# ListingLens: a quality gate for product photos with verifiable autonomy

Technical report for the OpenCV AI Competition 2026 (powered by AWS). Paths: overall + **Agentic Vision** + **Best Use of COOL**.

**Summary.** ListingLens is an OpenCV 5 agent that sits between a seller's camera and a store listing. It is allowed to act on its own only where it can check its own work: after every action it measures the photo again and keeps the change only if the measurement proves it helped. Everything else goes to the seller (reshoot, with a concrete reason) or to a person (approve / reject). The design choice is bounded autonomy over clever generation: no generative editing, every decision traceable to an OpenCV number. Key results: 89.6 % correct decisions on 1,768 held-out photos vs 85.2 % for the same pipeline without verification (whose "fixes" fail the checks 25 % of the time); 27 of 33 problem photos stopped on 106 hand-labelled real catalog photos; COOL on Graviton4 speeds up the operations it targets by up to 2.9x. For a shop with 500 SKUs and 4 photos each, that is 2,000 photos screened in about 35 minutes of Lambda time at $0; a person reviews only the flagged ones, which on our labelled sample of real catalog photos was about half of them. Cutting that review load is the main open problem.

## 1. Problem and users

Every online-store listing needs a main photo that is sharp, well exposed, neutral in colour, with the whole product centred on a plain light background. Marketplaces enforce this (white background, product filling most of the frame, at least 1000 px for zoom), and listings that miss it get rejected or convert worse. Small sellers shoot on phones in kitchens and warehouses and learn about the problem only after rejection.

Two common answers both fail them. Manual retouching is slow. Generative "enhancement" repaints the product, and a product photo that no longer matches the item is a consumer-protection problem.

**Users:** small marketplace sellers and the content teams of small online shops (one person approving dozens of photos a day). **Goal:** in about a second per photo, tell them for each photo either "fine", "fixed, here is the result and what changed", "reshoot, and here is exactly what to do differently", or "a person should decide, here is why". Every answer comes with its evidence.

## 2. What the system does

```
photo -> PERCEIVE (OpenCV 5) -> issues -> DECIDE (planner) -> ACT (OpenCV tool) -> VERIFY (perceive
again)
                                    ^                                                  |
                                    +------ keep the result, or revert it and mark -----+
                                            that tool as failed for that issue
final: accept | auto_fixed | reshoot (+ concrete tip) | human (+ reason)  + full JSON trace
```

The OpenCV measurements are not just shown to the user: they decide the next tool call, and the measurement taken **after** each action decides whether the action is kept, whether another remedy is tried, or whether the photo is escalated. This is the Agentic Vision loop.

Architecture diagram: architecture.svg.

## 3. OpenCV 5 implementation

All vision code is OpenCV 5.0 (`opencv-python-headless==5.0.0.93`) plus NumPy. No neural networks are needed for the decisions, which keeps every number explainable and the Lambda package small.

### 3.1 Perception (`listinglens/perceive.py`)

Images are analysed at 1024 px on the long side.

| Measurement | OpenCV 5 method |
|---|---|
| Product mask | LAB conversion, median colour of a 4 % border strip as background model, Euclidean LAB distance, Gaussian smoothing, Otsu threshold on the distance map with a JPEG-noise floor, elliptical close/open, external contours above 0.2 % of the frame |
| Mask confidence | separation of the distance distribution inside vs outside the mask, excluding the morphological-gradient edge band |
| Framing | bounding rect of the kept contours: occupancy (max of width/height share), centre offset, and **edge contact** (longest run of product pixels along any frame edge: the cut-off signal) |
| Background | border strip minus the dilated product: mean L, L standard deviation, Canny edge share (clutter), chroma (cast) |
| Exposure | product L mean, share of clipped highlights, 2nd-percentile L (black point) |
| Focus | Crete-Roffet no-reference blur index (re-blur comparison with 1x9 / 9x1 box filters) and variance of the Laplacian, both on the product's bounding box only, so a large white background does not dilute them |
| Noise | Immerkaer fast noise estimate (3x3 kernel, `filter2D`) |
| Duplicates | 64-bit perceptual hash from the 8x8 low-frequency block of a 32x32 `dct` |

### 3.2 Issues and policy (`listinglens/policy.py`)

Thresholds live in one dictionary, so every decision in the trace can be traced to a number. Issues carry a severity: `fixable`, `reshoot` (detail is physically missing: blur, cut-off, clipped highlights, low resolution) or `human` (ambiguous evidence: unclear product, possible cut-off, duplicate). On a cluttered background the mask-based readings are unreliable, so only the clutter is reported first; after the background is cleaned the agent measures again and the real issues appear.

### 3.3 Tools (`listinglens/tools.py`)

Every tool is deterministic and global or geometric; none paints new content.

| Tool | OpenCV 5 |
|---|---|
| `recrop` | product bbox to a square canvas at 85 % occupancy; up to 2x Lanczos upscale, never more |
| `white_balance` | LAB a/b shift that neutralises the measured background cast |
| `exposure` | levels on L (background to white, or gamma toward mid-tone for dark scenes) + CLAHE on the product only |
| `whiten_background` | GrabCut seeded from our mask (sure-FG core by erosion, sure-BG border), 640 px, 3 iterations, feathered composite on white. Returns seed/GrabCut IoU, and a **scene flag** when the foreground runs along the frame |

| Tool | OpenCV 5 |
|---|---|
| `denoise` | `fastNlMeansDenoisingColored`, strength derived from the measured noise sigma |
| `sharpen` | unsharp mask, allowed only for mild softness |

### 3.4 The agent loop and its guards (`listinglens/agent.py`)

- Step budget of 6 actions, then `human`.
- After every action: re-perceive, compute the new issue set. The action is **reverted** if it introduced a new issue, did not fix the issue it targeted, failed its tool-specific check (sharpening must lower the blur index by a margin, so it cannot pass by gaming the metric), GrabCut disagreed with the seed mask (IoU < 0.80), the foreground ran past the frame (scene photo), or the product's chroma moved by more than the measured background cast + 8 (truthfulness guard).
- A reverted tool is excluded for that issue; the planner must pick another remedy or escalate.

### 3.5 Planners (`listinglens/planners.py`)

- `RulePlanner` (deployed): hard stops first (reshoot / human), then the least invasive remedy.
- `BedrockPlanner`: Amazon Nova via the Bedrock Converse API with a forced tool call. The model sees the metrics, thresholds, issues, reverted tools and the list of **legal** actions, and must pick one. Any illegal choice or API error falls back to `RulePlanner` and the fallback is written to the trace (covered by a unit test). It is off in the public deployment because Bedrock is not in the free tier and the competition budget for this entry was $0 for serving.

### 3.6 Example trace (dev set, grey background)

Initial issues: `soft_focus` (blur index 0.50, Laplacian var 35.5), `background_not_white` (L 163).

| step | action | outcome |
|---|---|---|
| 1 | exposure | **reverted**: introduced blown_highlights, unclear_product |
| 2 | whiten_background | kept: fixed background_not_white, product chroma shift 0.1 |
| 3 | accept | all checks pass |

The first remedy's measurement made the agent change course; the second remedy fixed the photo. The full JSON is in example_trace.json, images `example_before.jpg` / `example_after.jpg`. Note the `soft_focus` reading also cleared: the low-contrast grey background had depressed the focus metrics. This is a measurement interaction, not real sharpening, and it is listed under limitations.

## 4. AWS deployment

| Component | Service | Notes |
|---|---|---|
| Inference + web page | **AWS Lambda, arm64 (Graviton)**, Python 3.12, 2048 MB | layers: OpenCV 5 (36 MB zip) and NumPy (12 MB zip); [verified: `/health` returns `opencv 5.0.0`, `aarch64`] |
| HTTPS endpoint | Lambda Function URL (auth NONE) | the page and `POST /api/check` on one origin |
| Observability | CloudWatch Logs, 14-day retention | one JSON line per request: decision, issue codes, steps, latency; no pixels, no personal data |

| Component | Service | Notes |
|---|---|---|
| Abuse / cost guard | CloudWatch alarm (>1000 invocations/hour) -> SNS -> `listinglens-killswitch` Lambda | sets reserved concurrency to 0 and emails the owner; re-enables itself after a quiet hour, stays off after 3 trips in one UTC day (counter in SSM Parameter Store). Tested live on 16 Oct: ALARM -> HTTP 429, OK -> HTTP 200 |
| Budget | AWS Budgets | $1/month alert on actual and forecast |
| Build & deploy | CloudShell + `deploy/build_lambda.sh`, `deploy/deploy_lambda.sh` | idempotent; no long-lived access keys were created |
| COOL benchmark | EC2 c8g.large (Graviton4) from the COOL 3.1 AMI | self-terminating user-data, results on the serial console |

Serving cost: inside the AWS always-free allowances (Lambda 1M requests and 400k GB-s per month). One photo uses 0.3-15 GB-s (median about 0.6), so the free allowance covers tens of thousands of photos per month. The COOL benchmark cost about $0.25 of EC2 time; the COOL software itself was in its 7-day free trial.

Local and cloud parity: the same test photos produce identical metrics (to 4 decimals) and decisions on the Mac (arm64) and on Lambda (aarch64).

## 5. COOL on Graviton4

**Setup.** One c8g.large (2 vCPU AWS Graviton4, Neoverse-V2, 2.8 GHz) launched from the AWS Marketplace AMI "Cloud Optimized OpenCV For AWS Graviton4", version 3.1. Two Python 3.12 environments on the **same instance**: COOL (`/opt/cool/venvs/python_3.12`, reports OpenCV `5.1.0-dev`, KleidiCV enabled) and a baseline venv with the stock `opencv-python-headless==5.0.0.93` wheel (which also ships KleidiCV for aarch64). The same `bench/bench_cool.py` runs in each, alternating, three rounds. Workload: 20 micro operations on a 1600x1600 photo (the calls ListingLens actually makes), then the full agent on 20 catalog photos x 13 defect variants = 260 images.

**Proof that COOL executes the workload:** each result line records `cv2.__file__` (`/opt/cool/python_3.12/site-packages/cv2/python-3.12/cv2.cpython-312-aarch64-linux-gnu.so` vs `/opt/stock/lib/python3.12/site-packages/cv2/__init__.py`) and the version string.

**Results (final run, median of 3 rounds each; speedup = stock / COOL; full table in [bench/RESULTS.md](bench/RESULTS.md)):**

| operation | COOL ms | stock ms | speedup |
|---|---|---|---|
| blur (box 1x9) | 2.744 | 7.862 | **2.87x** |
| threshold (Otsu) | 0.856 | 1.658 | **1.94x** |
| CLAHE apply | 4.755 | 6.663 | **1.40x** |
| findContours | 0.158 | 0.174 | 1.10x |
| grabCut 1024 px, 2 it | 720.8 | 761.1 | 1.06x |
| fastNlMeansDenoisingColored 1024 px | 764.3 | 778.3 | 1.02x |
| imencode JPEG | 5.975 | 6.109 | 1.02x |
| resize, cvtColor, Laplacian, filter2D, morphology, warpAffine | | | 0.97-1.02x |
| GaussianBlur sigma 1.6 | 10.547 | 9.942 | 0.94x |
| Canny | 2.296 | 2.075 | 0.90x |

| operation | COOL ms | stock ms | speedup |
|---|---|---|---|
| adaptiveThreshold (Gaussian) | 11.586 | 8.973 | **0.77x** |
| **full agent, mean per image (260 images)** | **344.5** | **356.1** | **1.034x** |
| full agent, p95 | 1226.2 | 1328.9 | 1.084x |
| full agent, throughput (img/s) | 2.903 | 2.808 | 1.034x |

Decision hash `f6c1d93bb5b62db5` is identical in all six runs: COOL changes speed, not answers. An earlier run on a previous commit (GrabCut at 1024 px) showed the same pattern (1.028x end to end).

**Where the time goes (second run with a per-stage profile, same instance type, 3 + 3 rounds; details in [bench/RESULTS.md](bench/RESULTS.md)).** Perception takes about 40 % of the agent's time and does not speed up at all with COOL (30.1 s vs 30.0 s per 260 photos): a profiler shows about 80 % of it is NumPy and Python glue (LAB distance, percentiles, masks), not OpenCV calls. Denoising gains 1.10x, GrabCut 1.01x. The whole agent: 1.025x in this run, 1.034x in the first; decisions identical in all twelve runs.

**Reading the numbers honestly.** COOL gives 1.4-2.9x on the box filter, Otsu threshold and CLAHE that the perception stage uses, but ListingLens spends most of its time in GrabCut and non-local-means denoising, where COOL is 2-6 % faster, so the end-to-end gain is 3.4 % on the mean and 8.4 % on the p95 tail (the GrabCut-heavy photos). With the COOL software fee ($0.01/h on top of $0.0798/h for c8g.large) the cost per 1000 photos on one c8g.large is $0.0086 with COOL vs $0.0079 with the stock wheel at these speeds. COOL would pay off for workloads dominated by the operations it accelerates (thresholding, box filtering, CLAHE), which is what a perception-only pre-filter in front of the agent looks like, and for a perception stage rewritten so that its arithmetic runs inside OpenCV instead of NumPy. One regression (adaptiveThreshold 0.77x) is worth reporting upstream.

## 6. Evaluation

### 6.1 Data

- 358 main catalog photos of 184 product types from **Amazon Berkeley Objects** (CC BY 4.0, AWS Open Data).
- 252 pass a strict "already good" filter; each is normalised (centred on white, 1200 px) and degraded in 13 controlled ways, so the correct answer is known: clean, strong blur, motion blur, soft focus, dark, colour cast, grey background, procedural cluttered background, small and off-centre, cut off, noisy, overexposed, low resolution. 3276 labelled images.
- Split by image id: **dev** 1508 images (all thresholds were tuned here), **test** 1768 images (held out).
- 106 catalog photos that failed the filter, plus 40 real lifestyle / in-use photos (secondary listing images), are used for failure analysis.

Some variants accept more than one answer, decided before looking at test: soft focus may be fixed or sent to reshoot; a cluttered background may be cleaned or sent to a human; a cut-off product may go to reshoot or a human. Two of these sets were widened **after** inspecting dev results (small/off-centre may also be "reshoot closer", because a 2x upscale genuinely lacks detail; cut-off may also be "human"). We state this because it raises the dev score.

### 6.2 Results (rule planner)

| split | images | correct | accuracy |
|---|---|---|---|
| dev | 1508 | 1363 | 90.4 % |
| **test (held out, run once)** | 1768 | 1584 | **89.6 %** |

Per variant on dev: strong blur, low resolution, cluttered background 100 %; colour cast and dark 98 %; clean 96 %; motion blur 95 %; small/off-centre 92 %; cut-off 90 %; noisy 89 %; grey background 88 %; soft focus 84 %; **overexposed 45 %**.

Per variant on test: strong blur, low resolution, dark, cluttered background 100 %; colour cast 97 %; clean 97 %; small/off-centre 97 %; grey background 93 %; cut-off 93 %; noisy 88 %; motion blur 85 %; soft focus 81 %; **overexposed 33 %**. The test split was first run after all dev tuning; nothing was tuned on it. It was re-run once after seeding GrabCut's random initialisation for repeatable decisions (1 of 1768 decisions changed; two consecutive runs are now identical).

Agent behaviour on dev: in 337 of 1508 photos at least one action was reverted because the follow-up measurement showed it did not help; the next step then changed (most often `sharpen -> reshoot`, `whiten_background -> human`, `exposure -> whiten_background -> accept`).

### 6.3 Ablation: what the verify loop buys (held-out test, 1768 photos)

| pipeline | correct (lenient) | correct (strict, pre-registered sets) | "fixed" photos that still fail the checks |
|---|---|---|---|
| **ListingLens agent** (perceive, decide, act, verify, revert) | **89.6 %** | **87.5 %** | 0 of 691 (by construction) |
| open loop (same rules and tools, apply first remedy once, no re-measure) | 85.2 % | 84.5 % | **255 of 1032 (25 %)** |
| rules only (thresholds, no actions) | 85.2 % | 84.5 % | n/a |

On the 371 test photos where the agent reverted at least one of its own actions, it reached the right decision 94.6 % of the time; the open-loop pipeline got 73.9 % of the same photos right. The strict column uses the single-answer sets written down before any result was seen (section 6.1). Script: `eval/ablation.py`, output `eval/results/ablation_test.json`.

### 6.4 Hand-labelled real catalog photos (106)

The 106 real Amazon catalog photos that failed our strict "already clean" filter were labelled by us against main-image rules (white background, whole product in frame, no text or badges on the image, not a room scene): 73 acceptable, 33 with a problem. Labels and decisions per photo: `eval/results/real106_labels.csv`.

| | agent | open loop |
|---|---|---|
| problems stopped (not accepted / not auto-passed) | **27 / 33 (82 %)** | 11 / 33 (33 %) |
| "accept" that was right | 18 / 23 (78 %) | 18 / 23 (78 %) |
| good photos passed automatically | 30 / 73 (41 %) | 44 / 73 (60 %) |
| good photos sent to reshoot (false alarm) | 12 / 73 (16 %) | 12 / 73 (16 %) |
| good photos sent to a person | 31 / 73 (42 %) | 17 / 73 (23 %) |

Read as a **quality gate**, ListingLens stops most bad photos before they are published, and pays for it with review load: four in ten good catalog photos still go to a person, mostly tight crops that touch the frame and multi-item composites that read as a cluttered background. The five problems it let through are text badges and overlays (it has no text detector) and two products cut by the frame edge. These are professional catalog photos, not the phone photos small sellers take; testing on seller phone photos is the next step.

### 6.5 Real photos and failures

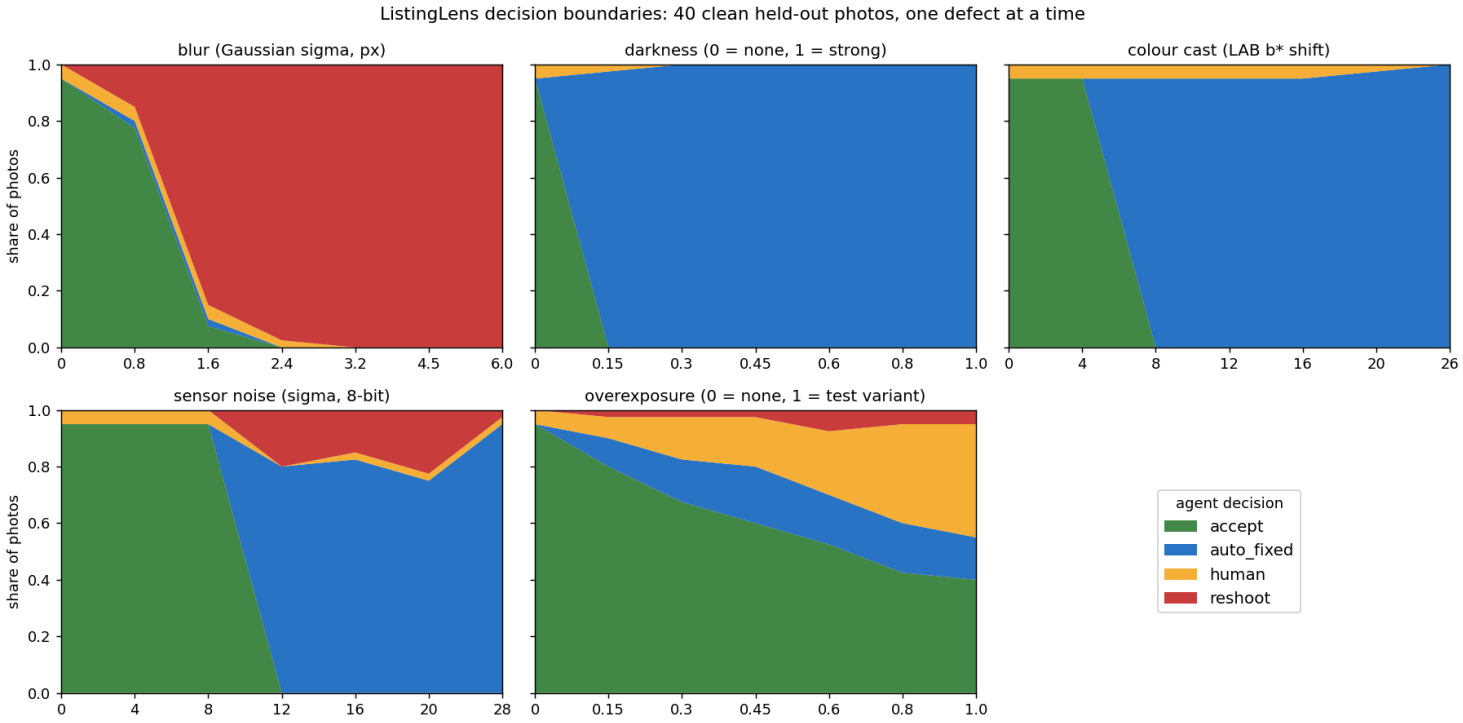
- **Lifestyle / in-use photos (40, real):** before the scene guard, 10 of 40 were "auto-fixed" by cutting people, rooms and tyres out of their context. After adding the scene check (foreground running along the frame), 1 of 40 is auto-fixed (an instruction diagram), 37 go to a person with an explanation, 2 to reshoot. These 40 were used to design the guard, so they are not a held-out score.
- **Overexposed products:** a white product on a white background looks like a blown-out one. The agent catches 46 % of synthetic overexposure, and on the first held-out test product (a white item) it flags "blown highlights" on every variant. This is the main open weakness.
- **Mild blur on smooth products** is sometimes accepted (16 of 116 soft-focus images on dev).
- **Noise:** 13 of 116 noisy photos end in "reshoot" because denoising raised the blur index; the result is safe but conservative.
- **Real professional catalog photos (106 that failed the strict filter, no labels):** the first version sent 38 to reshoot, mostly false alarms: fabric and printed packaging read as sensor noise, and white products on white read as blown highlights. Measuring noise on flat regions only and turning "blown highlights" into a human decision brought this to 18 reshoot / 52 human / 23 accept / 13 auto-fixed (dev accuracy moved from 91.1 % to 90.4 %). The remaining reshoots are mostly tight crops that touch the frame, which some marketplaces accept: the threshold is a policy choice.

### 6.6 Latency

Mean 0.82 s per image on an Apple M2 Pro, 0.34 s on one Graviton4 c8g.large with COOL (0.36 s stock), median 0.17 s; the slow tail is GrabCut and denoising. On Lambda arm64 the round trip is 1.0-1.6 s for typical photos.

### 6.7 Statistics, decision boundaries and real phone photos (added 16-22 Oct)

Full numbers in `eval/STATS.md`, `eval/phone24/RESULTS.md`; nothing in the deployed rules was changed. - **Paired tests.** Held-out test: agent 89.6 % (Wilson 95 % CI 88.1-90.9) vs open loop 85.2 % (83.5-86.8); the agent is right where the open loop is wrong on 97 photos, the reverse on 20 (exact McNemar p = 2.6e-13). 106 real catalog photos: 27/33 problems stopped (CI 66-91 %) vs 11/33; 16 vs 0 discordant, p = 3.1e-5. Price: 31/73 good photos go to a person (CI 32-54 %). - **Boundary map** (`docs/boundary_map.png`, `eval/boundary_map.py`): 40 clean held-out photos, one defect at a time, 7 strengths. Blur: most pass up to sigma 0.8 px, almost all reshoot from 1.6 px. Darkness and colour cast: auto-fixed from the first visible step. Noise: accepted below sigma 8, then denoised or reshoot. Overexposure: even at full test strength 40 % still pass as accept, the measured blind spot.




- **Real phone photos (24, one pre-registered run).** Protocol and hashes committed before the run (`eval/phone24/PREREGISTRATION.md`); labels by the author's assistant, not an independent person. All 8 bad photos stopped (CI 68-100 %), and all 16 good ones too: 0/16 passed (CI 0-19 %). Causes: Telegram had shrunk the files to 960 px (rule: 1,000 px), and in a diagnostic run without that rule the background check read wood grain and window-light falloff on a white sheet as clutter (border spread 13.6-21.9 vs 9.0 with almost no edges). Six of the white-sheet scenes re-sent at 1512x2016 confirm it: the resolution rule no longer fires, the light gradient still sends them to a person.
- **Cost per 1,000 photos** on Lambda: under $0.05 at the x86 list price; about 160,000 photos a month fit in the always-free tier.

## 7. Limitations

- Thresholds are hand-set on one catalog source; other photo styles (dark product shots, jewellery on black) need their own calibration. On real phone photos the background rule is too strict: a smooth light gradient on a plain sheet counts as clutter (section 6.7). Separating illumination from texture is the next fix, and it needs a fresh set of phone photos to verify.
- The synthetic defects are controlled approximations; real phone blur and real mixed lighting are harder.
- Background replacement uses GrabCut and refuses scenes; it is not suitable for transparent or very thin products (glass, cables), which mostly go to a human.
- Focus metrics are affected by background contrast (see the example trace).
- White balance is estimated from the background. When a strong cast pushes a white background into clipping, the background under-reports the cast: on 40 dev colour-cast photos the product's residual chroma error is 6.9 (CIE *ab*, mean) after correction, roughly half of the injected cast. We tried von Kries RGB gains and a hybrid; both were worse (8.3).
- The LLM planner is implemented and unit-tested but not evaluated at scale, because Bedrock usage is not free and the serving budget was zero.

## 8. Responsible use

- **Truthfulness:** no generative editing; colour changes are bounded by the measured background cast; any larger change is reverted and sent to a person. Every automatic change is listed in the trace and the seller can keep the original.
- **Human control:** "Needs review" photos have Approve / Reject buttons; auto-fixed photos have "Use fixed / Keep original"; all choices export as JSON.
- **Privacy:** photos are processed in memory and never stored; logs contain decisions and timings only.
- **Abuse and cost:** public endpoint protected by an hourly invocation cap with an automatic kill switch, plus a $1 budget alert.
- **Data rights:** only CC BY 4.0 images (ABO), attributed.

## 9. Reproducibility

Pinned dependencies (`requirements.txt`, `requirements-lambda.txt`), 8 unit tests, dataset builder with a fixed seed, evaluation runner, Lambda build/deploy scripts, and the self-terminating COOL benchmark (see README).

## Architecture diagram

