

Review Loop: Spec

A human-in-the-loop QA system for AI-generated UGC variants. It sits **around** an existing automated scorer rather than replacing it: it takes the scorer's flags, makes manual review fast, captures every human decision as a label, and reports where the scorer is wrong so it can be tuned.

One-line pitch: the scorer is the engine; this is how it gets better every batch.

1. Problem

AI variant pipelines generate thousands of videos. An automated scorer catches some defects (extra fingers, face drift, product morphing, flicker), but not reliably, so a person still reviews batches by hand. Today that manual work:

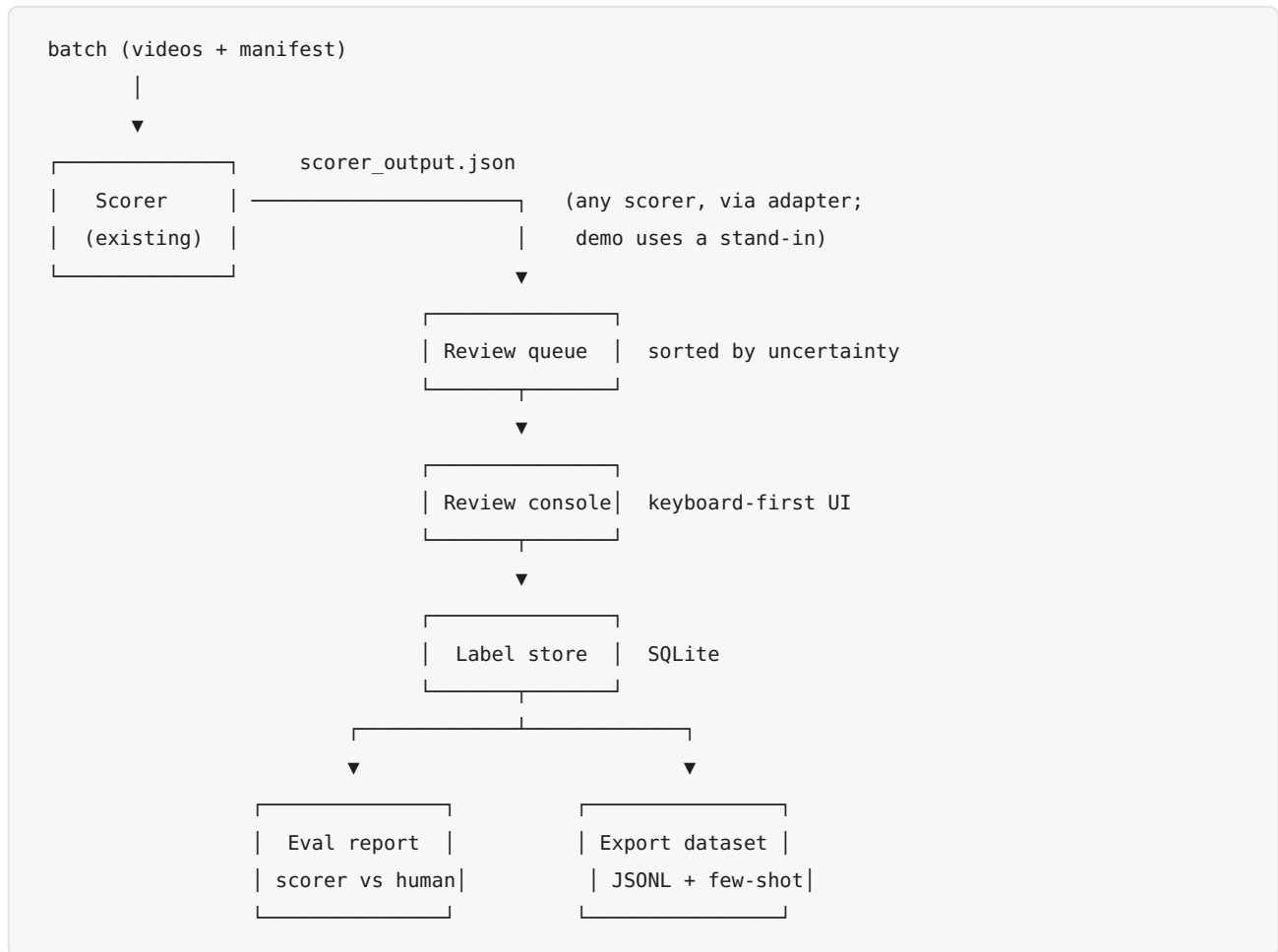
1. **Is slow:** every video gets watched, including the obvious passes
2. **Disappears:** each approve/reject decision is made once and lost
3. **Doesn't improve the scorer:** nobody can say which defects it misses or which it over-flags, so it stays unreliable and the manual load never shrinks

2. Goals

goal	metric
Make manual review faster	median seconds per variant reviewed
Review fewer videos	% of batch auto-cleared at a safe threshold
Turn reviews into data	100% of decisions logged with defect type and timestamp
Show exactly where the scorer fails	per-defect precision and recall, false-positive examples
Give the engineer a concrete fix list	threshold recommendation + labeled export

Non-goals: replacing the existing scorer, generating video, judging creative quality (optional appendix only).

3. System overview



Every component is a module with `run(input: dict) -> dict`, so each can be wrapped as a node in any custom node framework.

4. Inputs

```
batch/  
  manifest.csv  
  originals/      ugc_001.mp4 ...  
  variants/       ugc_001_v01.mp4 ...  
  scorer_output.json  # optional; if missing, run the stand-in scorer
```

manifest.csv

column	example
variant_id	ugc_001_v01
original_id	ugc_001
change_type	microchange / avatar_swap / product_swap / hook_swap
change_description	"shirt changed to red"

4.1 Scorer adapter (the integration point)

The console doesn't care which scorer produced the flags. Any scorer is converted into this schema by a small adapter function:

```
{
  "variant_id": "ugc_001_v01",
  "verdict": "pass | fail | unsure",
  "confidence": 0.82,
  "flags": [
    {
      "type": "hand_deformity",
      "t_start": 4.1,
      "t_end": 4.6,
      "confidence": 0.71,
      "note": "extra finger on left hand"
    }
  ]
}
```

This is the selling point for integration: plugging in a real scorer is writing one adapter, not rebuilding anything.

4.2 Stand-in scorer (demo only)

For the demo, a simple scorer produces the flags so the loop has something to review: - ffmpeg blackdetect, freezedetect, aspect ratio and resolution checks - Frame-to-frame SSIM spikes that don't line up with real cuts (flicker) - Claude vision on sampled frames for hand/face/object artifacts (prompt in section 10)

It is deliberately imperfect. Its mistakes are what the eval report exposes.

5. Defect taxonomy

One fixed list, shared by the scorer adapter, the console hotkeys, and the eval report.

key	type	description
1	hand_deformity	extra, missing, or fused fingers; warped hands
2	face_drift	identity changes mid-clip, uncanny or melting face
3	product_morph	product shape, label, or color changes across frames
4	flicker	temporal instability, textures crawling
5	object_merge	objects melting into each other or the body
6	text_garble	unreadable or mutated on-screen text or labels
7	lipsync	mouth out of sync with audio (avatar swaps)
8	change_failed	the intended edit didn't happen (shirt still blue)
9	other	free-text note required

6. Review queue

Order matters more than anything else for saving time.

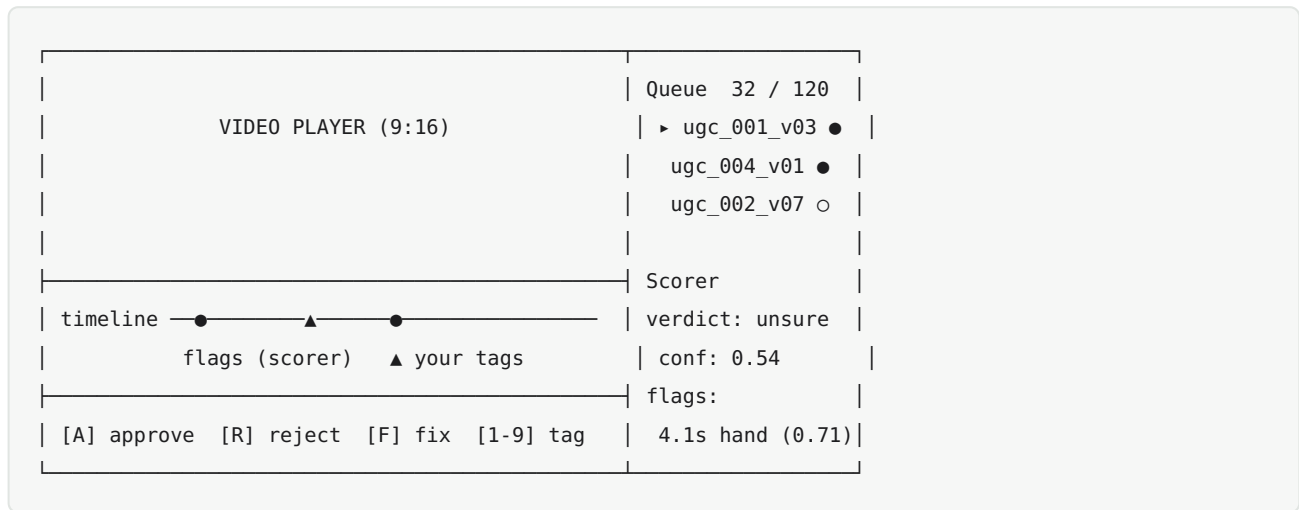
1. **Uncertain first:** variants where scorer confidence is closest to the decision threshold. These are where a human adds the most information
2. **Flagged fails next:** confirm or overturn
3. **Confident passes last:** spot-check a random sample (default 10%) and offer bulk-approve for the rest

The spot-check sample is what lets you estimate how many defects the scorer misses among its "pass" calls, which is the number that decides whether auto-clearing is safe.

7. Review console

Web app, keyboard-first, works on a laptop. Built so a reviewer never touches the mouse.

7.1 Layout



- Original and variant can be toggled side by side (0) to check what changed
- Scorer flags show as markers on the timeline; human tags show in a different color

7.2 Hotkeys

key	action
Space	play / pause
, / .	step one frame back / forward
N	jump to next scorer flag
1 - 9	tag defect type at current timestamp
A	approve (ship)
R	reject (kill)
F	needs fix (repairable)
X	mark scorer flag at playhead as false positive
0	toggle original side by side
J / K	previous / next variant
U	undo last action

7.3 What gets recorded per decision

- Final verdict (approve / reject / fix)
- Each defect tag with type and timestamp
- Which scorer flags were confirmed vs. marked false positive
- Time spent on the variant (start to verdict)

- Reviewer name and timestamp

8. Label store

SQLite, three tables.

```
CREATE TABLE variants (  
  variant_id TEXT PRIMARY KEY,  
  original_id TEXT,  
  change_type TEXT,  
  change_description TEXT,  
  scorer_verdict TEXT,  
  scorer_confidence REAL  
);  
  
CREATE TABLE reviews (  
  review_id INTEGER PRIMARY KEY,  
  variant_id TEXT REFERENCES variants,  
  reviewer TEXT,  
  verdict TEXT,           -- approve | reject | fix  
  seconds_spent REAL,  
  created_at TEXT  
);  
  
CREATE TABLE tags (  
  tag_id INTEGER PRIMARY KEY,  
  review_id INTEGER REFERENCES reviews,  
  source TEXT,           -- human | scorer  
  defect_type TEXT,  
  t_start REAL,  
  t_end REAL,  
  scorer_confidence REAL, -- null for human tags  
  status TEXT           -- confirmed | false_positive | human_added  
);
```

A scorer flag the reviewer confirms becomes `confirmed`; one they dismiss becomes `false_positive`; a defect the scorer never flagged becomes `human_added` (a miss).

9. Eval report

The output that makes the existing scorer better. Generated as a single HTML page after any batch.

9.1 Headline numbers

- Variants reviewed, median seconds per variant
- Scorer agreement with human verdict (%)
- **Misses:** defects humans found that the scorer didn't flag
- **False alarms:** scorer flags humans dismissed

9.2 Confusion matrix (verdict level)

	human: approve	human: reject/fix
scorer: pass	correct pass	missed defect
scorer: fail	false alarm	correct catch

9.3 Per-defect breakdown

defect	precision	recall	top false-alarm cause
hand_deformity	0.xx	0.xx	e.g. fast hand motion blur
flicker	0.xx	0.xx	e.g. real camera cuts

Each row links to 3 example frames: one correct catch, one miss, one false alarm.

9.4 Threshold curve

A chart of "% of batch auto-cleared" vs. "% of defects that slip through," across confidence thresholds. Answers the question that actually matters: **how much review can be skipped while keeping misses under X%?**

9.5 Fix list

Plain-language recommendations generated from the numbers, for whoever maintains the scorer: - "Hand detection misses 40% of defects; most misses are in frames where the hand is partly out of shot" - "Flicker flags fire on real cuts; ignore SSIM spikes within 2 frames of a detected scene change" - "At threshold 0.85, 60% of the batch could be auto-cleared with ~3% misses"

10. Exports (closing the loop)

- `labels.jsonl`: every reviewed variant with verdict and timestamped defects, ready for tuning thresholds or training a classifier
- `fewshot/`: frame crops of confirmed defects per type, for adding as examples to a vision-model prompt
- `thresholds.json`: recommended per-defect thresholds from the curve

Stand-in scorer vision prompt (sketch):

You are reviewing frames from an AI-generated short-form video. Report only clearly visible artifacts from this list: {taxonomy}. Here are confirmed examples of each: {fewshot}. Return JSON: {"flags": [{"type", "frame_index", "confidence", "note"}]}. If unsure, lower the confidence rather than omitting the flag.

The few-shot folder grows with every batch reviewed, which is the loop: **review → labels → better prompt and thresholds → fewer flags to review.**

11. Demo dataset

Built from your own footage only. No client material.

- 5-8 source clips you shoot yourself (talking to camera, holding a product, podcast-style setup)
- 40-80 variants generated with Seedance and Wan Animate: color changes, avatar swaps, product swaps
- Keep the naturally defective outputs and include them deliberately: you want a realistic mix (roughly 30-40% with visible defects)
- **Ground truth:** you label every variant yourself once, independently of the scorer, before running the demo. This is what the eval is measured against

12. Tech stack

piece	choice	why
Backend	Python + FastAPI	same language as ffmpeg/vision tooling
Storage	SQLite	zero setup, one file, easy to ship
Frontend	Vite + React (or plain HTML/JS)	fast to build; keyboard handling is simple either way
Video	ffmpeg, HTML5 <video> with frame stepping	no extra dependencies
Vision	Claude API	stand-in scorer artifact checks
Charts	Chart.js or Recharts	threshold curve, confusion matrix

Runs locally: `python run.py batch/` to score and load, `npm run dev` for the console, `python report.py` for the eval page.

13. Repo layout

```
review-loop/
  backend/
    adapters/
      stdin_scorer.py
      example_adapter.py      # template for plugging in any scorer
    scorer/
      technical.py
      artifacts.py
    queue.py
    store.py
    eval.py
    export.py
    api.py
  console/                    # frontend
  report/                     # eval page template
  batch/                       # demo data (gitignored if large)
  run.py
  README.md
```

14. Build plan (target: Wed Nov 4)

day	work
Wed-Thu	shoot source clips, generate variants, hand-label ground truth
Thu-Fri	stand-in scorer + adapter schema + SQLite store
Sat-Sun	review console: player, timeline flags, hotkeys, logging
Mon	eval report: confusion matrix, per-defect table, threshold curve, fix list
Tue	record demo video, pull numbers for the portfolio
Wed	buffer, send to Kevin first

If time runs short, cut in this order: side-by-side original toggle → threshold curve → few-shot export. Never cut: the console, label logging, and the per-defect breakdown.

15. Next (pitch, don't build)

Source triage (upstream): score source clips before generation for risk factors that cause defects (hands near face, fast motion, occlusion, low light, busy background). Fewer bad variants ever get made.

Auto-repair (downstream): when a defect is confirmed at 4.1s, regenerate only that segment with a new seed or prompt and splice it back. Turns "reject" into "fix" without a full regeneration.

Appendix: optional creative quality layer

If review ever needs to judge whether a variant is a **good video** and not just a clean one, the same console can carry a quick 0-10 rubric:

criterion	weight	10 looks like
Hook (0-2s)	30%	spikes curiosity; an open loop the viewer needs closed
Pattern interrupts	30%	an interrupt (cut, zoom, text pop, angle change) at least every ~3s
Pacing	15%	dead air removed except deliberate pauses before a reveal
Visual clarity and setting fit	15%	product clearly visible; setting matches the brand's buyer
Captions	5%	simple, readable; single-word for fast content, multi-word for informational
CTA	5%	clear and matches the hook's promise

Kept out of the core demo so the pitch stays focused on QA.