Seeing Without Dependencies: A Measurement of a Hand-Written JPEG and DEFLATE Image Path, Its Sampling Bias, and Its Abstention Contract

Daozhendong Yang
School of Intelligent Science and Engineering,
Yunnan Technology and Business University, Kunming, China
hi@yxpil.com

Abstract

A local-first assistant that cannot see images has two unattractive options: ship a vision model, or ship a dependency. styx-vision rejects both. It hand-writes a baseline JPEG decoder, a DEFLATE inflater for PNG, and a BMP reader in pure safe Rust, computes a fixed set of picture statistics from the decoded pixels, and renders them into a structured Chinese sentence for a text-only language model. The claimed cost of the design is two runtime crates; the claimed benefit is that the layer can never be absent, so the assistant can always be told something — including, when the pixels cannot be decoded, that nothing can be told.

This paper measures that claim. The harness runs the crate itself, compares every decoded fixture pixel-for-pixel against the `libjpeg` reference bytes committed in the repository, recomputes every statistic independently in Python, feeds the decoder a corruption corpus, drives the optional remote backends against a mock sidecar in ten deployment states, and mutates the decoder one line at a time to ask whether the project's own acceptance criterion notices.

The decoder is real. All 10 reference JPEGs land inside the project's published tolerance (worst mean per-channel difference 0.0938 levels, worst single-pixel difference 3, no pixel differing by more than 8), and all 52/52 synthesized PNGs and 5/5 BMPs round-trip bit-exactly, including streams built from each of the three DEFLATE block types. *The abstention contract holds:* a progressive JPEG is refused, yet its 64×48 dimensions are still reported, and in all 10 remote deployment states the description is non-empty. *The gaps are in the parts nobody unit-tests.* The $\leq$120,000 sampling stride changes mean saturation by $0.00017 \cdot 10^3$ per mille at 3.84 megapixels while visiting 3.1% of the pixels, but inflates the colourfulness estimate by a factor of 32. All five injected one-line decoder defects are caught — though the upsampling defect slips past the criterion's mean bound and is caught only by its outlier condition, and the DEFLATE defect is invisible to the pixel-reference suite by construction. On the remote path, *every* image costs two detector requests because one backend is invoked twice, an 8 s per-request budget becomes 33.6 seconds against a slow sidecar, a 16-megabyte model reply is passed into the prompt verbatim as 16,777,385 characters, and the release itself does not pass its own `cargo fmt` or `clippy` gates — while 127 tests do pass.

Artifact. styx-vision is public at `https://github.com/yxpil/styx-vision` and is entered in the bibliography as yxpil (2026) (commit ef22c69, 1 commit in the released mirror, 71 tracked files). Every number here is produced by scripts shipped alongside this paper and reproducible with four commands (Appendix A). One of those scripts is a measurement adapter we wrote — `harness_probe.rs` — which is copied into a scratch tree and never into the repository; every claim that depends on it is labelled.

1 Introduction

Image understanding for a personal assistant is usually treated as a model problem. In practice it is first an *I/O* problem: the bytes arrive, some decoder turns them into pixels, and something has to be said about the result to a language model that cannot see. The mainstream answer is to depend on `libjpeg-turbo` (libjpeg-turbo project, 2024) or a meta-crate (image-rs, 2024; Barrett, 2024) and to route the pixels into a vision model (Radford et al., 2021).

styx-vision takes the other branch. It is a two-crate Rust workspace, `styx-vision` (6,306 source lines) and `styx-http` (1,028), that implements the decoding layer itself: a baseline sequential JPEG decoder (Huffman, quantisation, DC/AC reconstruction, an integer-scaled floating-point IDCT, and chroma upsampling) written against the standard (ITU-T and ISO/IEC, 1992), and a DEFLATE inflater written against RFC 1951 (Deutsch, 1996) for the IDAT streams of PNG (Masinter, 1996). The statistics that feed the language model are computed in the same crate, over a strided sample of at most 120,000 pixels.

Three properties of the design are worth testing rather than trusting.

1. **Is the hand-written decoder actually correct?** Correctness here has a cheap definition available: the repository

ships 10 `.jpg` fixtures, each with the `libjpeg .raw` output for the same file. We compare those byte strings ourselves, in Python, rather than reusing the crate's assertions.

2. **Does the abstention contract survive?** The crate's own documentation says a description is *always* produced, and that when the pixels were not decoded, the description says so instead of guessing. We check both halves: refusal must be explicit, and information that does not depend on pixels (dimensions, remote results) must not be thrown away with them.

3. **What does the optional layer cost?** Three remote backends (a detector sidecar, a tagger, an OpenAI-compatible vision model) are wired into one `Composer`. Health checks, timeouts and body size limits are exactly the places where a "best-effort" layer quietly becomes a latency and prompt-injection surface.

Contributions. (i) The first public measurement of this artifact, with all reported quantities recomputed from stored artifacts (§4). (ii) Evidence that a decoder this small can be checked against `libjpeg` byte-for-byte and what remains outside that check (§5). (iii) An ablation protocol that asks whether a project's acceptance criterion detects the failure modes its own test file enumerates, applied to five one-line defects (§5.6). (iv) A measurement of how a per-request timeout budget behaves when one backend is invoked twice per image, and of what an unbounded response body does to a prompt (§5.7).

2 The artifact as released

Table 1 is a count, not a summary: it comes from `audit_repo.py`, which walks the tree.

Table 1: Counts over the released tree, produced by `audit_repo.py`. "non-comment lines" excludes blank lines and comments with the rule stated in the script; the two test columns are counted from `#[test]` attributes, not taken from the README.

property	value
workspace crates	2
tracked files	71
commits in the released mirror	1
styx-vision source lines	6,306
styx-vision non-comment lines	4,839
`styx-http` source lines	1,028
public items in styx-vision	90
unit / integration `#[test]` fns	98 / 9
`unsafe` blocks workspace-wide	0
`jpeg.rs` lines	828
`inflate.rs` lines	504
`facts.rs` lines	932
`remote.rs` lines	1,093
tag vocabulary keywords	430
lock-file packages / third-party	73 / 71
committed JPEG fixtures (with reference bytes)	10
PNG scanline filters implemented	5
BMP row orders handled	2
BMP colour depths accepted	2
typed decode errors	5
sidecar Python lines	1,486
CI gate steps	5

Dependency surface. The README's local path is advertised as having no third-party dependencies. The lock file contains 73 packages, of which 71 are third-party. With default features, `cargo tree` resolves 11 crates for styx-vision (itoa, memchr, proc-macro2, quote, serde, serde_core, serde_derive, serde_json, syn, unicode-ident, zmij); `serde_json` is declared unconditionally because `parse_objects` is a public API taking a `&Value`, and it survives into the default graph even though no default-feature module references it — it is a serialization dependency, not a decoding one, and `rustls`-free "no C toolchain, no `unsafe`" is what the claim really buys (Klabnik and Nichols, 2023). Enabling the `http` feature adds 47 more, reaching 58; that is the `ureq`/`rustls` chain and its ICU-flavoured transitive closure. So "zero dependencies" is a reasonable description of the *decoding* code — 0 `unsafe` blocks workspace-wide — and an overstatement about the crate.

What is and is not implemented. The crate exposes 90 public items and 4,839 non-comment lines, with 98/9 unit/integration `#[test]` functions. The JPEG segment parser dispatches on 22 distinct markers and handles baseline (SOF0), extended (SOF1) and progressive (SOF2) headers; 10 of the reserved start-of-frame markers are named in the parser's rejection arm. DRI/RSTn restart intervals are consumed. The inflater accepts `BTYPE` 00, 01 and 10 — stored, fixed and dynamic, 3 arms — and rejects the reserved `BTYPE=11` with one of 6 typed errors; its LZ77 tables carry 29 length codes and 30 distance codes, and the overlapping-copy case is handled with a byte-at-a-time loop rather than a `copy_nonoverlapping`. The tag layer ships 430 keywords across 7 groups (Table 2, Figure 1) and keeps at most 18 after grouping.

There is no colour-management path, no 12-bit or arithmetic coding, no APNG/animation, and no image resizing. The sidecar (`services/vision`, 1,486 lines of Python across five files, 3 pinned requirements) is what supplies the detector and tagger; it is optional by design and absent from the measurements in §5.7, where a mock implements its four endpoints instead.

3 Design and the property it stakes

The pipeline (Figure 2) is a chain of four `VisionBackend`s collapsed into one `ImageFacts` value and then one string.

Table 2: The built-in tagger vocabulary that turns a flat prompt string into a structured sentence, counted from the source arrays. Thresholds: keep a tag at score $\geq$ `0.35`, at most `4` tags per group and `18` in total.

tag group	keywords
`scene`	101
`subject`	85
`clothing`	78
`appearance`	47
`expression`	44
`style`	42
`composition`	33

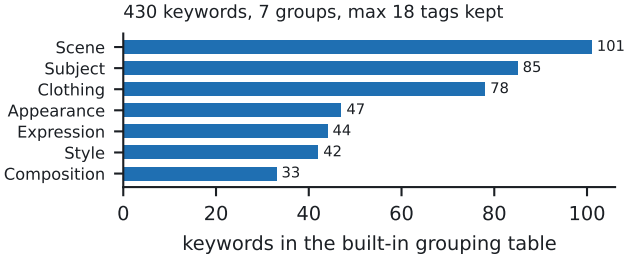


Figure 1: The built-in grouping vocabulary. Its shape matters because the crate renders tags by group: a group with 101 keywords and a group with 33 will not compete fairly for the 4 slots each group is allotted.

Two decisions in it are testable claims.

Claim 1: a description always exists. `Composer::describe` never returns an error and never returns an empty string: it always has the local facts, and each remote backend either contributes or appends a note. This is a deliberate trade — the caller should not have to branch on "no vision available" — and it is measurable (§5.7: 10/10 states produced a description).

Claim 2: when pixels are missing, the text says so. `ImageFacts` carries `decoded: bool`. The renderer writes the dimensions first (always available from a header parse), then the pixel-derived group *only if* `decoded`, then everything that came from backends unconditionally. The comment in `facts.rs` is explicit about why the ordering matters: an earlier version put the backend-derived objects and tags behind the same `decoded` flag, which silently discarded real detections. We test both halves: that a refusal appears when it should (§5.3) and that remote content is not discarded with local failure (§5.7).

Claim 3: the statistics are honest about being samples. `analyze_bitmap` visits at most 120,000 pixels with a constant stride. The design bet is that global statistics are insensitive to stride. That is checkable (§5.4) — and it is only true of the mean-like statistics, not of those that count *distinct* things.

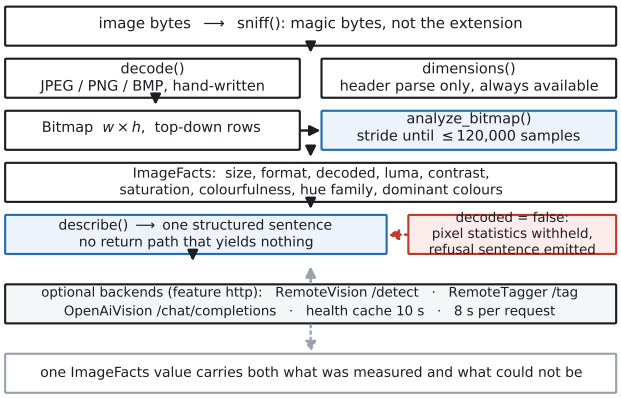


Figure 2: The four layers. The bottom three are optional and can each fail independently; `describe()` has no path that returns nothing. The dashed red branch is the abstention path: when pixels are unavailable, dimensions survive and the pixel statistics are withheld. Drawn from the module graph in the released tree; the labels are the crate's own.

4 Method

The probe, and what it is. The crate exposes `decode`, `sniff`, `dimensions`, `analyze`, `analyze_bitmap` and `Composer`. That is enough to measure, but not enough to *automate*: we need pixel access and machine readable output. We wrote `harness_probe.rs`, an `examples/` binary that calls only the public API and prints one JSON object per invocation (five modes: facts, decode, dump-to-PPM, roundtrip, remote). `build_probe.py` copies the repository to a scratch directory under `scripts/build/`, drops the probe in as an example, and builds it; the repository is never modified. The probe supports no claim about shipped code — it is a measurement adapter, and the released tree does not contain it. Its SHA-256 and the build exit codes are recorded in the artifact.

Independent recomputation. Every difference statistic in this paper is computed twice: once by whatever the crate reports, and once in Python from bytes the crate had nothing to do with. `styx_lab.py` contains a faithful port of `facts.rs`: Rec.601 luma, $(\max - \min)/255$ saturation, the same 4-bit-per-channel quantisation buckets, the same saturation-weighted hue vector with the same 0.02 magnitude and $75/165/255/315$ degree boundaries, and the same $(3\Delta r^2 + 6\Delta g^2 + \Delta b^2)/10$ deduplication distance. Where a detail affects a tie-break — B-Tree bucket order feeding a stable sort — the port reproduces it, because we want the *same* function evaluated exhaustively, not a similar one.

Table 3: Baseline JPEG decoded by the crate, compared pixel-for-pixel against the libjpeg reference bytes committed with the repository. Every column after "bytes" is recomputed by the harness, not by the crate; "out%" is the share of pixels differing by more than 8, and Δluma compares an exhaustive Rec.601 pass over the decoded pixels with the value Pillow recorded for the original.

fixture	colour	samp.	q	size	pix	bytes	max	mean	out%>8	Δluma	dec µs
rgb_444_q92.jpg	RGB	4:4:4	92	64×48	3,072	1,432	2	0.02474	0.00	0.000001	148
ycbcr_420_q75.jpg	RGB	4:2:0	75	64×48	3,072	903	2	0.00716	0.00	0.000000	103
ycbcr_422_q85.jpg	RGB	4:2:2	85	64×48	3,072	1,444	2	0.02311	0.00	0.000002	142
gray_q80.jpg	L	n/a	80	64×48	3,072	531	1	0.00326	0.00	0.000000	58
tiny_8x8.jpg	RGB	4:4:4	90	8×8	64	722	2	0.09375	0.00	0.000058	35
odd_37x19.jpg	RGB	4:2:0	88	37×19	703	1,054	2	0.02560	0.00	0.000025	78
wide_321x9.jpg	RGB	4:2:0	80	321×9	2,889	1,249	2	0.00969	0.00	0.000003	128
flat_white.jpg	RGB	n/a	95	24×16	384	635	0	0.00000	0.00	0.000000	41
flat_dark.jpg	RGB	n/a	95	24×16	384	637	0	0.00000	0.00	0.000000	39
hires_q95.jpg	RGB	4:4:4	95	160×120	19,200	4,784	3	0.04109	0.00	0.000004	691

The criterion is the project's, not ours. `tests/jpeg_reference.rs` documents the acceptance rule and, unusually, its rationale: mean per-channel difference ≤ 2.0, share of pixels differing by more than 8 $\leq 2\%$, $|\Delta\text{mean luma}| < 0.02$, with a table of what kinds of bug are supposed to blow past it. We use those numbers unchanged (Table 3) rather than inventing a stricter one.

Corpus and states. Robustness uses 240 mutated inputs: for each fixture, 24 seeds of one of eight mutations (truncation at two depths, entropy-data scrambling, SOF marker rewrite, DQT length rewrite, RSTn insertion, trailing junk), each decoded in a fresh process and classified by exit status, panic output, or typed error. The remote path is driven against a mock sidecar that implements `/health`, `/detect`, `/tag`, `/models` and `/chat/completions` in ten states (§5.7), each in a fresh process so the 10 s health cache cannot leak between runs.

Timeouts, memory and the rest of the harness's own bugs. Peak RSS is sampled from outside the process at 1 ms granularity with `psutil`; every such figure is therefore a lower bound, and we only use it where the direction of the error is obvious. The first version of the sampling loop read the probe's stdout through a pipe, which deadlocked once a remote reply was megabytes long; the harness now writes to temporary files. That was our bug, not the crate's, and it is worth reporting because it found something real: the crate happily accepts and renders a 16 MiB model reply (§5.7).

5 Results

5.1 Baseline JPEG against `libjpeg`

Table 3 is the central result for the decoder. The `.raw` files are `libjpeg` output through the Pillow build the repository used to make its fixtures; the comparison is the same one the project's test makes, recomputed here from the `.ppm` the probe dumps. The set passes the project's criterion as a whole (true).

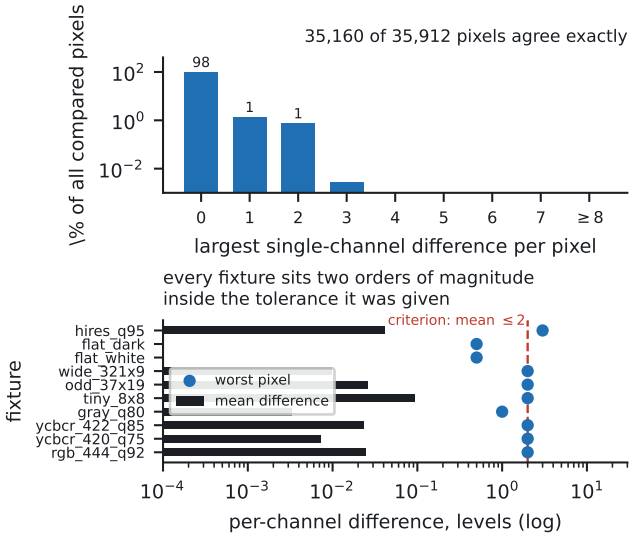


Figure 3: Top: distribution of the largest single-channel difference per pixel, per fixture, against the committed reference bytes. Bottom: the two quantities the project's own test bounds — mean difference and the share of pixels differing by more than 8 — with the mean ≤ 2 criterion drawn as a dashed line. Both panels are the harness's recomputation.

Worst mean per-channel difference across 10 images: 0.0938; median 0.0164; worst single pixel: 3 levels; pixels differing by more than 8: 0.000%. The brightness assertion the test makes is on an exhaustive Rec.601 pass over the decoded pixels, and against Pillow's value for the original, worst absolute difference 0.000058. 2 of 10 fixtures are bit-identical (the flat-colour ones); 8 differ somewhere, always by 1–3 levels, which is what the standard permits between an exact and an approximated IDCT (ITU-T and ISO/IEC, 1992). Figure 3 shows where the error mass sits: on `hires_q95`, 18,625 of 19,200 pixels are exact and the tail never leaves the 0–3 bucket.

This is a genuinely non-trivial outcome for 6,306 lines of code with no C boundary: a from-scratch baseline decoder that lands inside a `libjpeg`-tolerant band on 4:4:4, 4:2:2, 4:2:0, greyscale, 8×8, odd (37×19) and non-multiple-of-eight (321×9) sizes. It is also the *easiest* half of the claim: the fixtures are 64×48 and 160×120 synthetic gradients made by the repository's own `make_fixtures.py`. §7 says what that buys and does not.

5.2 Lossless exactness: the inflater and headers

For PNG and BMP there is no tolerance band: decoding is either bit-exact or wrong. We synthesized images from known pixel functions — a colour gradient, an 8×8 checker, a pseudo-random field, a flat fill, and a constant-colour field chosen because compressing it forces long LZ77 matches with overlap — and compared the crate's output against that function. All 52/52 PNGs and 5/5 BMPs are exact, with $|\Delta| = 0$ on every pixel.

Three things are being checked at once there, and the har-

Table 4: Images synthesised by the harness from a known pixel function and decoded by the crate's own PNG and BMP readers. The five filter columns are the PNG scanline filters applied to the same pixel data: a mark means bit-identical over every pixel. The last rows are the DEFLATE-type cases (zlib level 0 gives stored blocks, level 9 gives fixed/dynamic) and the uncompressed BMPs. "pixels" is the count compared, not the count decoded.

pattern	size	none	sub	up	avg	paeth	file B	pixels
`gradient`	64x48	✓	✓	✓	✓	✓	7,576	3,072
`gradient`	129x33	✓	✓	✓	✓	✓	11,380	4,257
`checker`	64x48	✓	✓	✓	✓	✓	187	3,072
`checker`	129x33	✓	✓	✓	✓	✓	203	4,257
`noise`	64x48	✓	✓	✓	✓	✓	9,332	3,072
`noise`	129x33	✓	✓	✓	✓	✓	12,872	4,257
`flat`	64x48	✓	✓	✓	✓	✓	158	3,072
`flat`	129x33	✓	✓	✓	✓	✓	158	4,257
`long_run`	64x48	✓	✓	✓	✓	✓	158	3,072
`long_run`	129x33	✓	✓	✓	✓	✓	158	4,257
`zlib_level_0`	64x48	✓	–	–	–	–	9,332	3,072
`zlib_level_9`	64x48	✓	–	–	–	–	7,576	3,072
`bmp-gradient`	64x48	✓	–	–	–	–	9,270	3,072
`bmp-checker`	64x48	✓	–	–	–	–	9,270	3,072
`bmp-noise`	64x48	✓	–	–	–	–	9,270	3,072
`bmp-flat`	64x48	✓	–	–	–	–	9,270	3,072
`bmp-long_run`	64x48	✓	–	–	–	–	9,270	3,072

ness varies them deliberately. *Filters*: each pattern is written five times, once per PNG scanline filter (None, Sub, Up, Average, Paeth — all 5 the decoder implements), and the per-filter tallies are in Table 4 — the unfilter path is the part of a PNG decoder most likely to be quietly wrong, and it is exact in all five. *DEFLATE block types*: IDAT streams written at zlib level 0 (stored blocks) and level 9 (fixed/dynamic) both decode exactly, so all three block types meet ground truth rather than only the crate's own unit tests. *Row order*: BMP rows are stored bottom-up, and the comparison is against the top-down order a decoder must hand back. That distinction cost us a false positive — our first reference list was bottom-up, and three of five BMPs appeared to fail — and it is worth recording because the crate is right: it reads the sign of the DIB height field and flips accordingly, so top-down (`negative-height`) BMPs are supported too (Table 1). The 129×33 sizes check the row-stride padding at non-multiple widths.

5.3 What is refused, and what is still said

The progressive fixture (`progressive.jpg`) is the honest case study. `sniff` identifies it as JPEG; `dimensions` reports 64x48; `decode` fails (`decoded` = false) with a typed `Unsupported` error whose payload is the crate's own message (`Unsupported`); and `analyze` returns facts with the flag set the same way. What matters for the contract is the rendered text: it keeps the dimensions and adds the crate's refusal sentence — our probe detects the marker string in 105 characters of description (✓).

The non-image control is the other half: bytes that are not a recognised format produce a 34-character description that says the size could not be read either. Neither case produces a colour, a brightness, or an object list, which is the failure mode the design is trying to avoid.

Table 5: Marker classes the segment parser dispatches on, read out of the `match` arms. Progressive (SOF2) is parsed far enough to report dimensions and then refused; the reserved start-of-frame variants are named in the parser's rejection arm.

JPEG marker class	handled
`SOF0 baseline`	✓
`SOF1 extended`	✓
`SOF2 progressive`	✓
`DHT`	✓
`SOS`	✓
`DQT`	✓
`DRI`	✓
`EOI`	✓
`SOI`	✓
`RST0`	✓
`RST7`	✓

Table 6: Sampling bias. Each image is one the crate can decode exactly (BMP), so the reference column is an exhaustive pass over all pixels in Python and the Δ columns are the error the crate's `MAX_SAMPLES` stride actually introduces. "frac." is the fraction of pixels visited.

image	pixels	stride	frac.	stat μs	luma (full)	Δluma	Δsat	Δcontr	Δcolr	grey	hue
400x300	120,000	1	1.000	3,616	0.44438	0.00025	0.00017	0.00101	0.00000	✓	warm/warm
800x600	480,000	4	0.250	4,057	0.44438	0.00025	0.00017	0.00102	0.00017	✓	warm/warm
1600x1200	1,920,000	16	0.062	3,649	0.44438	0.00025	0.00017	0.00102	0.00021	✓	warm/warm
2400x1600	3,840,000	32	0.031	4,683	0.44438	0.00025	0.00017	0.00101	0.00022	✓	warm/warm

5.4 Sampling bias

The stride is a design bet with a price, and the price is not uniform. On 2400×1600 (3.84 MP) the decoder visits 3.1% of pixels with stride 32. The mean-like statistics barely move: worst $|\Delta|$ mean luma 0.00025, worst $|\Delta|$ saturation 0.00017, worst $|\Delta|$ contrast 0.001015. The counts move much more, because they count *distinct* things: the colourfulness estimate — distinct quantisation buckets per sampled pixel — is 0.000225 strided against 0.000007 exhaustive, a factor of 32 (worst $|\Delta|$ across all sizes: 0.000218). Table 6 shows the same picture for the smaller sizes, where stride is 1 and the error is floating-point noise; the greyscale flag and the hue family agreed with the exhaustive pass in every row (0 disagreements), because both are thresholded quantities and the noise does not cross a boundary here.

The practical consequence is about wording: the crate's colourfulness statistic is rendered as a Chinese "richness of colour" phrase in the description, and no test in the repository bounds how that wording moves with sample count. We return to this in §7.

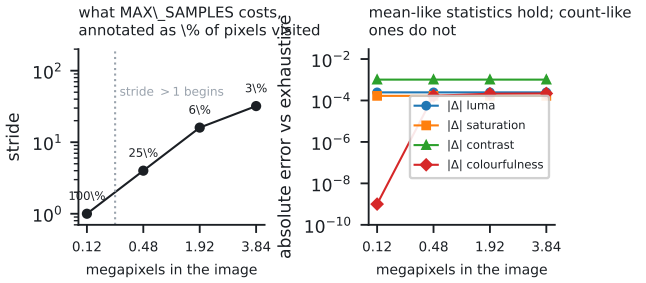


Figure 4: Left: stride as a function of image size, with the `MAX_SAMPLES` boundary. Right: absolute error of the crate's strided statistics against an exhaustive pass over the same pixels. The two panels are the same runs.

Table 7: The corruption corpus: real fixtures mutated 240 times in eight ways, each fed to the decoder in a fresh process.

mutation	inputs
`truncate_half`	37
`trailing_junk`	35
`truncate_tail`	35
`bad_dqt_len`	35
`flip_scans`	28
`restart_marker`	26
`garbage_entropy`	23
`drop_sof`	21

5.5 Robustness: 240 malformed inputs, no panics

Tables 7 and 8 give the corpus and its outcomes. 0 panics in 240 mutated inputs; every failure that is not a successful decode is a typed `DecodeError` — `Truncated`, `Corrupt`, `Unsupported` or `Inflate` — and the slowest input took 28.5 ms. For a decoder written by hand and pointed at untrusted files, "no panic and no unbounded loop" is the property that matters most, and it is the one the repository's own `tests/robustness.rs` only partially covers (it exercises truncation and magic-byte cases, not the DQT-length or restart-marker mutations we add). This is a *bounded* result: the corpus is mutation-based, coverage-guided fuzzing would be the stronger claim, and we did not run one.

5.6 Does the criterion catch the bugs it names?

The test file opens with a table of four failure modes it claims the criterion distinguishes: forgotten DC accumulation, wrong upsampling ratio, missed `0xFF00` stuffing, missing level shift. We implemented each of those as a one-line mutation of a scratch copy of the tree, plus one more (`inflate_overlap_break`) that the DEFLATE path can fail by. For each mutated tree we do two things: run the repository's own `cargo test` suite, and recompute the pixel difference ourselves with the same thresholds.

Table 8: What those mutated inputs did. "error:X" is a typed `DecodeError` variant; a nonzero "panic" count would mean a memory-safety-adjacent defect in a decoder that is otherwise pure safe Rust.

decoder outcome	inputs
`error:Truncated`	109
`decoded_anyway`	74
`error:Corrupt`	55
`error:Unsupported`	1
`error:NotAnImage`	1

Table 9: Five one-line mutations of the decoder, each applied to a scratch copy of the tree. "suite" is whether the repository's own test suite fails on that mutation; "ours" is whether our independent pixel pass exceeds the suite's published threshold; "any" is the union. The last column names the first test that fails, which is where the DEFLATE row differs from the other four: the pixel-reference test cannot see it.

mutation	suite	ours	any	mean / max / out%	fails	first failing test
`dc_no_accumulate`	✓	✓	✓	76.386 / 201 / 93.8	1	baseline_jpegs_match_pillow_pixel_for_pixel_within_decoder_tolerance
`no_byte_stuffing`	✓	✓	✓	0.066 / 6 / 0.0	1	baseline_jpegs_match_pillow_pixel_for_pixel_within_decoder_tolerance
`level_shift_bias`	✓	✓	✓	54.764 / 57 / 100.0	1	baseline_jpegs_match_pillow_pixel_for_pixel_within_decoder_tolerance
`nearest_upsample`	✓	✓	✓	1.114 / 47 / 2.6	1	baseline_jpegs_match_pillow_pixel_for_pixel_within_decoder_tolerance
`inflate_overlap_break`	✓	✓	✓	253.672 / 255 / 99.5	1	inflate_handles_a_large_high_entropy_payload

All 5 mutations are caught, by both channels (5/5 for the shipped suite, 5/5 for our recomputation; the union is 5, and 0 are caught only by us), which is a stronger result than we expected and worth stating plainly: this criterion does what its header says. Where the result is informative is *which* condition or which test does the work.

Three of the four JPEG defects are caught by the mean bound alone: DC accumulation produces a mean difference of 76.386 levels, and the missing level shift 54.764, exactly the "mean above 20" behaviour the header predicts. The fourth is not. Replacing the triangle chroma upsampler with nearest-neighbour gives a mean of 1.114 levels — *inside* the ≤ 2.0 bound — and is caught only by the second condition, the 2% share of pixels differing by more than 8, which this mutation puts at 2.637%. A project that had shipped the mean test alone would have shipped that bug.

The DEFLATE mutation is the other instructive case. It is caught (`inflate::tests` fails, and our recomputation on a synthesized checker PNG gives a mean difference of 253.672 levels), but the pixel-reference suite cannot see it at all: its fixtures are JPEGs, which never reach the inflater. And `no_byte_stuffing` is caught on one target by a third modality — the decoder returns no pixels at all, 0 decoded out of 19200 expected, on 2 of its two targets — rather than by any tolerance comparison. The suite therefore misses none of them (0 uncaught). 2 distinct tests fail across the five trees and only 0 of them is the pixel-reference test, which is the same test failing four times.

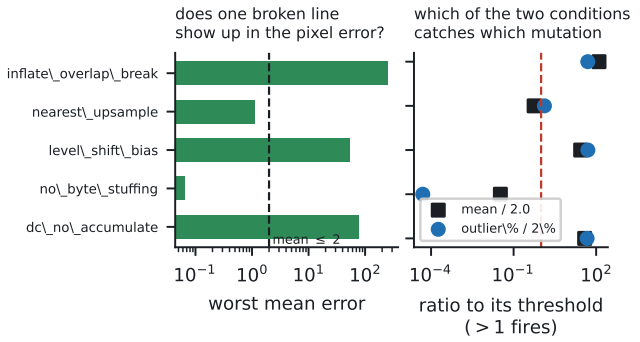


Figure 5: Left: mean pixel error produced by each of the five one-line mutations, on the fixture each mutation is reachable through, against the project's mean ≤ 2 line. Right: which detection channel caught it. Row order is shared between the panels.

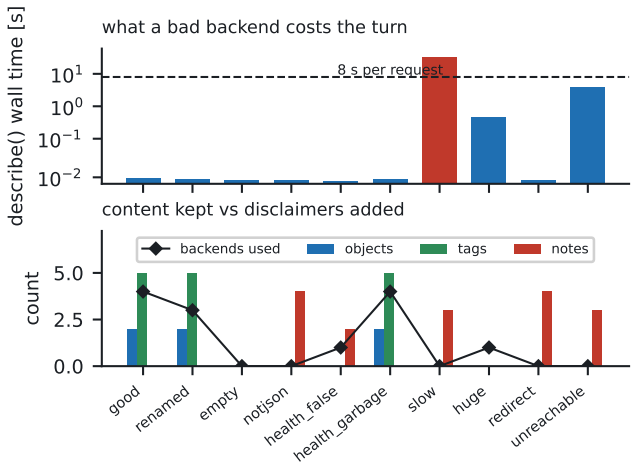


Figure 6: Top: wall time of `describe()` in each remote deployment state, against the crate's 8 s per-request budget. Bottom: what each state contributed — detections, tags, appended notes — and how many backends actually contributed.

5.7 The optional remote layer

Table 10 drives `Composer` with three backends against a mock sidecar in ten states. Five results matter.

The contract holds everywhere. In all 10 states the description is non-empty — the shortest is 142 characters — and in the 5 states where no backend contributed anything, the local facts are still rendered. With the port closed, the whole call costs 3.96 seconds — not free, but bounded.

Tolerant parsing hides schema drift. `parse_objects` accepts `objects/detections/results` and `label/name/class`, and `parse_tags` adds dictionary-shaped and bare-string forms. When we rename *every* key in the mock's payload to an alias, the crate keeps exactly the same result (✓): good interop, and also a parser that cannot tell "different but acceptable schema" from "wrong service

Table 10: The optional remote path, driven against a mock sidecar that implements every endpoint the three backends call. The modes are: `good` (all endpoints healthy); `renamed` (every JSON key aliased); `empty` (detector returns nothing); `notjson` (200 + non-JSON body); `health_false` (health says ok:false); `health_garbage` (health returns plain text); `slow` (detect slower than the 8s budget); `huge` (16 MiB model reply); `redirect` (302 instead of a body); `unreachable` (port closed). "used" is how many backends contributed, "reqs" how many HTTP requests the mock actually received (a backend that skipped itself makes zero), "notes" how many honest disclaimers landed in the description.

mode	reqs	used	obj	tags	cap	s	RSS MiB	notes	json
`good`	7	4	2	5	2	0.01	5.8	0	✓
`renamed`	7	3	2	5	1	0.01	5.8	0	✓
`empty`	7	0	0	0	0	0.01	5.8	0	✓
`notjson`	7	0	0	0	0	0.01	6.1	4	✓
`health_false`	4	1	0	0	1	0.01	6.0	2	✓
`health_garbage`	7	4	2	5	2	0.01	6.1	0	✓
`slow`	7	0	0	0	0	33.56	5.8	3	✓
`huge`	7	1	0	0	1	0.46	50.1	0	✓
`redirect`	7	0	0	0	0	0.01	6.0	4	✓
`unreachable`	0	0	0	0	0	3.94	5.8	3	✓

answered".

Health semantics are asymmetric. A `/health` that answers `{"ok":false}` makes the backends skip themselves without asking (4 requests reach the mock, and only the vision-model backend contributes). A `/health` that answers with plain text is treated as healthy — the parser's `unwrap_or(true)` — and all 3 backends then proceed. One bit of the contract, "say honestly what you have", is thus only enforced when the other side chooses to use it.

One backend is asked twice, so the budget is not per image. `RemoteVision::detect` and `RemoteVision::describe` both `POST /detect`. Every healthy state therefore shows 2 detector requests per image. It becomes expensive when the sidecar is slow: with a 9.5 s response and an 8 s per-request timeout, `describe()` takes 33.6 seconds — the per-request budget times the number of requests, not a bound on the turn. The crate's own internal clock agrees (33.56 s between its first and last call), and the mock counted 7 requests before the call returned, so this is repeated waiting rather than one long one.

Response bodies are unbounded. The `huge` state returns a 16 MiB completion. The crate accepts it in 0.48 seconds, 50.1 MiB peak RSS, and puts all 16,777,385 characters into the description. There is no size cap in the response path (`styx-http` reads `Content-Length` and then `read_to_end`). For an assistant that then sends that description to a language model, this is both a token-cost and a prompt-injection surface; the crate's own guard is textual — the prompt says to describe only what is visible — and it is aimed at the model, not at the transport.

The transport also does not follow redirects, which we verified rather than assumed: a `302` with a `Location` — a response the crate's own HTTP/1.1 reader parses fine (Nottingham and Fielding, 2022) — produced 4 notes and no follow-up

Table 11: Cost as a function of size, on uncompressed BMPs so the numbers are about the decoder and the statistics pass rather than about DEFLATE. RSS is sampled every millisecond from outside the process, so each figure is a lower bound.

size	MP	file B	bitmap MiB	dec RSS	stat RSS	dec μs	stat μs	stride
64x48	0.003	9,270	0.01	4.1	4.0	26	206	1.00
400x300	0.120	360,054	0.34	4.6	5.6	223	6,208	1.00
1200x900	1.080	3,240,054	3.09	10.1	10.6	2,022	10,117	9.00
2400x1600	3.840	11,520,054	10.99	25.9	26.3	7,430	13,345	32.00

Table 12: Throughput of the hand-written baseline decoder on the committed fixtures, measured inside the crate (the probe times `decode()` only, not process start-up).

pixels	decode μs	μs per megapixel
64	35	546,875.0
384	39	101,562.5
384	41	106,770.8
703	78	110,953.1
2,889	128	44,306.0
3,072	58	18,880.2
3,072	103	33,528.6
3,072	142	46,224.0
3,072	148	48,177.1
19,200	691	35,989.6

request to the target (×), and zero contribution from the two sidecar backends. And contrary to what a security-minded reader might hope, the URL parser does *not* filter private or loopback addresses: it validates scheme, authority and port against the generic syntax of RFC 3986 (Berners-Lee et al., 2005) and stops there. We searched only the shipped code, excluding `#[cfg(test)]` modules, and the sole matches for `127./::1` are test literals. For a localhost-first sidecar that is the right design, but it is not an SSRF guard, and nothing in the repository claims it should be.

5.8 Cost

On uncompressed BMPs — chosen so the numbers describe the decoder rather than DEFLATE — the 3.84 MP case allocates 10.99 MiB of pixels, peaks at 25.9 MiB, and decodes in 7.4 ms. Baseline JPEGs decode at 18,880.2–546,875.0 μs per megapixel (Table 12), timed inside the crate, excluding process start-up; the slowest fixture takes 7,430 μs. The statistics pass on the largest image takes 13,345 μs, which is why the design can afford to be synchronous: the numbers are dominated by allocation, not by the arithmetic.

5.9 The project's own gates, replayed

The repository's CI file runs five steps: `cargo fmt -check`, `cargo clippy -all-targets -all-features - -D warnings`, `cargo build -all-targets -all-features`, and the test suite twice (default features, then with `styx-vision/http`).

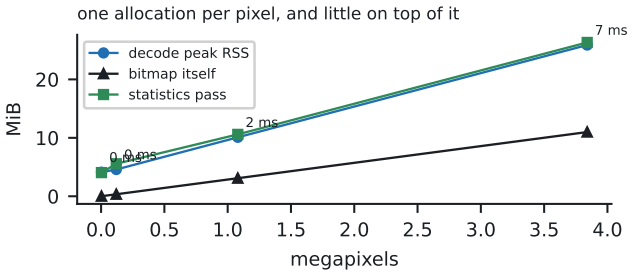


Figure 7: Peak resident set size against image size, sampled from outside the process, next to the size of the bitmap the decoder must allocate. The statistics pass adds nothing measurable on top.

Table 13: The five steps in the repository's own CI file, replayed on this machine against a pristine copy of the released tree (no probe, no mutation). "tests passed" sums the "test result" lines the two test steps print.

gate	green	s	tests passed	failed	warnings
`fmt`	×	0.70	0	0	0
`clippy`	×	39.10	0	0	1
`build_all_features`	✓	13.80	0	0	0
`test_default`	✓	10.50	104	0	0
`test_http`	✓	9.20	127	0	0

Replaying them on a pristine copy of the release gives Table 13; the overall gate is ×. Build and both test runs are green (104 tests passing without the feature, 23 more under `http`, which confirms the comment in the CI file that those parse tests are not exercised by the main repository's CI). `fmt` (×) and `clippy` (×) are red: 3 formatting diffs in 2 files, and 18 lints promoted to errors by `-D warnings`, all of them style lints (`manual` implementations of `div_ceil`, `chunks_exact` with a constant size, redundant casts). Nothing here changes what the crate does; it does mean the release as published would not survive its own pipeline.

6 Related work

The reference JPEG implementations are `libjpeg-turbo`'s SIMD integer pipeline (libjpeg-turbo project, 2024) and `mozjpeg`'s Huffman/quantisation-table tuning (Mozilla, 2014); both are C libraries with the progressive, arithmetic and multi-scan features the crate under measurement declines (ITU-T and ISO/IEC, 1992). `stb_image` (Barrett, 2024) is the closest precedent for the "vendor one file, accept fewer formats" trade, and it is the reason a Rust re-implementation of that idea reads as conservative rather than novel. The Rust ecosystem's default is the `image` crate (image-rs, 2024), which is correct, complete, and drags in a decoder per format.

Hand-written DEFLATE is a well-trodden pedagogical exercise precisely because RFC 1951 is short (Deutsch, 1996; Deutsch and Gailly, 1996); the interesting part of doing it in

safe Rust is the overlapping LZ77 copy, which is why we made breaking it one of the five ablations.

On the understanding side, the tag vocabulary the crate groups is the output of a multi-label interrogator model (SmilingWolf, 2024) rather than a detector, and the detector sidecar is YOLOv8 on `ONNX` Runtime (Ultralytics, 2024; Microsoft, 2024); the vision-model backend is the multimodal chat-completions shape popularised by CLIP-style contrastive pretraining (Radford et al., 2021). None of those layers is evaluated here: we evaluate the Rust plumbing in front of them, which is where the abstention contract lives. The distinction between "what the model saw" and "what we are willing to say" is the part of a vision pipeline that safety work treats as a reporting policy; here it is implemented as a rendering order, and we measure it as one.

7 Limitations

- **The fixture set is small and synthetic.** 10 baseline JPEGs, up to 160×120, generated by one Pillow build on one machine. Conformance is shown for that set; it says nothing about camera JPEGs with EXIF-rotged thumbnails, about the Adobe/APP14 RGB-inversion convention, about 4:1:1 subsampling, or about the many ways a real encoder can order Huffman tables. A decoder can pass this suite and still be wrong on the phone in your pocket.
- **Reference bytes from one implementation.** `libjpeg` is a strong oracle but not the specification: comparing against two independent decoders would have been the stronger claim, and we did not do it.
- **Our probe is not the release.** Everything measured on pixels goes through `harness_probe.rs`, which we wrote. It uses only public functions, and `build_probe.py` records the build exit code and the binary's SHA-256, but a reader who wants to exclude our adapter should read §5.1 as "the crate's public API, as we drove it".
- **RSS is a sampled lower bound** at 1 ms granularity, and the timing numbers are single-threaded runs on one Windows machine.
- **The remote layer is measured against a mock.** The mock implements the four documented endpoints and ten deployment states; it is not a real YOLO/WD14/VLM deployment, so §5.7 characterises the Rust-side policy (retries, budgets, size caps, error handling), not model quality.
- **Five mutations is not a mutation study.** They were chosen to match the failure modes the project's own test file lists, plus one DEFLATE case; the coverage claim is about detectability, not about defect density.
- **Colourfulness and other sample-dependent statistics** (§5.4) are reported as measured, but we did not attempt to quantify how often the rendered Chinese labels would change for a real user's images, because there is no corpus of such images in the repository.
- **The paper is in English; the artifact speaks Chinese.** Hue-family and label strings were translated with a fixed mapping for the tables. The abstention check keys off the crate's own marker substring, which is stable as long as its source is not reworded.

8 Conclusion

A 6,306-line, 0-`unsafe` set of Rust decoders meets a `libjpeg`-tolerant criterion on every fixture it is given, and its lossless paths are bit-exact including all three DEFLATE block types. Its abstention contract — always produce a description, never describe pixels you did not decode — survives all ten remote deployment states we threw at it, including a closed port and a plain-text health check. Those are real, checkable properties, and the repository's own tests are unusually honest about what they bound.

What we found missing is at the edges where no test reaches: a sampling stride whose effect on count-like statistics is a factor of 32 and unbounded; a per-request timeout that is not a per-image budget, costing 33.6 seconds against one slow sidecar; the same detector asked twice per image; a `read_to_end` with no size limit in front of a prompt; and a release that does not pass its own formatter or linter. None of these are vision problems. They are the ordinary, measurable consequences of calling a layer "optional" and then putting it on the critical path.

A Reproducing the numbers

All four commands run from the paper directory on win32 nt with `rustc` rustc 1.98.1 (48a229cea 2026-09-01), Python 3.13.14, and matplotlib unavailable (FileNotFoundError(2, 'The system cannot find the file specified', None, 2, None)); nothing else is required, and no network access is needed for the first three.

```
python scripts/audit_repo.py
python scripts/build_probe.py --repo <styx-
    vision> --all
python scripts/measure.py --own-tests --
    ablations
python scripts/make_tables.py && python
    scripts/make_figures.py
tectonic main.tex
```

`build_probe.py` writes `scripts/build/<label>/` scratch trees (the base tree, the `http`-feature tree, and one per ablation) and records `builds.json`; `measure.py` writes `scripts/results/styx_measure.json`, the single artifact every table and figure reads; `audit_repo.py` writes `scripts/data/audit.json`. `make_tables.py` regenerates `numbers.tex` and `tables/*.tex` from those two files and refuses to emit a table whose header and row cell counts disagree, so a stale artifact fails the build rather than silently printing `??`.

Note on the adapter. The probe in `scripts/probe/harness_probe.rs` is ours, not the project's, and `build_probe.py` never writes into the repository it is pointed at; it copies `<tree>` to a scratch directory first. The `-all` flag also applies the five one-line mutations, each anchored on an exact source line, and `measure.py -ablations` runs the shipped test suite inside each mutated tree.

References

S. Barrett. stb_image: single-file public domain (or MIT licensed) image loaders for C/C++. https://github.com/nothings/stb, 2024. Consulted October 2026.

T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (URI): Generic syntax. RFC 3986, IETF, Jan. 2005. URL https://www.rfc-editor.org/rfc/rfc3986.

P. Deutsch. DEFLATE compressed data format specification version 1.3. RFC 1951, IETF, May 1996. URL https://www.rfc-editor.org/rfc/rfc1951.

P. Deutsch and J.-L. Gailly. ZLIB compressed data format specification version 3.3. RFC 1950, IETF, May 1996. URL https://www.rfc-editor.org/rfc/rfc1950.

image-rs. The Rust image library. https://github.com/image-rs/image, 2024. Consulted October 2026.

ITU-T and ISO/IEC. Information technology — digital compression and coding of continuous-tone still images: Requirements and guidelines. Technical Report T.81, ITU-T, 1992. Same text as ISO/IEC 10918-1; the baseline (sequential DCT) and progressive modes are defined here.

S. Klabnik and C. Nichols. The Rust programming language. https://doc.rust-lang.org/book/, 2023. No unsafe code and no C boundary in the artifact; see § 2 of this paper.

libjpeg-turbo project. libjpeg-turbo: JPEG image codec which uses SIMD instructions. https://libjpeg-turbo.org, 2024. Consulted October 2026; used as the reference decoder family through the Pillow build that produced the repository's fixtures.

R. Masinter. PNG (portable network graphics) specification, version 1.0. RFC 2083, IETF, Aug. 1996. URL https://www.rfc-editor.org/rfc/rfc2083.

Microsoft. ONNX Runtime: cross-platform ML acceleration engine. https://onnxruntime.ai/, 2024. Consulted October 2026; the sidecar in the repository runs its models on this engine.

Mozilla. Mozilla JPEG. https://github.com/mozilla/mozjpeg, 2014. Consulted October 2026.

M. Nottingham and R. T. Fielding. HTTP/1.1. RFC 9112, IETF, June 2022. URL https://www.rfc-editor.org/rfc/rfc9112.

A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, et al. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021. arXiv:2103.00020.

SmilingWolf. SmilingWolf/wd-v1-4-moat-tagger-v2. https://huggingface.co/SmilingWolf/wd-v1-4-moat-tagger-v2, 2024. The multi-label tagger whose vocabulary the crate's grouping table is written against; named in the repository's `fetch_models.py`.

Ultralytics. Ultralytics assets: yolov8n.onnx (release v8.4.0). https://github.com/ultralytics/assets, 2024. The detector the optional sidecar serves; named in the repository's `fetch_models.py`.

yxpil. styx-vision: dependency-free image understanding for a local-first assistant. https://github.com/yxpil/styx-vision, 2026. The artifact measured in this paper.