Monitait Vision Engine — User Manual

Version: 3.22.5 Last Updated: 2026-06-11 Application URL: `http://localhost` (port 80, or whichever port your install uses)

> Note on screenshots: The images embedded in this manual were captured against MVE v3.21.18. They predate the Process tab filter bar (3.21.24), ColorE/Area/E controls (3.22.2–3.22.5), the (i) info tooltips, and the reordered card layout (3.22.4). The text reflects 3.22.5. Refreshed screenshots will land in a follow-up release.

Table of Contents

---

1. Introduction

What is Monitait Vision Engine (MVE)?

MVE is an industrial-grade quality-control platform that uses computer vision (YOLO + a math channels module) to inspect products in real time, save annotated frames + per-frame detections to a database, and trigger ejection of bad units.

It runs as a web application — operators control it from a browser. No client install needed.

Headline features as of 3.22.5

- Detection pipelines — chain a YOLO model with the Math channels module (FFT, blob, gradient, color, etc.); per-phase enable + per-phase stride.
- Per-class controls — for every class the model detects, the operator gets a card with: Show / Narrate / Beep / Store / ColorE / Area toggles, Min conf / Severity inputs, and read-only analytics (confidence p5/p50/p95, absolute CIELAB color, scalar E, bbox area).
- Ejection procedures — rule-based product rejection with conditions on count, area, color ΔE (relative to a reference), and absolute E (color magnitude, no baseline, 3.22.5).
- DB-backed configuration — `mve_config_kv` table in TimescaleDB is the source of truth; `.env.prepared_query_data` is kept as a downstream snapshot for backward compatibility (3.22.0).
- AI Assistant — a chat interface against your line's recent state; OpenAI-compatible (3.21.23 added custom `base_url` + `model_id` so any compatible endpoint works — Anthropic-via-proxy, local Ollama, custom enterprise inferences).
- AI Trainer — one-click upload of a defect frame to the central training service from the defect drawer (3.21.22).
- Multi-camera — every metric has a per-camera breakdown so you can compare cam 1 vs cam 2 vs … without juggling pages.
- Shipment quality scoring — a server-rendered PDF report per shipment, configurable cron, threshold-graded RELEASE / RE-INSPECT / HOLD.

Audience

Two main personas:

- Operator (on the floor) — uses Dashboard, Gallery, defect drawer; rarely touches Advanced.
- Inspector / Engineer — configures cameras, models, ejection procedures, per-class thresholds via Cameras / Inference / Process / Advanced.

The manual is written for both. Operator-only sections are tagged [operator] . Engineer-only sections are tagged [engineer] .

---

2. System Requirements

Recommended hardware (per inspection line)

- CPU: 8+ cores (12+ if running 2+ cameras at 1 fps)
- RAM: 16 GB minimum, 32 GB if also running the math module
- GPU: NVIDIA with CUDA support (8 GB VRAM or more for `yolov5s`)
- Disk: SSD, 500 GB minimum (raw frames + annotated images add up fast — see §15 for retention)
- Network: 1 Gbps LAN to the inspection station, internet optional (for AI Assistant + AI Trainer)
- Encoder/ejector hardware: Watcher Jet is the Monitait-supported, open-source watcher device that handles encoder pulse counting, ejector relay control, and serial I/O over USB — it's the recommended option. Generic Arduino/PLC over USB serial is also supported as a fallback if you already have one wired up. Both are optional but strongly recommended for closed-loop ejection.

Software

- Ubuntu 22.04 LTS or 24.04 LTS (some sites also run Windows + Docker Desktop)
- Docker 24.x + Docker Compose v2
- NVIDIA driver matching the CUDA version baked into the YOLO image
- For Windows hosts: WSL2 backend, GPU pass-through enabled

Supported browsers

Chromium-based browsers (Chrome / Edge / Brave / Opera) and recent Firefox. The UI uses Server-Sent Events for the live dashboard — that works in all modern browsers.

---

3. Installation & First Boot

One-time setup

1. SSH into the host. Clone the repo:

   ```
   git clone https://github.com/SMART-FALCON-AI/monitait_vision_engine.git
   cd monitait_vision_engine
   ```

2. Pull the MVE image (proprietary models — YOLO and math — get loaded separately):

   ```
   docker pull registry.monitait.com/monitait/mve:3.22.5
   ```

3. Load the YOLO and Math images from the disk transfer you received from Monitait support:

   ```
   docker load < yolo_inference.tar
   docker load < math_inference.tar
   ```

4. Bring everything up:

   ```
   docker compose up -d
   ```

5. Open `http://localhost` in your browser.

On first boot, MVE migrates the legacy `.env.prepared_query_data` file (if present) into the `mve_config_kv` Postgres table (see §15). It logs `3.22 migration: copied N top-level key(s) into mve_config_kv` — that's a one-time event.

Configuring cameras

After first boot, MVE has no cameras configured. Go to the Cameras tab to add them (§11).

Default endpoints

- Web UI: `http://localhost`
- DB: `monitait_timescaledb:5432` (Postgres + Timescale extension)
- Redis: `monitait_redis:6379`
- YOLO inference: `http://yolo_inference:4442`
- Math inference: `http://math_inference:4443`

---

4. Quick Start — 5-Minute First Run

1. Open the UI. Browser → `http://localhost`. You land on the Dashboard.
2. Add a camera. Go to Cameras tab → Add Camera → pick USB or RTSP → save. The card appears.
3. Enable ROI on the card and adjust `roi_xmax / roi_ymax` to roughly match the area you want inspected. (This matters — see §11; many "no detection" complaints come from ROI mismatch.)
4. Define and activate a capture state. Go to Cameras → Capture States → add a state → add at least one phase to it (which cameras to fire, light mode, delay between phases, steps per cycle). Save it. Then activate it (set it as current state). Capture is event-driven: the watcher loop runs whichever state is current, firing the phases in order. There is no separate "Start capture" button — capture begins as soon as a state is active and MVE is running. Stop by switching the current state to a state with no enabled phases, or by stopping the MVE container.
5. Watch detections. The dashboard timeline composite shows recent frames per camera. If your YOLO model is loaded, you should see bounding boxes immediately.
6. Tune per-class behavior. Go to Process tab. For each class you care about, tick Show (draw bbox), set Min conf above the noise floor (see §13), and optionally Store to write detections to DB.
7. Optional: add ejection rules. Go to Advanced → Procedures → Add Procedure to route specific detection patterns to your ejector hardware. See §17.

You're now inspecting.

---

5. The Tab Layout

Top of the UI: a row of tabs.

Tab	Audience	What's there
[clipboard] Dashboard	[operator]	Live timeline composite, counters, capture/inference status.
[chart] Charts	[engineer]	Time-series analytics. Click any dot → see the frame + nearby detections.
[AI] AI Assistant	both	Chat with an OpenAI-compatible model against your line's recent state.
[gallery] Gallery	[operator]	Browse stored annotated frames. Click for a drawer with the raw image + detections + AI Trainer upload.
[hardware] Hardware	[engineer]	Encoder / ejector / serial port wiring + tests.
[camera] Cameras	[engineer]	Per-camera config — source, ROI, exposure, gain, white balance.
[inference] Inference	[engineer]	Model registry + pipeline editor.
[gear] Process	[engineer]	Per-class card grid: Show / Narrate / Beep / Store / ColorE / Area + analytics.
[tools] Advanced	[engineer]	Procedures, timeline config, raw `mve_config_kv` JSON editor, system controls.

Top-right corner: language picker, save-all-configuration button, current shipment label.

---

6. Dashboard

The operator's main view.

Left panel — live metrics

A column of mini-cards showing:

- Encoder — current absolute encoder count from the ejector hardware (if wired).
- Speed — derived linear speed (units depend on encoder pulses-per-meter calibration).
- Pulses/s, Movement — raw encoder pulses per second + whether movement is detected.
- Ej Queue / Ej Active / Ej Enable / Ej Offset / Ej OK / Ej NG — ejection queue depth, active flag, software enable toggle, configured frame offset between detection and ejection, and lifetime OK/NG counters.
- OK / NG — current shipment OK and NG counts (Phase 2 of the quality system, see §17).
- Downtime, Analog, Power — secondary metrics (lower priority).
- Capture: <state> — name of the active capture state.
- Inference: <pipeline> — name of the active pipeline.
- Latency / Inf FPS / Cap FPS — three numbers: average inference latency in ms, inference frames per second, capture frames per second.
- OK pill — green when inference is keeping up with capture, amber when the inference queue is backing up.

Center panel — timeline composite

A live image showing the most recent N frames per camera, stitched into a grid (cameras as rows, frames as columns left → right oldest → newest).

If a frame had detections, bounding boxes are drawn directly on the thumbnail.

If you see no bounding boxes but you know detections are happening:

- Check the Process tab — has Show been ticked for the relevant class? (The 3.21.25 strict opt-in rule means classes without `show=True` don't draw.)

- Check that the class has an entry in `audio_settings` — auto-register fires on first detection per class, but classes detected before the upgrade may need re-touching.

If the entire timeline is empty:

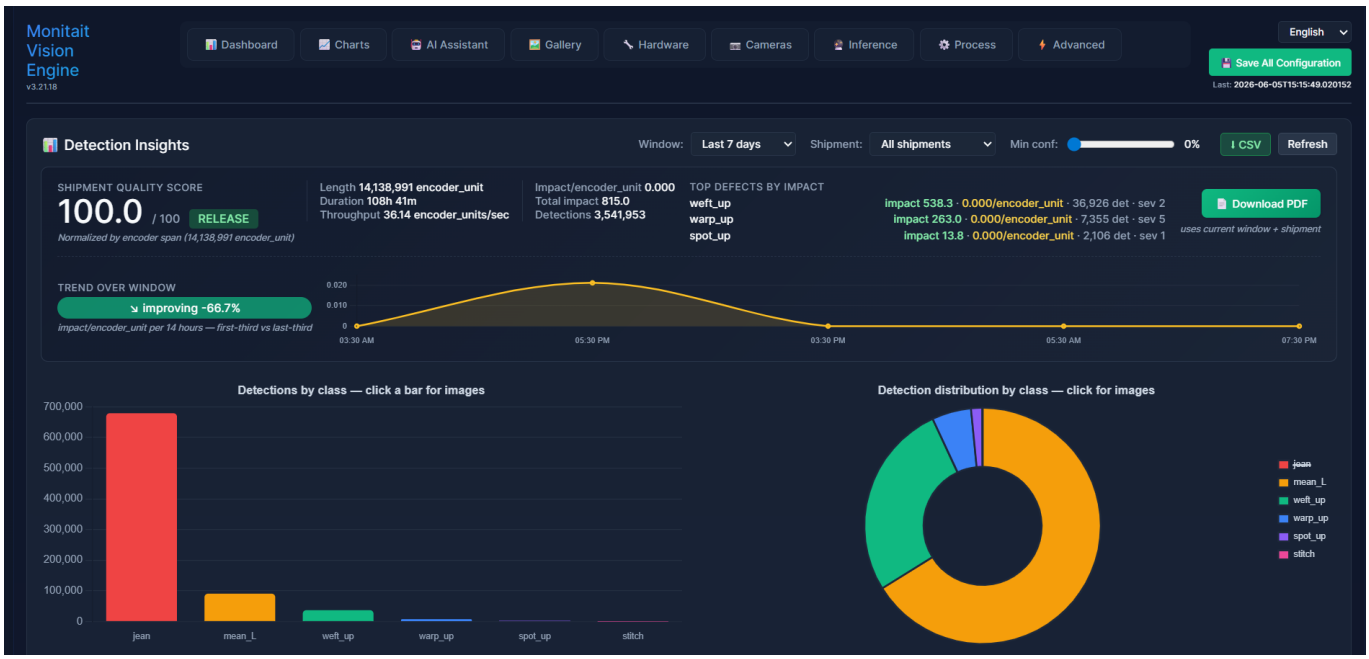
- Inference → check that the active pipeline has at least one enabled phase, the model is reachable.
- Cameras → check that the active capture state lists the right cameras under its phases.

Bottom-right — system stats

CPU, RAM, Disk. Watch the Disk % during a long shift — see §18 for cleanup options.

---

7. Charts



Charts are time-series widgets driven by `inference_results` rows.

Available cards

- Detection counts by class over time.
- Inference latency and capture FPS trends.
- Per-class confidence (live distribution).
- Shipment quality score card (3.21.13) + trend chart (3.21.16).

Click a dot to see the frame

Every dot on the time series is clickable. Clicking opens the same detail drawer as the Gallery — raw frame, detections list, AI Trainer upload button.

This is the fastest way to ask "what triggered that spike?".

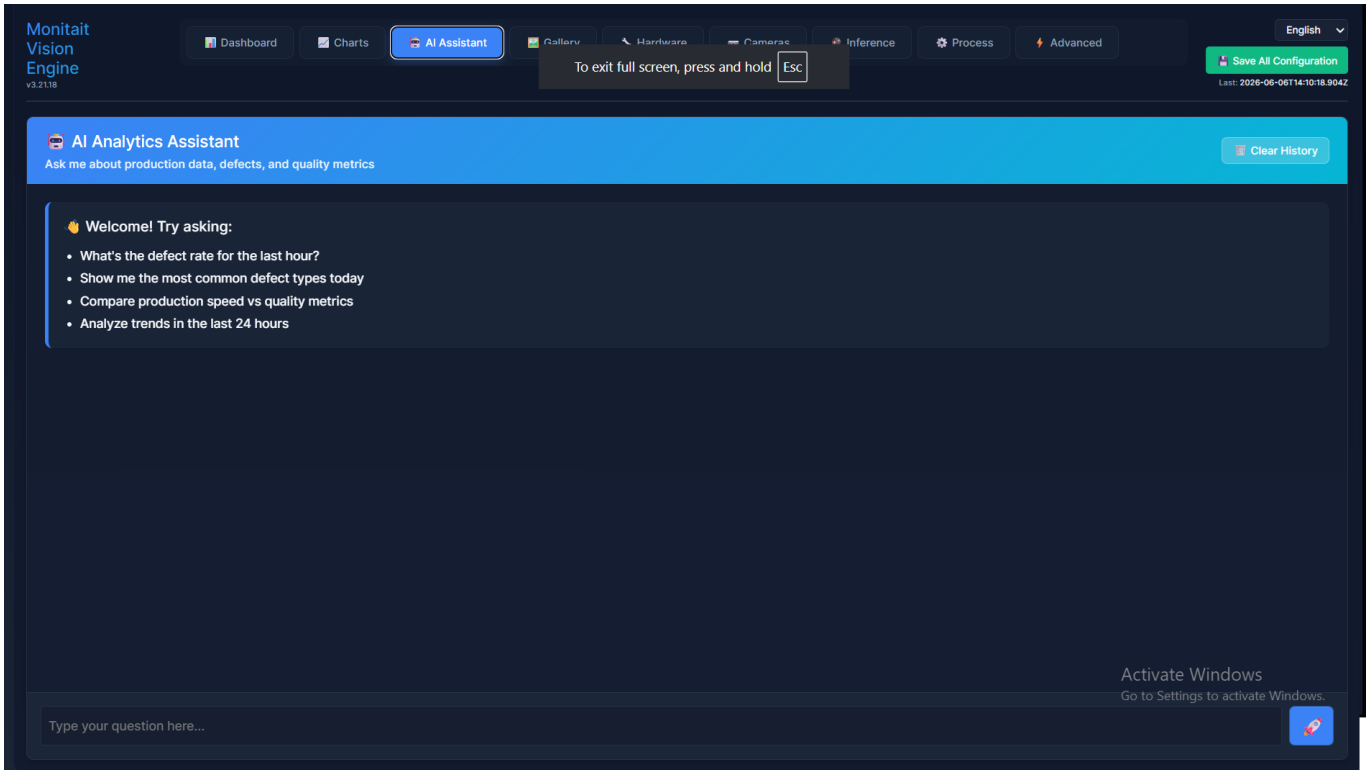
Time-window controls

Top of each chart: a window selector (last 5 min / 1 h / 6 h / 24 h / 7 d). Wider windows aggregate.

---

8. AI Assistant

A chat interface backed by an OpenAI-compatible model.



Setting up the model

1. Open the AI Assistant tab.
2. Click AI Configuration (gear icon at the top of the chat).
3. Fill in:
   - Name — any friendly label (e.g. `kimi-k2.6`, `gpt-4o`, `claude-haiku`).
   - Provider — `chatgpt` (works for any OpenAI-compatible endpoint), `claude`, `gemini`, `local`.
   - API key — your key.
   - Base URL (optional) — for non-OpenAI endpoints. E.g. `http://host.docker.internal:11434/v1` for a local Ollama. Leave empty to use the provider's default.
   - Model ID (optional) — overrides the default model name. E.g. `kimi-k2.6`, `gpt-4o`, `llama3:8b`.
4. Save. The new model becomes active.

This was extended in 3.21.23 to let you bring any OpenAI-compatible inference (Anthropic via proxy, local Ollama, custom enterprise endpoints) without code changes.

Built-in context

The Assistant has a pre-loaded tool that hits read-only `/api/*` endpoints (cameras, inference, system metrics, timeline counts, latest detections). Ask it questions like:

- "What's the current latency?"
- "How many TB defects in the last hour?"
- "Which camera produced the most detections today?"
- "Summarize the last shipment's quality."

---

9. Gallery & Defects

A browsable grid of stored annotated frames.

Tile = one detection event. Click any tile → drawer with:

- The raw image (no bbox)
- The annotated image (with bboxes)
- A table of detections (name, confidence, bbox coords, LAB color if ColorE was on, area if Area was on)
- Buttons:

- [upload] Upload to AI Trainer (3.21.22) — posts the raw image to `https://ai-trainer.monitait.com/{task_id}/upload` for human re-labeling. The `{task_id}` is configured under Advanced → AI Trainer Integration.

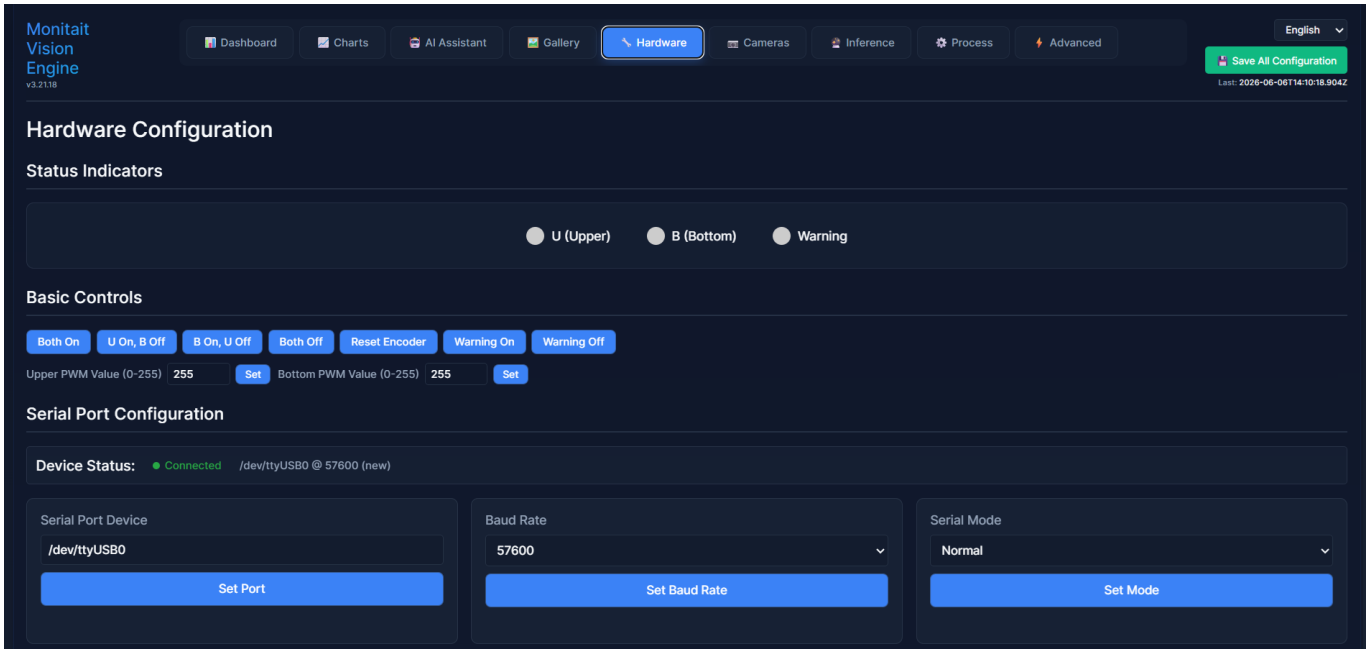
- [search] Open in Charts — pivots to the Charts tab around that timestamp.

Use filters at the top to narrow by class, camera, shipment, time window.

---

10. Hardware

Wiring and tests for the encoder + ejector + serial peripherals.



Recommended device — Watcher Jet

The Monitait-supported watcher device is Watcher Jet — open-source firmware + reference hardware design that handles all three jobs in one board: encoder pulse counting, ejector relay control, and serial I/O over USB. It exposes the standard wire protocol MVE expects on `/dev/ttyUSB0`, so out of the box you get encoder counts, movement detection, and ejector test/fire commands working with zero glue code.

If you don't have a Watcher Jet on site, any Arduino or PLC speaking the same serial protocol works — see the firmware in the Watcher Jet repo for the wire format.

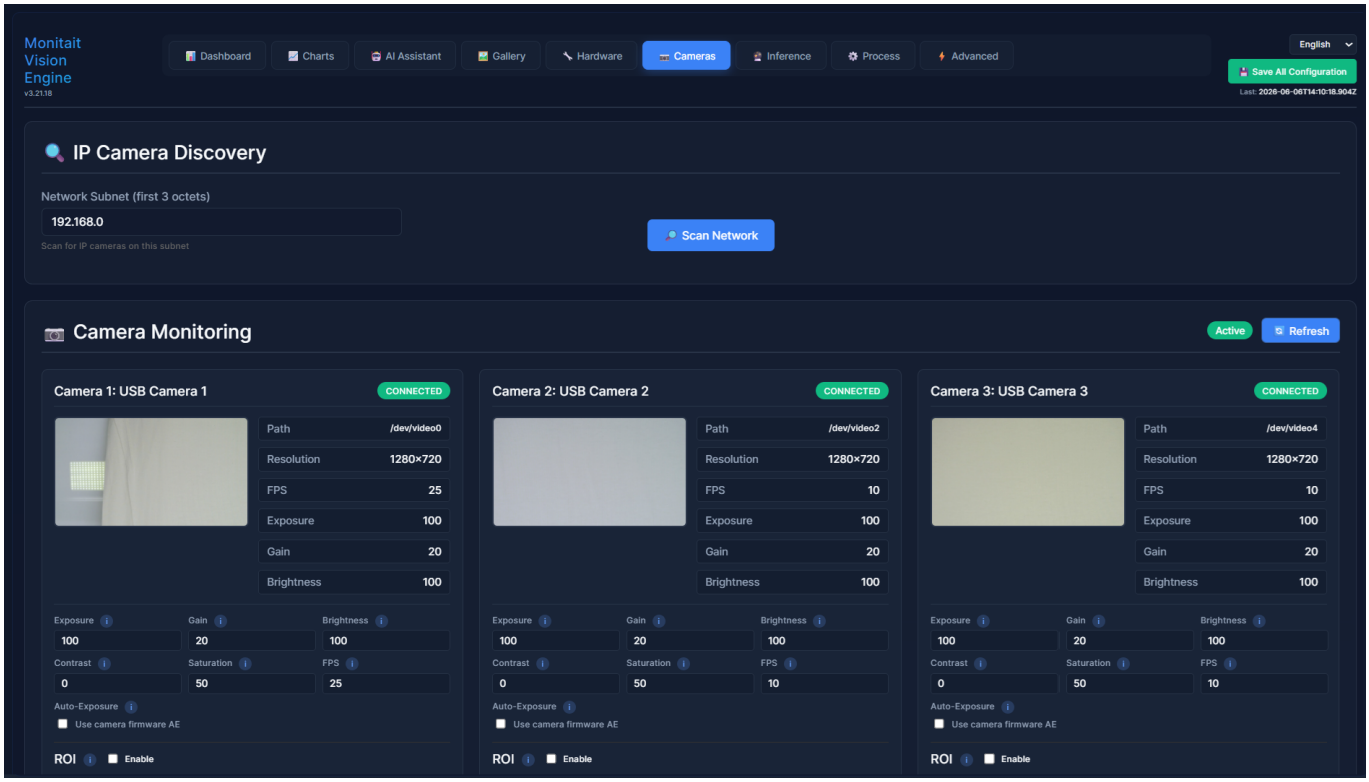
Configuration

- Serial port — pick the Watcher Jet / Arduino / PLC device path (`/dev/ttyUSB0`, etc.).
- Encoder — pulses-per-meter calibration, software enable toggle.
- Ejector — frame offset (how many frames between detection and physical ejection), enable, test buttons.
- Test ejection — fires the ejector once. Use this to confirm wiring before a shift.

If your line has no ejector, you can run with the ejector software-disabled; everything else still works.

---

11. Cameras



Per-camera config card. The fields below explain themselves but a few are worth highlighting:

ROI (Region of Interest)

Most-common gotcha — if a camera was producing zero bboxes after install, the cause was almost always ROI:

- `roi_enabled` must be ticked.
- `roi_xmax / ymax` must match (or be reasonably close to) the resolution the YOLO model was trained on. A standard YOLO weights file expects ~640×640 or ~900×600; a 1280×720 full-sensor frame is at a different scale and the model returns `[]`.

If you flip the box and detections start firing, that confirms the issue.

Exposure / Gain / Brightness / Contrast / Saturation

Standard V4L2 / DirectShow controls. Most cameras like `auto_exposure=false` with a fixed value for inspection — auto exposure causes brightness drift across a shift.

Source

- USB: `/dev/video0`, `/dev/video2`, …
- IP: `rtsp://user:pass@ip/stream`

Discover / Rescan

Top-right of the tab — auto-detects new USB cameras.

Capture States

Below the camera cards, the Capture States panel is where the capture loop's behavior is defined. A state is a named sequence of phases. The watcher executes the active state continuously, one phase at a time, looping back to phase 0 when the last phase finishes.

Each phase has:

- `cams` — list of camera ids that fire on this phase (e.g. `[1, 2]`).
- `light_mode` — light pattern to switch to before capture (`U_ON_B_OFF` / `U_OFF_B_ON` / etc., depending on your wiring).
- `delay` — seconds to wait after switching lights, before grabbing frames. Lets the light settle and the camera's auto-anything stabilise. Typical: `0.1`–`0.5`.
- `steps` — how many times this phase fires per cycle. `1` is normal; `-1` means "loop forever on this phase" (used in the built-in `infinite` state).
- `analog_threshold` — optional sensor gate; only fire when an analog input reads above this value. `-1` disables.

A few states usually ship pre-configured:

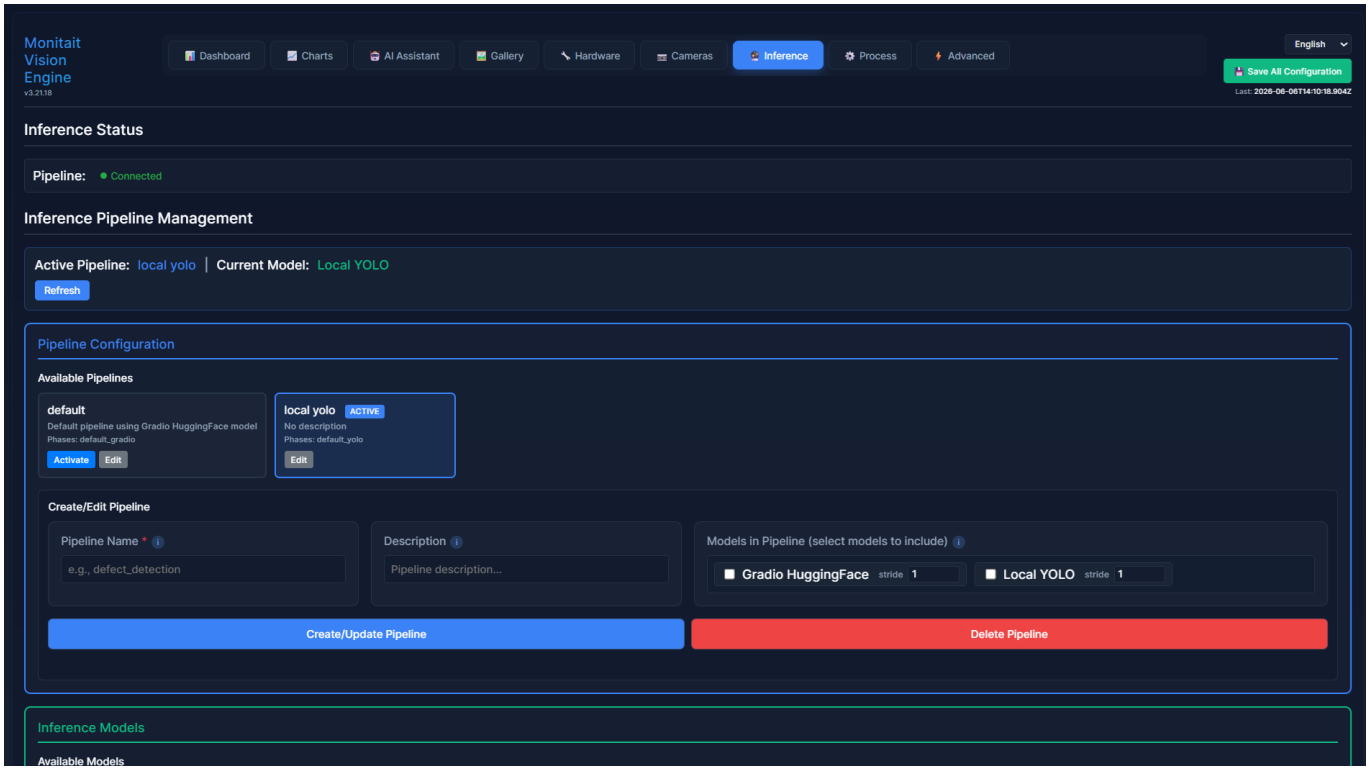
State	Typical use
`default`	Single phase, both cameras, with a small delay. Used during normal inspection.
`infinite`	Single phase, `steps=-1`, runs forever. Useful for free-running cameras with no encoder gating.
`infinite-max`	Same as `infinite` but `delay=0` — maximum throughput.
`KC-Back` (site-specific)	Custom state for back-cam inspection at the KC factory.

To activate a state, set it as Current State. The watcher picks up the change without a restart. There is no separate "Start capture" button — capture is the watcher executing the current state. To pause capture, switch the current state to one whose phases are disabled (or whose `cams` list is empty), or stop the MVE container.

> Editing live states — if you stop and reconfigure a running state, remember that MVE rewrites the config file on shutdown from in-memory state. Stop the container, edit the state, then start it (or use the UI to edit while running — the UI write goes through the API which is safe).

---

12. Inference (Models + Pipelines)



Models

Top section. Each registered model has:

- Name (label)
- Model type — `yolo`, `math`, `gradio`, `legacy`
- Inference URL — HTTP endpoint inside the docker network
- Model name — what to pass through to the inference container
- Confidence threshold — server-side floor (rarely useful; the per-class Min conf in Process tab is the real control)

Pipelines

Bottom section. A pipeline is a list of phases; each phase has:

- `model_id` (which registered model)
- `enabled` (true/false)
- `order` (phase order; lower = first)
- `stride` (run every Nth frame; useful for math-only phases)

The active pipeline runs on every captured frame. The phases run in `order`, gated by `stride`.

[!] Math stride pitfall (workaround: odd stride)

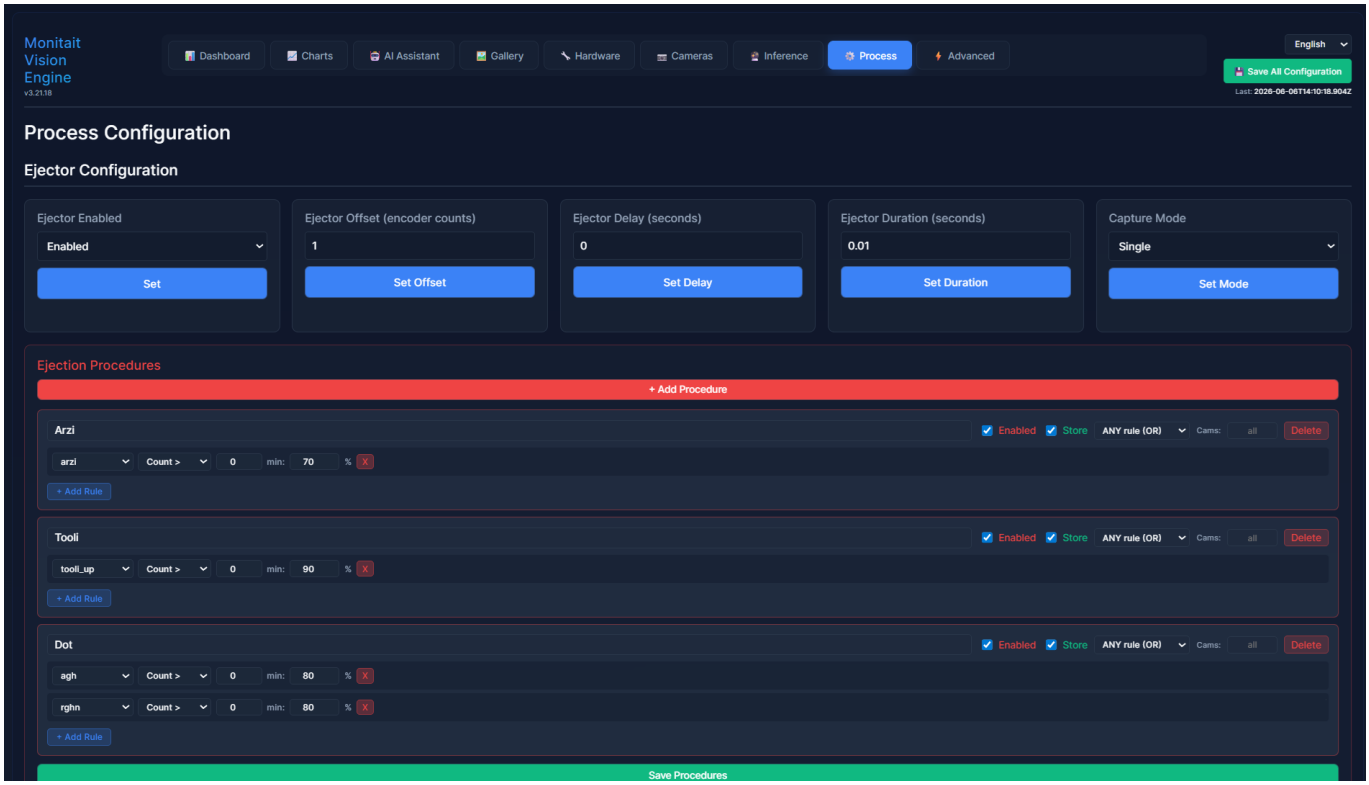
With a global frame counter and N cameras alternating, an even stride means math only ever fires on one specific camera. Use an odd stride (`9`, `11`, etc.) so math hits each camera in rotation. (A per-camera counter

refactor is queued for a future release.)

---

13. Process Tab — Per-Class Cards & Analytics

[engineer] — The single biggest tab. Where operators tune per-class behavior.



Filter bar (3.21.24)

Top of the grid. Three controls:

- [search] Search box — live filter by class-name substring.
- [x] Show only active (last 1 h) — hide classes that haven't fired in the window.
- Window selector — `15m / 1h / 6h / 24h / 7d`. Drives both the toggle's filter and the busy-first sort.
- Counter chip — `Showing X of Y · Z active in last 1h`.

Even with the toggle off, active classes float to the top of the grid (busy-first sort), so the ones you care about are always at the top.

Per-class card layout (3.22.4)

Top to bottom on every card:

1. Header — class name + (i) info tooltip.
2. Toggles row (6 checkboxes, wraps to 2 lines on narrow cards):

- [x] Show — draw bounding box on annotated frames. Strict opt-in (3.21.25): only `show=true` draws. Missing entries don't draw.
- [x] Narrate — TTS speak the class name when detected.
- [x] Beep — play the beep sound (configurable below) when detected.
- [x] Store — persist this class's detections to `inference_results`. Off = silently dropped. Default off for new classes.
- [x] ColorE (3.22.2) — extract CIELAB color from this class's bbox on every detection. Drives the [color] color row + [E] E row.
- [x] Area (3.22.3) — store bbox area on the detection (`det.area`) so it shows up in the [area] area row and is available to ejection procedures.

3 Min conf + Severity inputs:

- Min conf % — per-class confidence floor. Detections below this never reach Store / Narrate / Beep / ejection.
- Severity (0–100) — weight used in the shipment quality score. 0 = cosmetic, 100 = critical.

4 [chart] normal conf (3.21.12, extended 3.21.24):

- Overall: `p5 · p50 · p95 % (p5–p50–p95) · n=…`
- Per-camera: `cam 1: … · cam 2: …`
- (i) tooltip: YOLO confidence percentiles over last 7 days. Tune Min conf above p5 (filters noise) but well below p50 (keeps real detections).

5 [color] color (3.22.2, refactored 3.22.4) — only visible when ColorE is on AND there are ≥5 LAB samples:

- Overall: `L: p5·p50·p95 a: p5·p50·p95 b: p5·p50·p95 (p5–p50–p95) n=…`
- Per-camera below.
- L\ lightness 0–100 · a\ green↔red · b* blue↔yellow.
- (i) tooltip: the full formula and drift-diagnosis tips (L moving = exposure / fade, a moving = green/red dye, b moving = blue/yellow / lighting color temperature).

6 [E] E (abs) (3.22.5) — single-scalar color magnitude when ColorE is on:

- Formula: $E = \sqrt{(L^2 + a^2 + b^2)}$.
- Stored on the detection (`det.E`) at extraction time so ejection procedures can use it directly.
- Display: `p5 · p50 · p95 (p5–p50–p95)`.
- (i) tooltip: the formula plus three usable ejection conditions (`E_greater`, `E_less`, `E_between`) — see §17.

7 [area] area (3.22.3) — only visible when Area is on:

- Overall: `p5 · p50 · p95 px² (p5–p50–p95) · n=…`
- Per-camera below.
- Formatted as `k` / `M` for readability (e.g. `12.4k` instead of `12,450`).
- (i) tooltip: spotting false positives (way below p5) and merged objects / yarn issues (way above p95).

8 Beep sound — dropdown (sine / square / sawtooth / triangle) + test button.

Strict show=true (3.21.25)

Before 3.21.25, the draw rule was inconsistent across the codebase, leading to either silent yolo classes (3.21.22) or math floods (3.21.23). The current rule is uniform:

> DRAW IFF `audio_settings[name].show == True`.

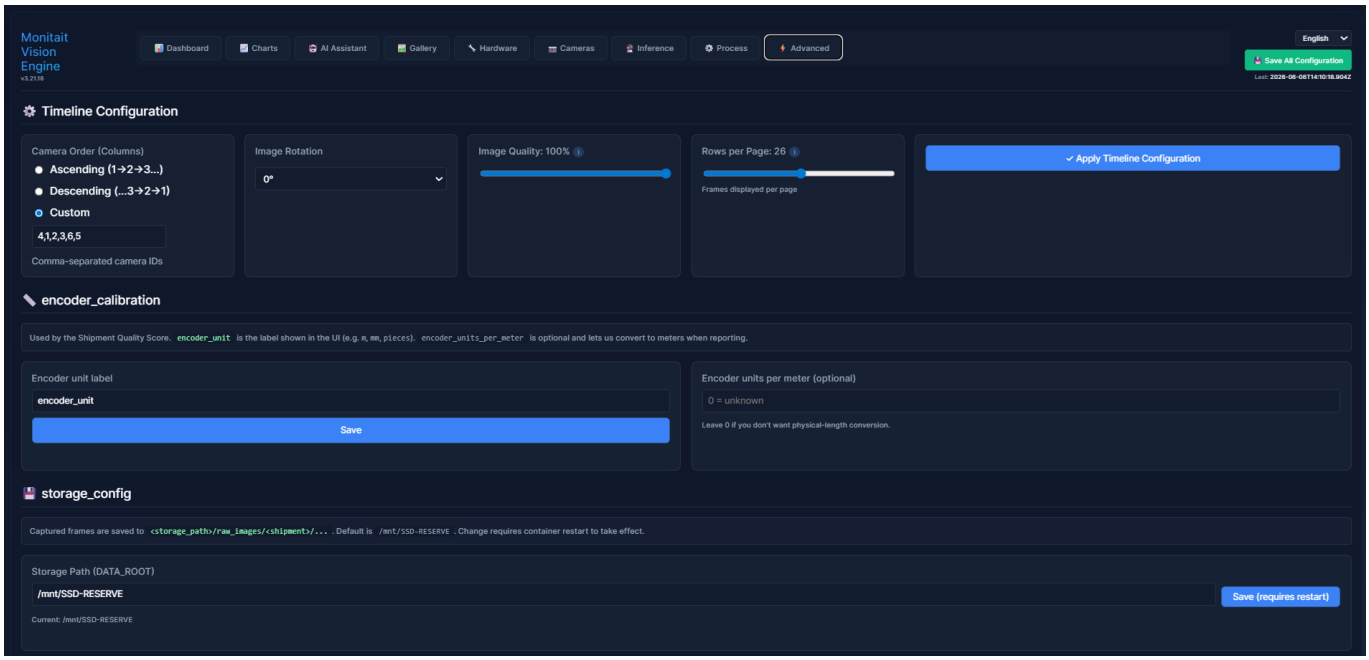
Anything else (False, missing, malformed) → skip. Combined with auto-register (3.21.25), every detected class gets a `show=false` stub the first time it's seen, so it appears in the Process tab as an unticked card. Operators opt in explicitly. No silent classes, no surprise draws.

Backed by a single source of truth: `services/draw_filters.py` (3.21.26) loads `audio_settings` itself with a 5 s cache, and every caller just asks `should_draw_class(name)` — no parameters.

---

14. Advanced — Procedures, Timeline, Data File

[engineer] — engineer-only.



Procedures (ejection rules)

A procedure = a named ruleset that triggers ejection when its rules fire on a frame. Each procedure has:

- Name
- Logic — `all` (every rule must trigger) or `any` (any one)
- Rules — see §17 for the full list of conditions

Timeline Configuration

- Image quality (JPEG quality, 0-100)
- Number of rows (frames per page in the composite)

- Image rotation (per-camera output rotation applied at composite time)
- Show bounding boxes (display toggle for the live composite)
- Camera order (`normal`, `reverse`, or `custom` with a comma-separated cam-id list)
- Procedures — the list referenced above
- Object filters — legacy per-class control; preserved for back-compat but auto-migrated into `audio_settings` on first run of 3.21.22+. The Process tab is now the single place to manage it.

Data File Editor

A JSON editor for `service_config`. As of 3.22.0, the source of truth is the `mve_config_kv` Postgres table; the JSON editor reads from DB and writes through to DB + the `.env.prepared_query_data` file snapshot. See §15.

System

- Restart MVE
- Save All Configuration (forces a write-through)
- Export Config / Import Config — round-trips the entire JSON; works across sites.

---

15. Configuration Storage (DB + File)

[engineer] — engineer-only.

What changed in 3.22.0

Before 3.22.0, the file `.env.prepared_query_data` (a single ~50 KB JSON) was the source of truth. Every config change rewrote the whole file. Concurrent writers could clobber each other. There was no audit trail.

3.22.0 moves the truth into a Postgres table:

```
mve_config_kv (
    key         TEXT PRIMARY KEY,         -- "service_config", "timeline_config", ...
    value       JSONB NOT NULL,
    updated_at  TIMESTAMPTZ DEFAULT NOW(),
    updated_by  TEXT                       -- "mve" / "migration:3.22.0" / "operator:..."
)
```

Read path

1. Try Postgres (`mve_config_kv`)
2. If unreachable or table empty, fall back to `.env.prepared_query_data` on disk

Write path

Dual-write, always:

1. Persist to Postgres
2. Dump file snapshot — so legacy tooling (`cat | jq`, the Advanced Data File Editor, pre-3.22 builds at other sites) keeps working

One-time migration

On MVE startup, `services/migrations.py::migrate_data_file_into_db()` runs. If the DB table is empty AND the file has content, it ingests the file. Idempotent; safe on every reboot.

`/api/config/db_status`

Quick health check:

```
curl http://localhost/api/config/db_status
# {"backend":"db","db_reachable":true,"key_count":2,"file_exists":true}
```

`backend: "db"` is the post-migration state. `backend: "file"` means MVE is in the fallback path (DB unreachable or empty).

Why it matters

- Atomic per-key writes
- Audit trail via `updated_at` + `updated_by` columns (the seed for an audit-log feature)
- Cross-site config push becomes possible (a central console writes to multiple sites' DBs)
- The "MVE owns the file and clobbers your edits" race goes away

---

16. AI Trainer Integration

[engineer] to set up, [operator] to use.

Configuration

1. Advanced → AI Trainer Integration.
2. Set the task id — the trainer-side identifier the uploads land under.
3. (Optional) override the base URL if your trainer isn't `ai-trainer.monitait.com`.

Usage (operator)

1. Open a defect drawer (Gallery → click a tile, or Charts → click a dot).
2. Click [upload] Upload to AI Trainer.

3 The raw image POSTs to `${base_url}/${task_id}/upload`. No bbox annotations are sent — the trainer's job is to re-label.

Why this exists

Closes the loop. False positives + true positives + missed defects all funnel back to a labelled dataset that drives the next model retraining.

---

17. Ejection Procedures Reference

[engineer]

Rule structure

A rule has:

- `object` — class name to evaluate
- `condition` — see below
- `min_confidence` — gate (per-class min applied first, then this one)
- (varies per condition) — `count`, `area`, `max_delta_e`, `E`, etc.

Available conditions

Count-based

Condition	Triggers when
`count_equals`	Detection count == `count`
`count_greater`	Detection count > `count`
`count_less`	Detection count < `count`

Area-based

Condition	Triggers when
`area_greater`	Best-conf detection's bbox area > `area`
`area_less`	Best-conf detection's bbox area < `area`
`area_equals`	Best-conf detection's bbox area == `area`

Color (relative — needs a reference)

Condition	Triggers when	Fields
`color_delta`	CIE76 ΔE between current LAB and reference > threshold	`max_delta_e`, `reference_mode` (`fixed` / `previous` / `running_avg`)

Color (absolute — single threshold, 3.22.5)

Condition	Triggers when	Fields
`E_greater`	`det.E > E`	`E`
`E_less`	`det.E < E`	`E`
`E_between`	`E_min ≤ det.E ≤ E_max`	`E_min`, `E_max`

Where `E = √(L² + a² + b²)`, stored on the detection at extraction time when ColorE is on for the class. Reference-free — set a fixed threshold and it holds across shifts, dye batches, operator changes.

Tip: look at the [E] E (abs) badge on the class's Process tab card. Set `E_less` ≈ p5 and `E_greater` ≈ p95 — that single procedure catches both ends of color drift.

Logic — `all` vs `any`

A procedure with multiple rules can require all to trigger (typical) or any (catch-anything-bad rule, fewer false positives).

Example procedure (color drift on TB)

```
{
  "name": "TB color drift",
  "logic": "any",
  "rules": [
    { "object": "TB", "condition": "E_less",    "E": 42, "min_confidence": 30 },
    { "object": "TB", "condition": "E_greater", "E": 64, "min_confidence": 30 }
  ]
}
```

This ejects TB detections that fall outside the typical color band — without any reference color setup or running average.

---

18. Troubleshooting

Symptom → cause table

Symptom	Likely cause
Dashboard timeline composite empty	No active pipeline, or active capture state has no enabled phases. Check Inference + Cameras.

Symptom	Likely cause
Timeline composite has frames but no bboxes	The relevant classes don't have `show=true`. Open Process tab, find the class, tick Show.
One camera produces no detections, the other does	Almost always ROI. Open Cameras → that camera → tick `roi_enabled` and check `roi_xmax/ymax` match your model's training resolution.
Math channels never fire on one specific camera	Math phase stride is even (2 cameras + even stride → only cam 2 hits the math counter). Set math stride to 9 (or any odd value).
AI Assistant returns 401 / "model not found"	API key wrong, base URL missing (for non-OpenAI providers), or model_id not recognized. Try with explicit `base_url` + `model_id`.
`/api/config/db_status` returns `backend: "file"`	Postgres is unreachable. Check `docker compose ps`. MVE keeps running on the file fallback.
Disk filling fast	Either raw frames or annotated frames. Use Advanced → Storage to archive old shipments. Also see §19 best practices.
503 from the shipment quality PDF endpoint	`reportlab` missing in the container. `docker exec monitait_vision_engine pip install reportlab`. Rides into the next image bake.
Browser shows old config after a UI change	Static-file cache. Hard-refresh (Ctrl+Shift+R). MVE doesn't need a restart for JS/CSS changes.
MVE crashes on boot with "image bind-mount mismatch"	Your bind-mounted code is older than the image expects. Re-pull the image OR update `./vision_engine` to a matching version.

Logs

- `docker logs monitait_vision_engine` — main app logs
- `docker logs monitait_timescaledb` — DB logs
- `docker logs yolo_inference` / `math_inference` — model containers
- `docker logs --since 5m monitait_vision_engine | grep ERROR` — recent errors only

---

19. Best Practices

Per-class hygiene

- Don't enable Store on every class. Math channels can flood `inference_results`. Be selective.
- Use Min conf above the p5 baseline (visible on the card) — that's the noise floor.
- Use Severity to tier classes: 100 for "ship-killer" defects, 20 for "log but don't reject".

Color tracking

- Turn on ColorE for the 2–3 classes whose color matters (typically the product class itself). Don't turn it on for everything — LAB extraction is per-bbox `cv2.cvtColor` and adds up.
- Use absolute E ([E] row) for ejection thresholds. It's baseline-free and stable across shifts.
- Use the L / a / b breakdown for diagnosis when E shifts — figure out which channel moved.

Disk

- Raw frames cost the most. Decide whether you actually need them or just the annotated ones.
- Set a retention policy on `inference_results` via TimescaleDB chunk-drop (consult Monitait support).
- Archive old shipments before disk hits 80%.

Backups

`mve_config_kv` is in Postgres — your existing pg_dump cron covers it. The `.env.prepared_query_data` snapshot is also written after every save; copy it to a network share nightly as a belt-and-suspenders.

Multi-site

If you run more than one MVE instance, the Export Config + Import Config buttons in Advanced let you take a JSON snapshot from one site and apply it at another. The format is stable across 3.21.x and 3.22.x.

---

20. FAQ

Q. Why does the same class draw on one camera but not the other? A. Two possible reasons:

1. The `show=true` rule was honored only when frames came through the per-camera draw path; the timeline composite path historically had a separate draw rule. Fixed in 3.21.26 — both paths now go through the single-source `services/draw_filters.py`. If you see this on 3.21.x lower than 3.21.26, upgrade.
2. ROI on one camera but not the other. Check Cameras tab.

Q. What's `det.E` vs `det.lab_color`? A. `lab_color` is the `[L*, a*, b*]` array (three numbers). `E` is the scalar `√(L² + a² + b²)` — convenient single number for ejection thresholds. Both are written at detection time when ColorE is on for the class.

Q. Can I roll back to 3.21.x? A. Yes. The DB migration is one-way (3.22.0 reads file → DB) but doesn't delete the file — `.env.prepared_query_data` is still updated on every save. A pre-3.22 build reads the file and works. New audit metadata is just ignored by older code.

Q. Why does cam 1 never get math detections? A. The math phase has a stride. With 2 cameras and an even stride, the global frame counter only hits the math-firing slot on cam 2 (the watcher captures cam 1 → cam 2 → cam 1 → cam 2…). Set math stride to 9 (or any odd value) — math then alternates between cameras. A per-camera-counter refactor is queued for a future release that removes this fragility.

Q. Can I bring my own AI model for the Assistant? A. Yes, any OpenAI-compatible endpoint. Set `base_url` and `model_id` in the AI Configuration. Examples:

- Local Ollama: `base_url=http://host.docker.internal:11434/v1`, `model_id=llama3:8b`
- Anthropic via OpenAI-compatible proxy: `base_url=https://your-proxy/v1`, `model_id=claude-sonnet-4`
- Custom enterprise: `base_url=https://your-internal-endpoint/v1`, `model_id=your-model`

Q. How do I figure out the right Min conf? A. Open the per-class card. The [chart] normal conf badge shows `p5 · p50 · p95`. Set Min conf slightly above p5 to filter noise. If you also see false positives in the Gallery despite that, raise it toward p50.

Q. How do I figure out the right E_greater / E_less for a color rule? A. Open the per-class card with ColorE on. The [E] E (abs) badge shows `p5 · p50 · p95`. Set `E_less ≈ p5` and `E_greater ≈ p95`. Anything outside that band is historically unusual color.

Q. Where do I report bugs? A. <https://github.com/SMART-FALCON-AI/monitait_vision_engine/issues> or your Monitait support contact.

End of manual.