

```
  const result = await ai.accessibility(screenshot, {
      checks: [Accessibility.CONTRAST, Accessibility.COLOR_ALONE],
    }
  );
  expect(result.pass).toBe(true);
```

JSON OUTPUT


```
{
  "pass": false,
  "reasoning": "Most primary content is readable, but some small
secondary text and labels appear very light and hard to read.",
  "issues": [{
    "priority": "major",
    "description": "Secondary text and chart labels are low
contrast, especially in the monthly spending card.",
    "suggestion": "Darken light gray and pale purple text against
the white background."
  }]
}
```

https://github.com/nullp2ike/visual-reasoning

Privacy & Assertion Library Project Status

Privacy

No data collection

The library does not transmit any user data except for what the core functionality does by sending API requests to LLM providers.

Always verify this yourself before adopting it.

Project status

- Proven to be useful in a real project, though **not an official stable release yet**
- Fork it, experiment, and see if it fits your use case.
- Supports currently **OpenAI**, **Anthropic**, and **Google** models, but more vendors can be easily added

More on GitHub: **github.com/nullp2ike/visual-reasoning**

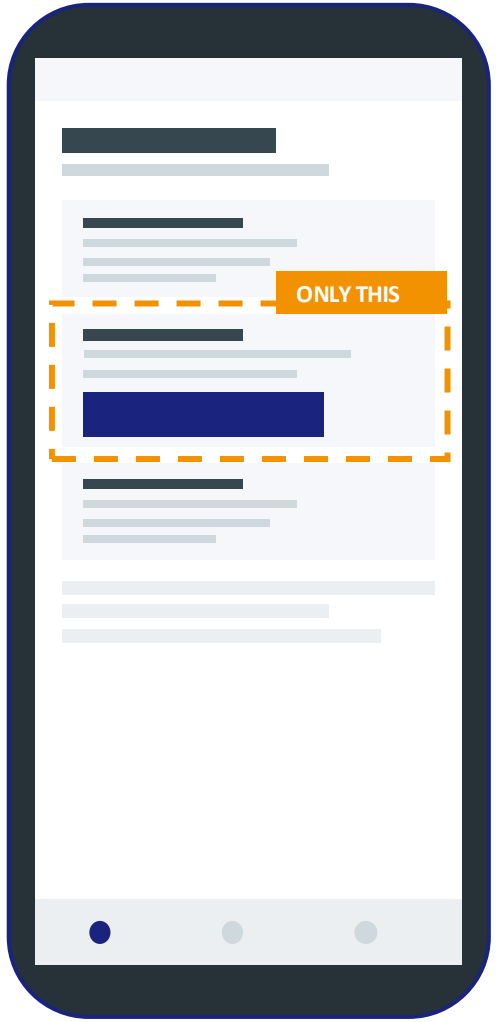
Select Only The Element You Are Interested In

Send the model only the element you care about — sharper answers, lower cost, faster runs.

- **Accuracy** — less visual noise, fewer distractors.
- **Cost** — far fewer image tokens per call.
- **Speed** — less data to upload and process.

crop-element.ts

```
// WebdriverIO
const cta = await $('~signupCta');
await cta.saveScreenshot('./cta.png');
// Playwright
const cta = page.getByRole('button', { name: 'Sign up' });
await cta.screenshot({ path: 'cta.png' });
// Hand the element image to the model
await ai.check('cta.png', ['Button says Get started']);
```



Monitoring Production in the Wild

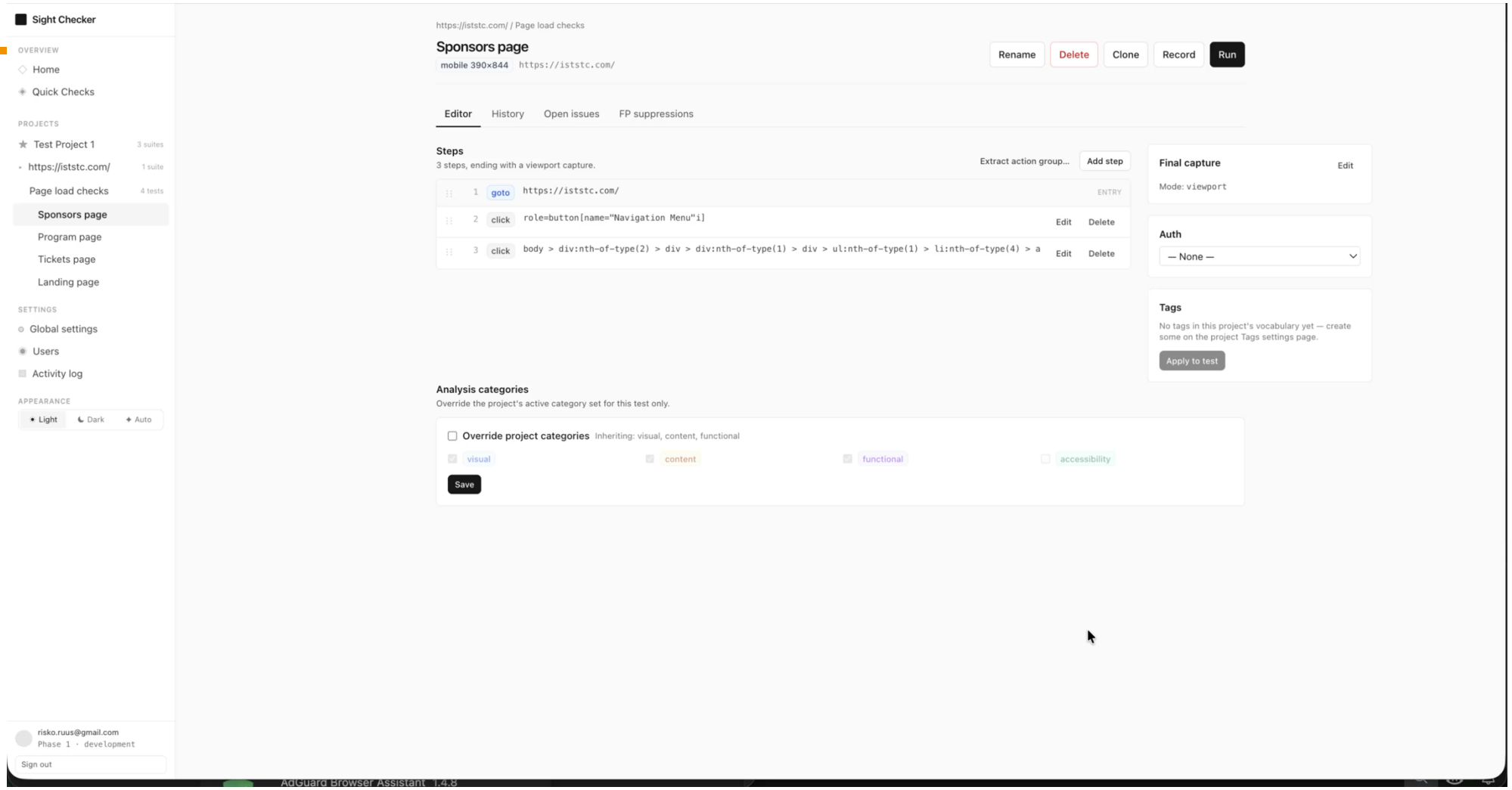
Run visual LLM assertions against live sites and apps — catch regressions users would actually see.

- **Synthetic checks** — scheduled crawls hit key flows, screenshot the elements that matter.
- **LLM assertions, not pixel diffs** — ask in plain English: "Is the CTA visible and legible?"
- **Resilient to noise** — shrugs off A/B variants, dynamic content, minor restyles.
- **Catches what users notice** — overlapping text, broken images, cut-off buttons, contrast fails.
- **Page on real failures** — route findings to Slack/PagerDuty with the offending screenshot attached.

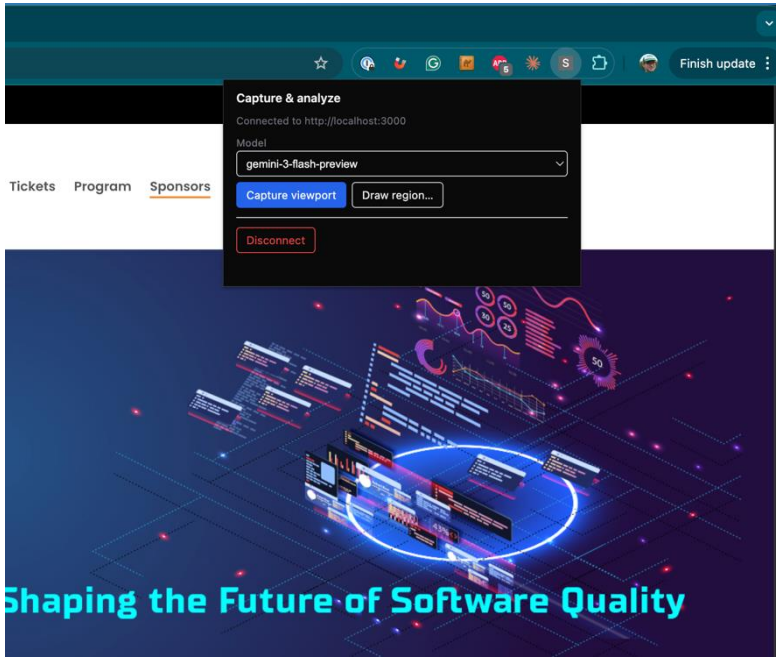
monitor-prod.ts

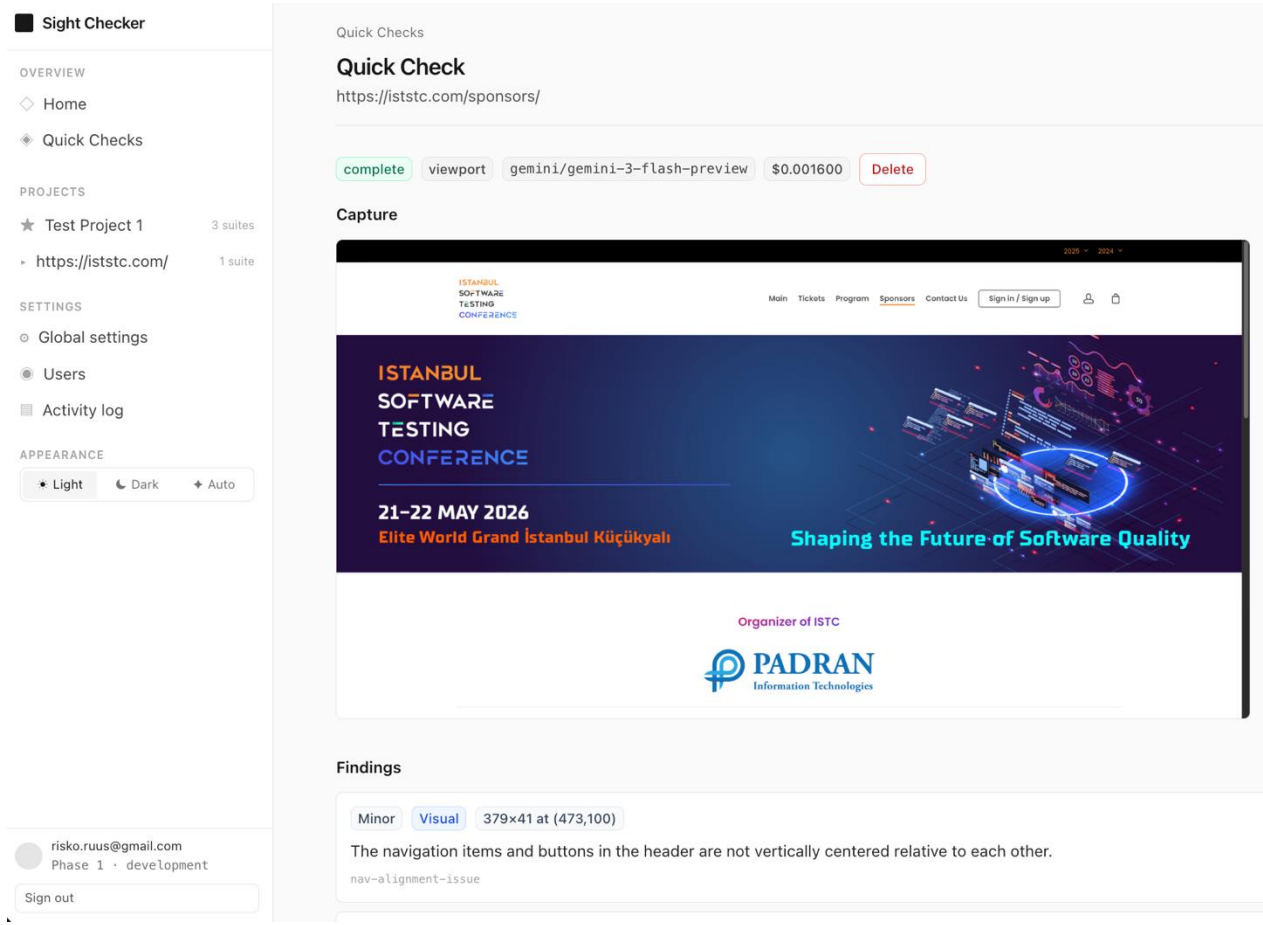
```
// Run every 5 min in prod
await page.goto('https://acme.com');
const hero = page.locator('[data-
test=hero]');
await
hero.screenshot({path:'hero.png'});

// Visual LLM assertions
const result = await
ai.check('hero.png', [
  'Headline is readable',
  'CTA button visible',
  'No overlap or cutoff',
]);

if (!result.ok) alert(result);
```

Monitoring Production in the Wild: No-Code Tooling



Empowering Manual Testing – Browser Extension





Empowering Manual Testing – Mobile Companion

Visual checks on real devices — a shortcut, your voice, and an LLM verdict on your phone.

- **Device shortcut** — iOS Shortcut or Android quick-tile fires a screenshot of whatever the tester is looking at.
- **A shortcut for the shortcut** — Bind the shortcut to the phone's action button or to back double/triple tap accessibility features
- **Voice the assertion** — tester speaks the check: "E.g. check the content copy text, critical issues only"
- **Sent for analysis** — screenshot + transcribed prompt are POSTed to the LLM service; verdict returns in seconds.
- **Companion app log** — pass/fail, reasoning, and the captured image stream into a session log, ready to attach to the bug and send to an issue tracker

Key Takeaways

1. LLMs bring semantic understanding to visual testing — they see meaning, not just pixels

Key Takeaways

1. LLMs bring semantic understanding to visual testing — they capture meaning, not just pixels

2. Multiple AI providers offer different strengths — use tiered strategies for balancing speed, cost and accuracy

Key Takeaways

1. LLMs bring semantic understanding to visual testing — they capture meaning, not just pixels
2. Multiple AI providers offer different strengths — use tiered strategies for balancing speed, cost and accuracy
3. LLMs are **non-deterministic**, keep using your existing deterministic tests together with visual AI checks

Key Takeaways

1. LLMs bring semantic understanding to visual testing — they capture meaning, not just pixels
2. Multiple AI providers offer different strengths — use tiered strategies for balancing speed, cost and accuracy
3. LLMs are **non-deterministic**, keep using your existing deterministic tests together with visual AI checks
4. Integration is straightforward — add AI checks as assertions in existing test suites

Key Takeaways

1. LLMs bring semantic understanding to visual testing — they capture meaning, not just pixels
2. Multiple AI providers offer different strengths — use tiered strategies for balancing speed, cost and accuracy
3. LLMs are **non-deterministic**, keep using your existing deterministic tests together with visual AI checks
4. Integration is straightforward — add AI checks as assertions in existing test suites
5. Start small, measure impact, and scale gradually across your test infrastructure

ISTANBUL SOFTWARE TESTING CONFERENCE

Thank you!

Questions? Get in touch:

risko.ruus@gmail.com | linkedin.com/in/risko-ruus/