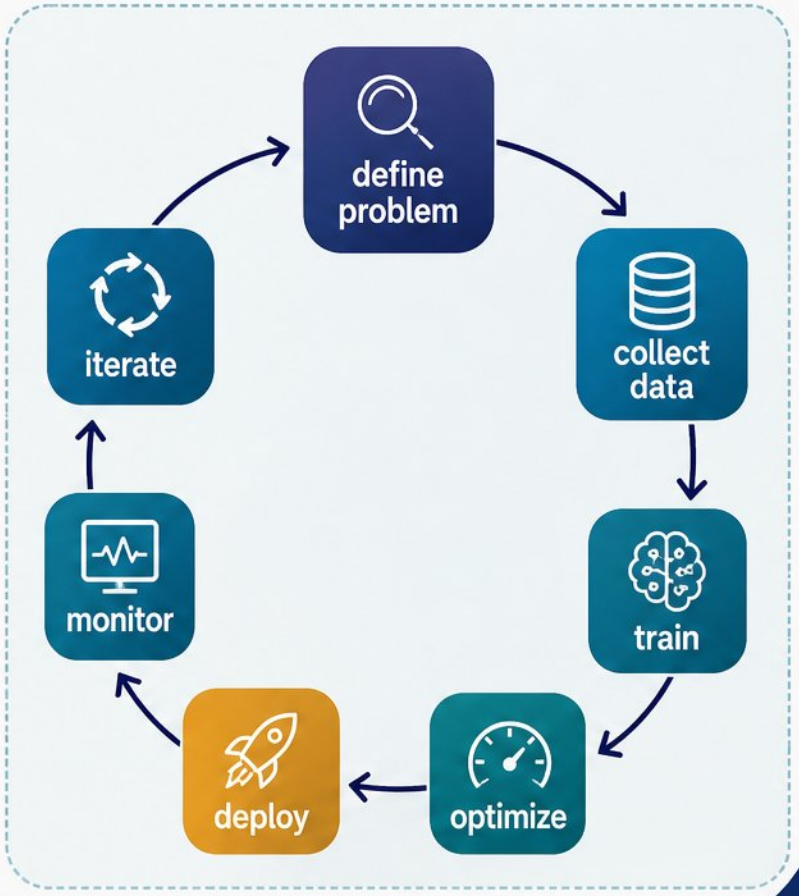
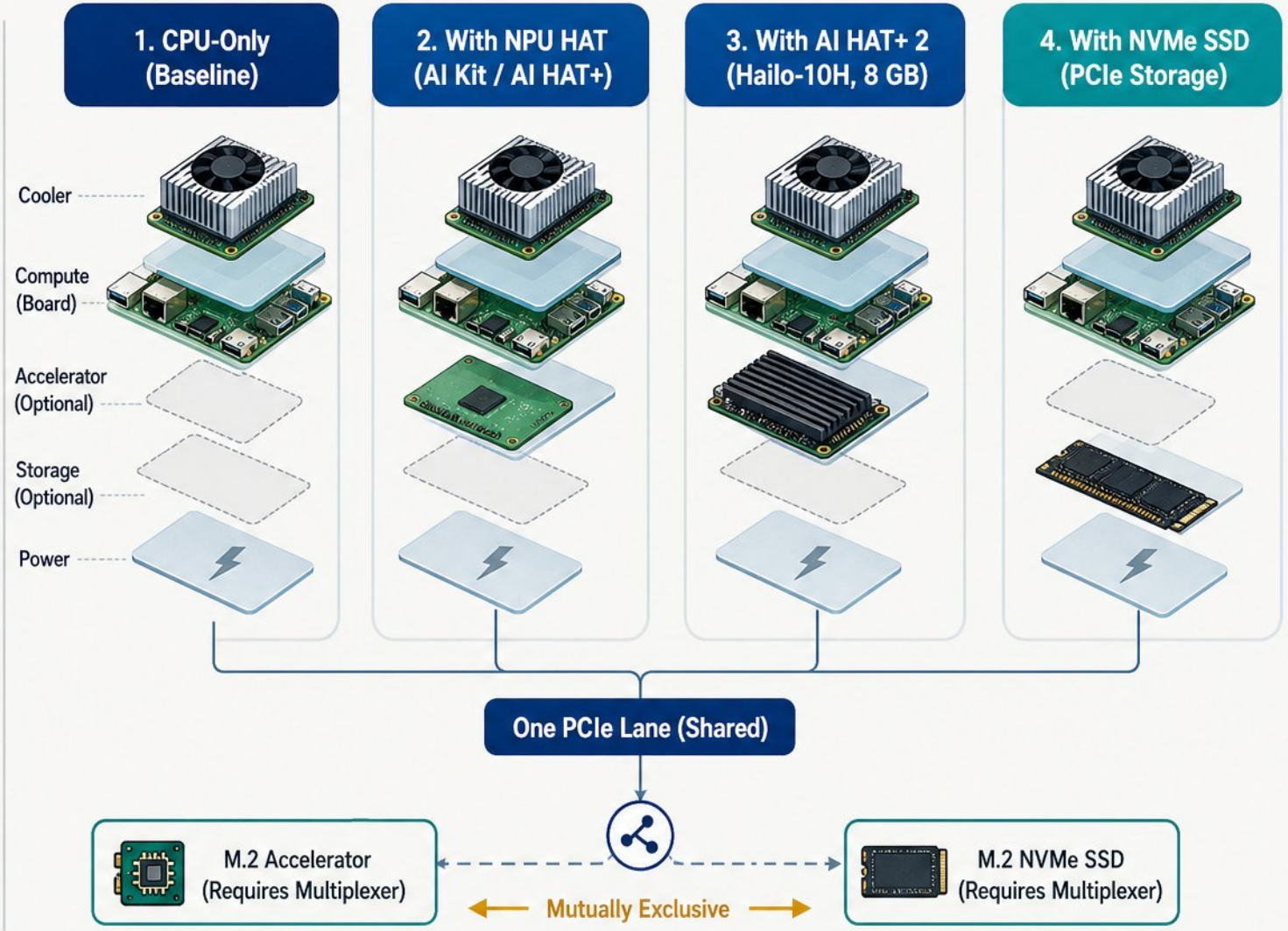
How to Use Raspberry Pi for Machine Learning: A Practical Workflow

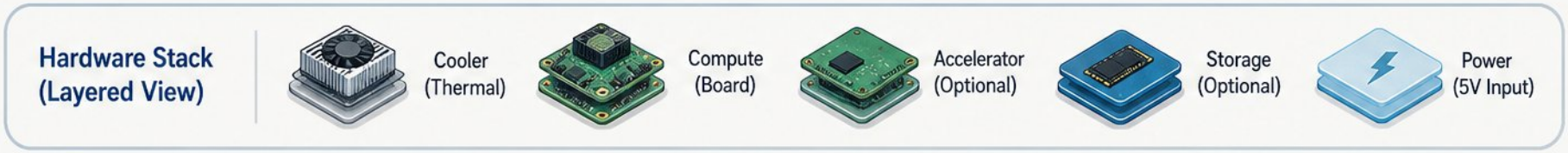


- **Goal:** take an edge-ML prototype from problem definition to deployed, monitored device
- **Workflow loop:** define problem → collect data → train → optimize → deploy → monitor → iterate
- **Who it's for:** makers, educators, students, developers with basic Python and Linux skills
- **Build targets:** real-time image classifier, keyword spotter, sensor anomaly detector
- **Reality check:** edge performance is limited by memory bandwidth and thermals, not TOPS

Choosing the Right Raspberry Pi Board and Accelerator

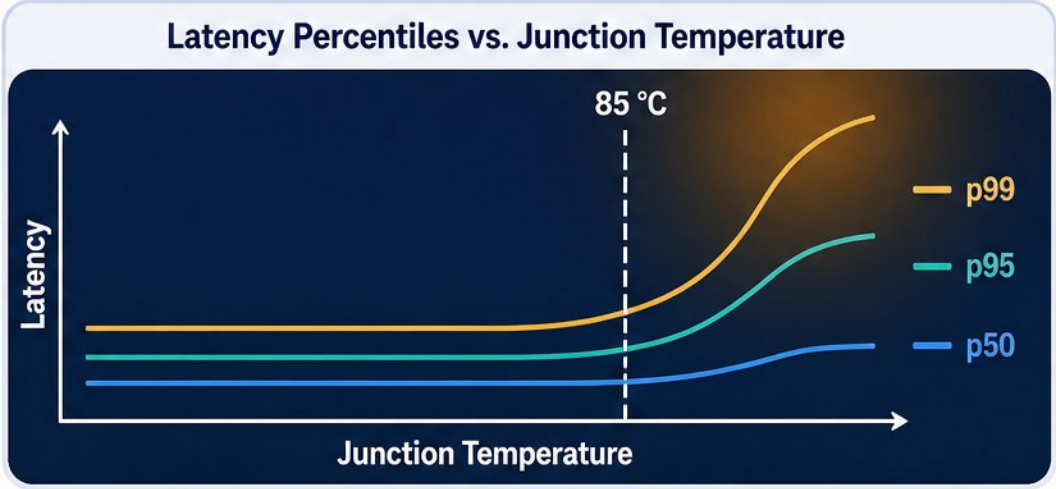
- Pi 5 baseline: quad-core Cortex-A76 to 2.4 GHz, higher memory bandwidth, one PCIe lane
- Pi 5 CPU runs MobileNet-class vision at 20–40 ms, enough for 20–30 FPS
- AI Kit / AI HAT+ (Hailo-8L, 13–26 TOPS) pushes YOLOv8 to 30–200 FPS
- AI HAT+ 2 (Hailo-10H, 40 TOPS, 8 GB) targets LLMs/VLMs; vision similar, tokens often slower than CPU
- PCIe slot is shared: NVMe SSD and M.2 accelerator need a multiplexer
- Budget tiers: 4 GB for TFLite; 8–16 GB for multi-model; accelerator only at a real wall

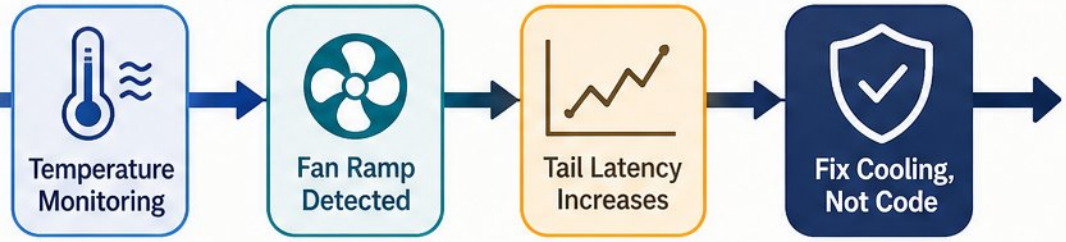




Cooling, Power, and Storage for Sustained Inference

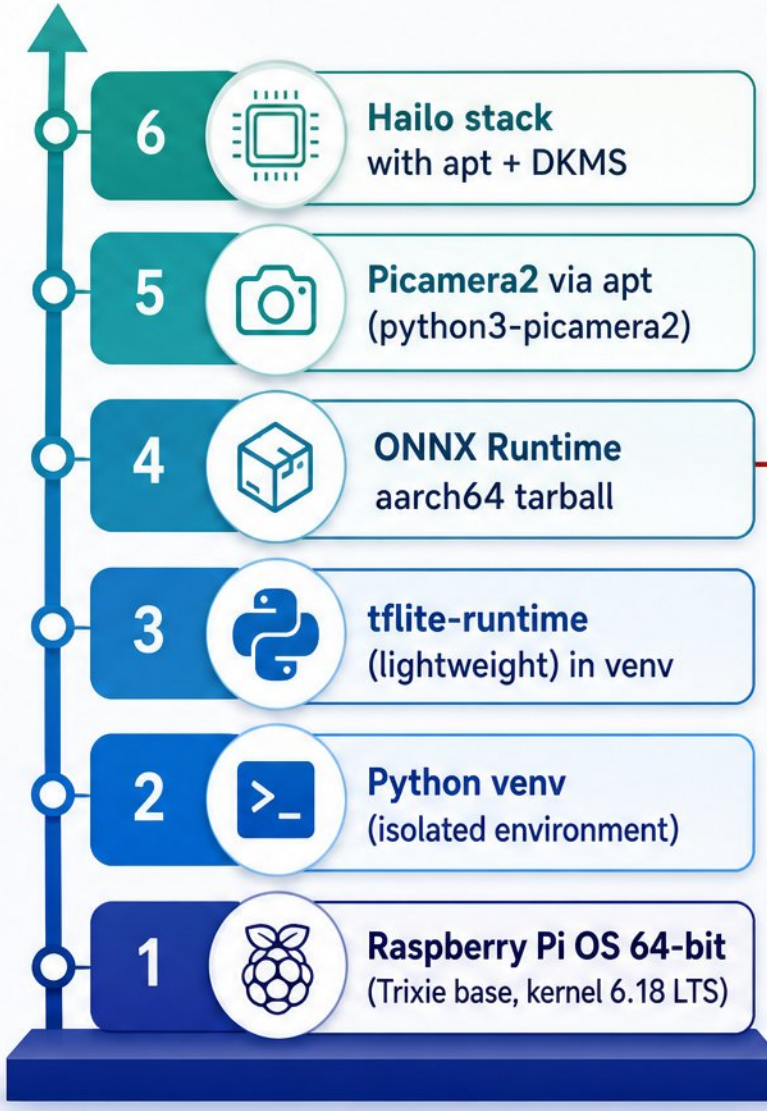
- Active cooling is effectively mandatory: uncooled Pi 5 throttles near 85 °C in ~4 minutes
- Official Active Cooler holds sustained load around 60–65 °C at full clock speed
- Report p50, p95, and p99 latency — throttling shows up in the tail
- Use the official 27 W USB-C supply; under-voltage causes camera drops and stalls
- Boot OS and models from NVMe or high-endurance microSD for fast loads and no dropped frames
- Log temperature alongside latency: a spike with fan ramp means fix cooling, not code





Setting Up Raspberry Pi OS and the ML Runtime Stack

- Use Raspberry Pi OS 64-bit (Trixie base, kernel 6.18 LTS); 32-bit blocks NEON/INT8 paths
- Preconfigure hostname, user, Wi-Fi, and SSH with Raspberry Pi Imager for headless boot
- Create a Python venv; install lightweight tflite-runtime, not full TensorFlow (500 MB+)
- Install ONNX Runtime from the aarch64 release tarball; armhf 32-bit is unsupported
- Get Picamera2 via apt (python3-picamera2), not pip; install Hailo stack with apt + DKMS
- Run a one-image smoke test before writing app code



Capturing Images, Audio, and Sensor Data on the Device

- Picamera2 is the native CSI capture path; install via apt, not pip
- Match capture resolution to model input, e.g. 224×224 RGB888
- High-rate DNG: copy buffers in a worker thread, large buffer_count
- Main thread must never do disk I/O; record raw with the base Encoder
- Audio: USB mic plus ALSA/PipeWire, save fixed-length 1-second clips
- Sensors: fixed-rate sampling with a sliding window matching model input
- One folder per class, consistent filenames, or a manifest CSV
- Keep datasets small; train heavier models on workstation or Colab

Selecting and Training an Edge-Friendly Model

- Start with MobileNetV2/V3 or EfficientNet-Lite for classification
- Detection: SSD MobileNet or YOLOv8-nano; anomaly work: small LSTM or autoencoder
- Train on a workstation or Colab, then export — not on the Pi
- Transfer learning on a few hundred to a few thousand labeled samples usually wins
- Fix your budget up front: latency target, model size cap, accuracy floor
- Prefer models with solid TFLite or ONNX Runtime op coverage
- Export paths: Keras → TFLite; PyTorch → ONNX (opset 17+) or via onnx2tf

MODEL CHOICE MATRIX

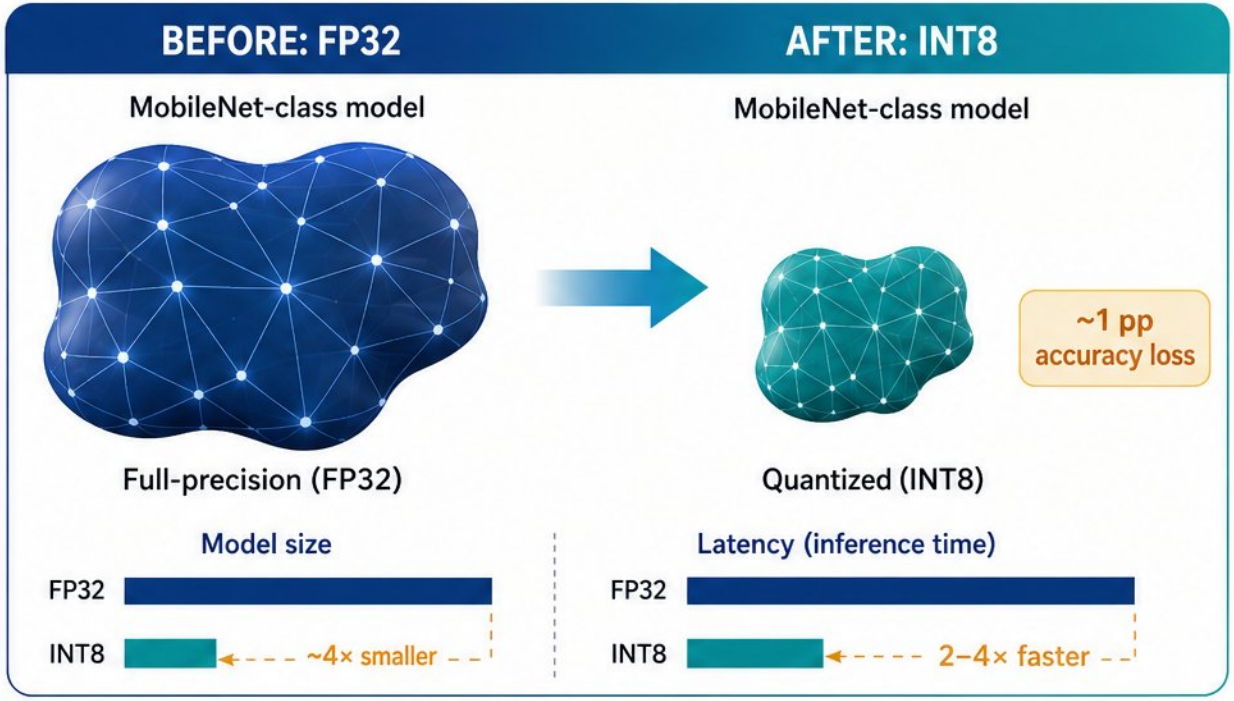
	CLASSIFICATION	DETECTION	TIME-SERIES ANOMALY
LATENCY TARGET	✓	✓	✓
MODEL SIZE CAP	✓	✓	✓
ACCURACY FLOOR	✓	✓	✓

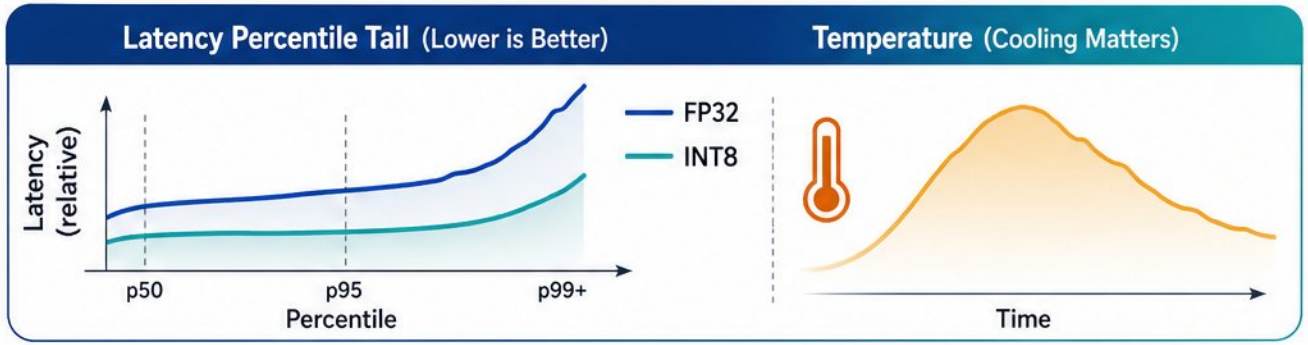
TFL **TFLITE**
Export Path
Keras → TFLite

ONNX **ONNX**
Export Path
PyTorch → ONNX (opset 17+)
or via onnx2tf

Optimization: Quantization, Compilation, and Benchmarking

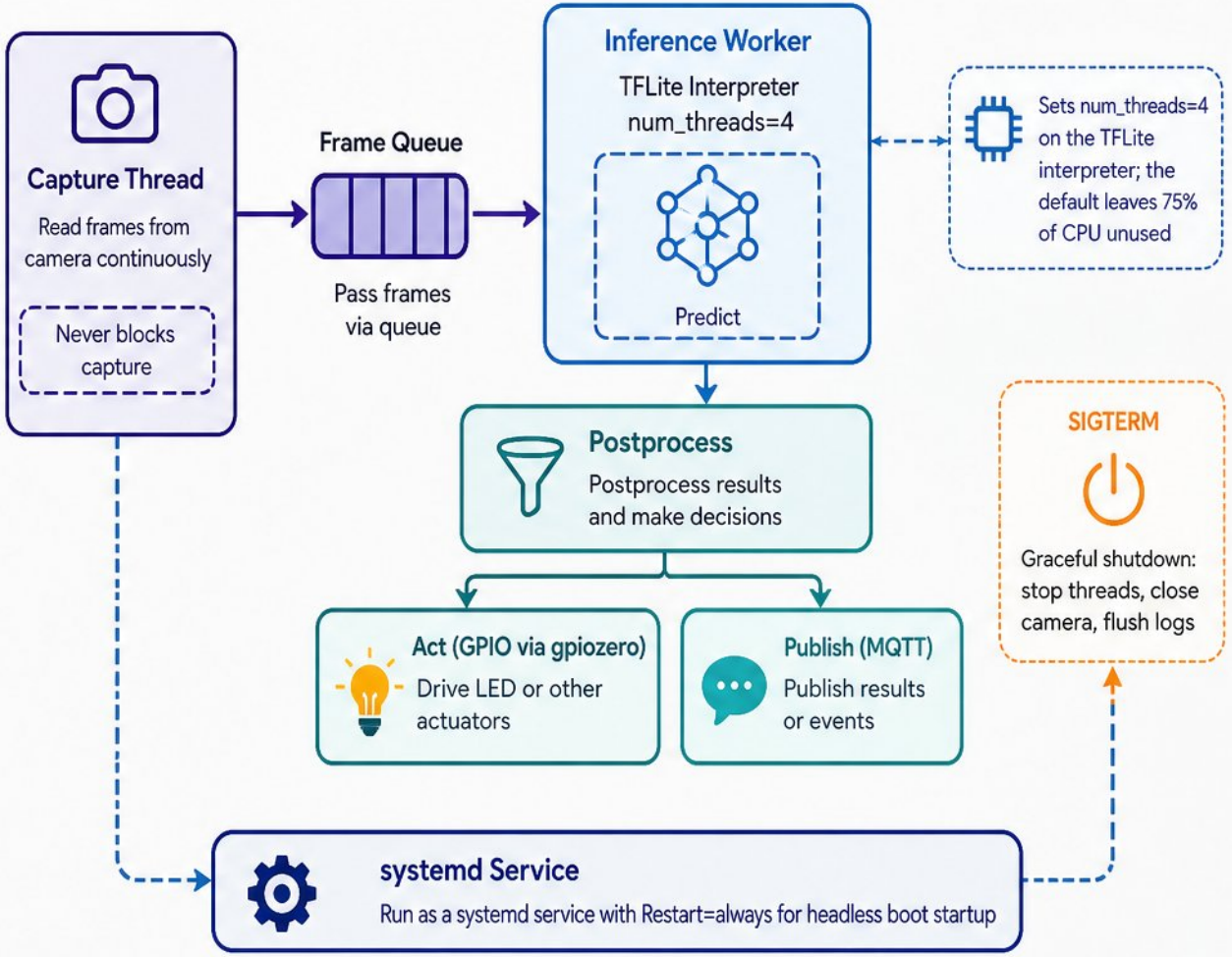
- INT8 post-training quantization: model ~4× smaller, 2–4× faster, ~1 pp accuracy loss
- Full integer needs a representative dataset (few hundred to 1,000+ in-domain samples)
- Always measure: FP32 can beat INT8 on very small models (Q/DQ overhead)
- ONNX Runtime: export format (QDQ vs QOperator) and graph opt level swing latency ~40×
- Hailo .hef compile only on x86_64 Linux, not the Pi; target --hw-arch hailo8l
- Benchmark properly: warm up, pin threads, report p50/p95, control cooling





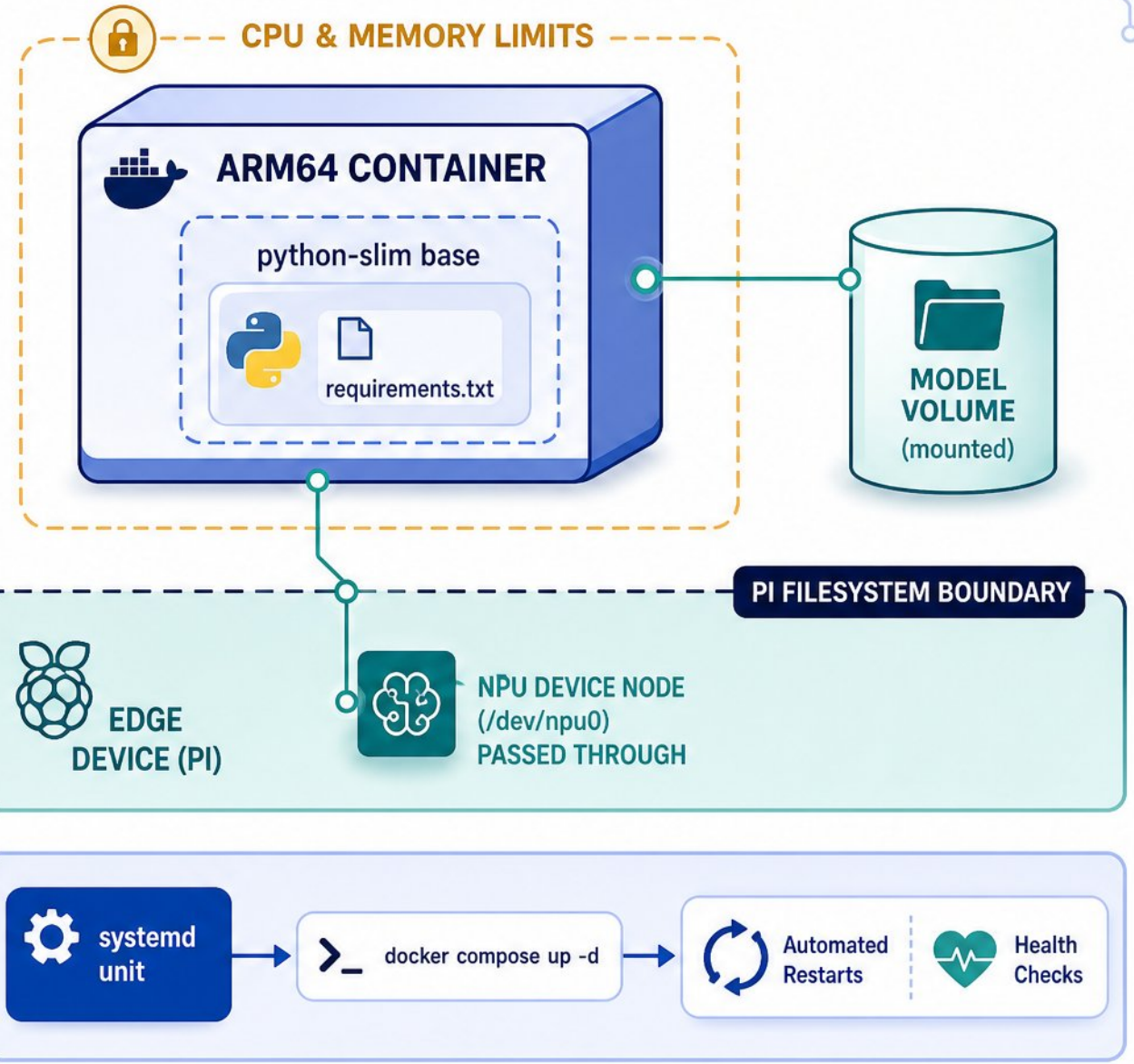
Deploying the Inference Application: Loops, Threading, and GPIO

- Pipeline shape: capture → preprocess → predict → postprocess → act, timing each stage
- Set num_threads=4 on the TFLite interpreter; the default leaves 75% of CPU unused
- Keep camera reads in their own thread; pass frames via queue so nothing blocks capture
- Run inference every Nth frame when slower than capture instead of dropping frames
- Move actuators with gpiozero; log timestamps, handle missing camera and SIGTERM
- Run as a systemd service with Restart=always for headless boot startup



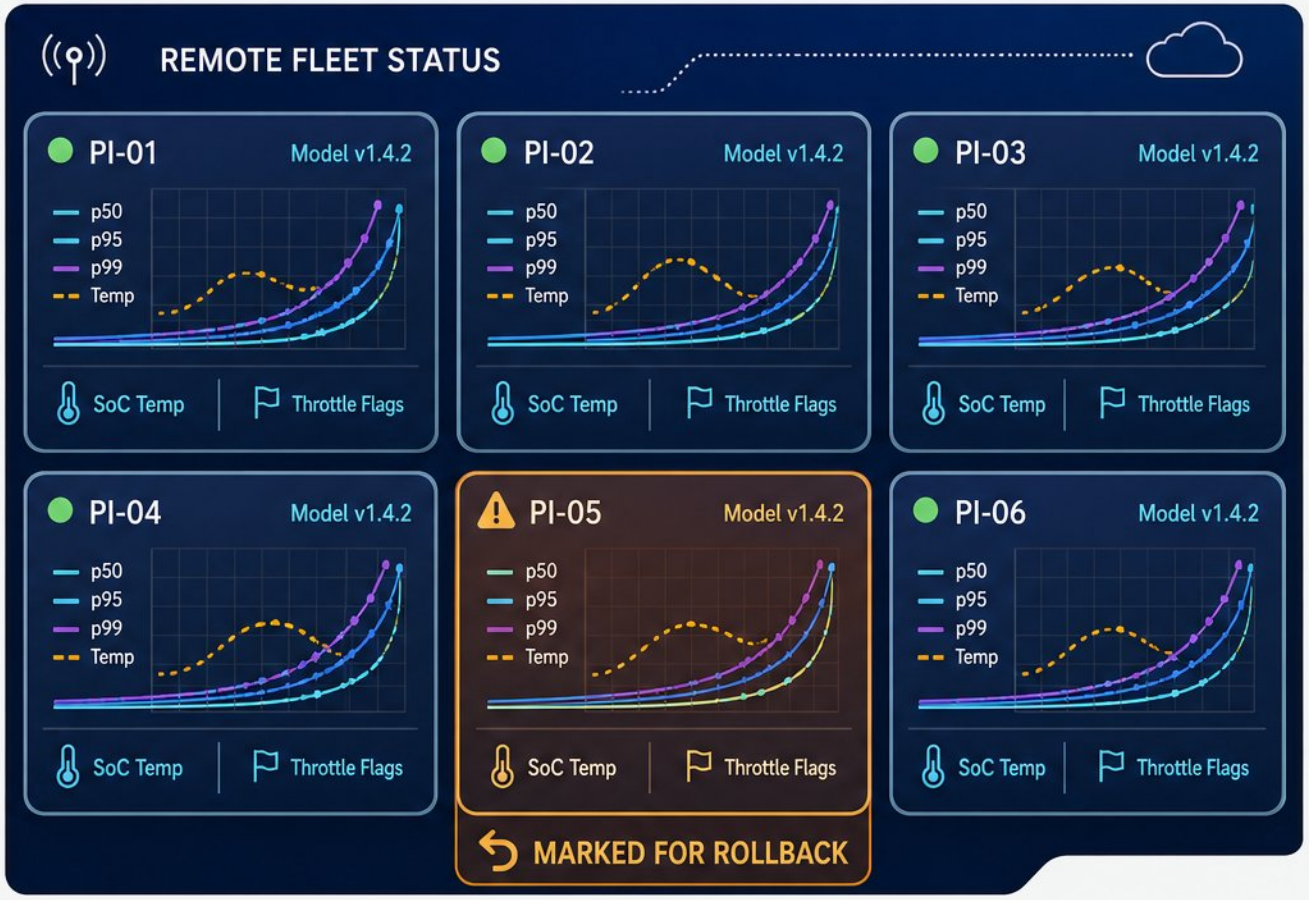
Containerizing and Service-Managing the Edge App

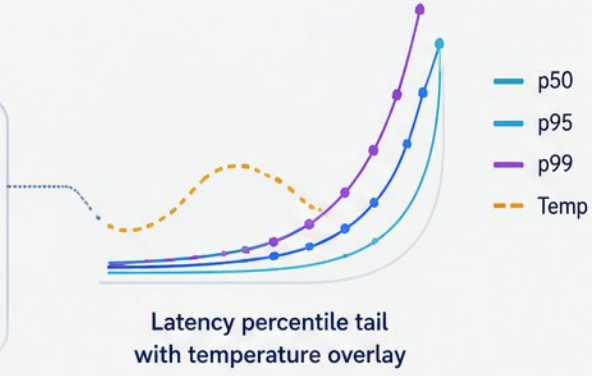
1. Build arm64 images with Docker Buildx; 32-bit armhf is fading out
2. Keep inference containers lean: python-slim base, model mounted as a volume
3. Set CPU and memory limits so one runaway container can't sink the device
4. Pass the NPU device node in; verify the runtime sees it first
5. Run a systemd unit with docker compose up -d; automate restarts and health checks



Monitoring, Update Strategy, and Fleet Maintenance

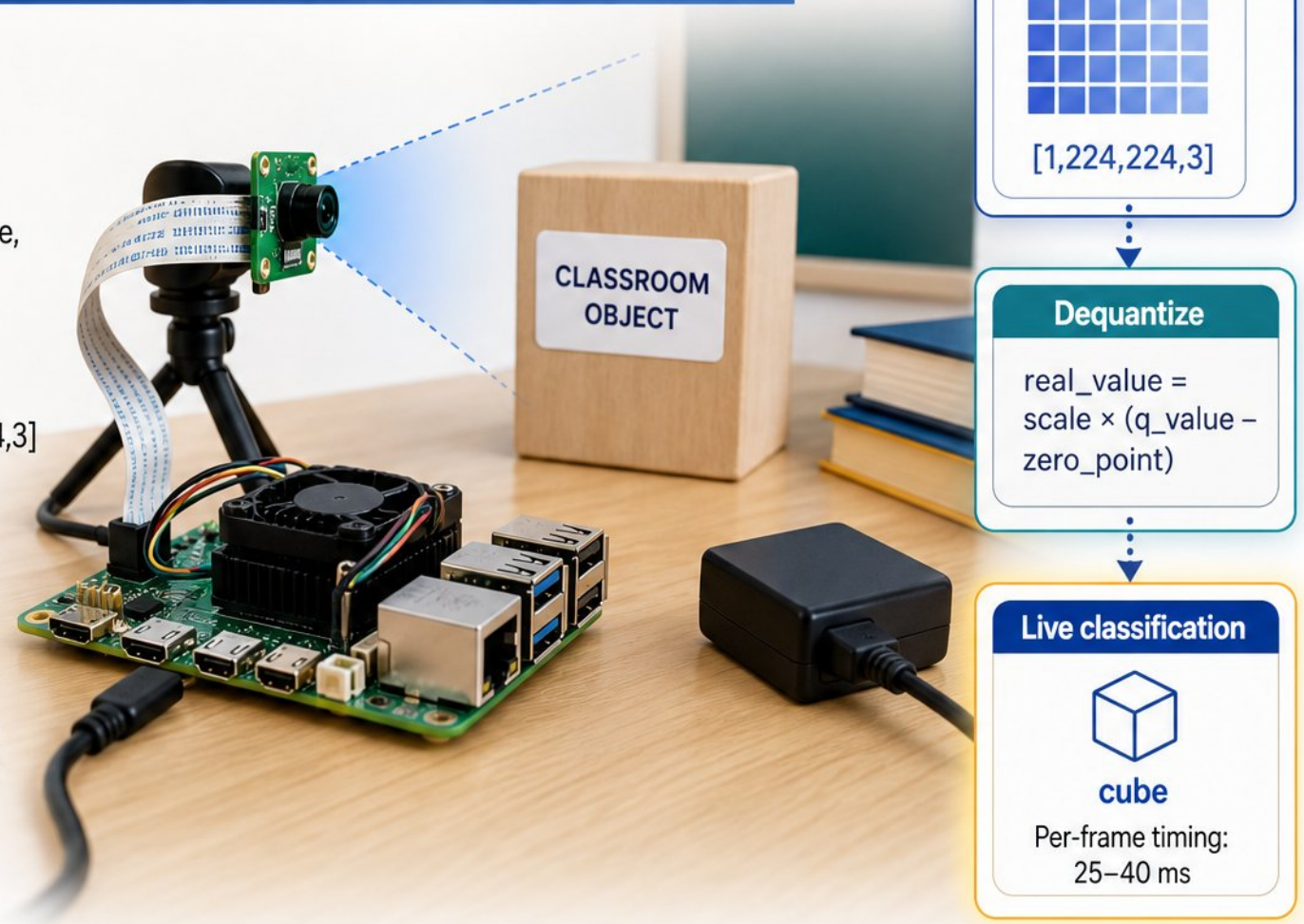
- Track p50/p95/p99 latency, SoC temperature, throttle flags, CPU/memory, NPU load
- Read temperature via vcgencmd and sysfs; push metrics out—devices sit behind NAT
- Tag model version with every metric to attribute latency or accuracy regressions
- Watch accuracy drift: sample inputs, keep a labeled audit set, set retrain threshold
- Immutable images, staged rollouts, atomic rollback (Mender, RAUC, balena, health-checked systemd)
- Handoff package: pinned versions, calibration sets, benchmark numbers, known limits

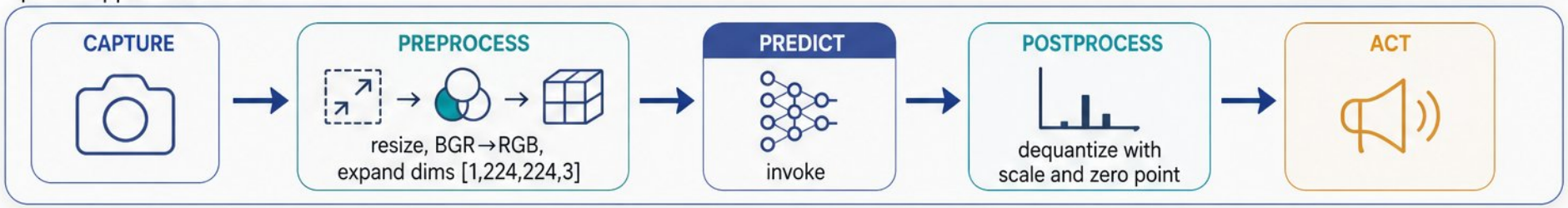




Project Walkthrough 1: Real-Time Image Classifier with Pi Camera

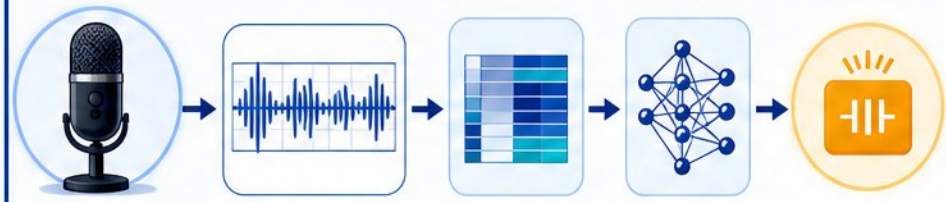
- **Hardware:** Pi 5, Camera Module 3, active cooler, 27 W supply, NVMe or A2 microSD
- **Software:** Picamera2 + tflite-runtime, INT8 MobileNetV2 at 224×224, num_threads=4
- **Pipeline:** capture array → resize, BGR→RGB → expand dims [1,224,224,3] → invoke → dequantize with scale and zero point
- **Expect** roughly 25–40 ms per classification on Pi 5 CPU — comfortably above 20 FPS
- **Teaching extension:** transfer-learn a classroom dataset, compare accuracy before/after INT8
- **Upgrade path:** swap .tflite for a compiled HEF, route camera through rpicam-apps for 30+ FPS





Project Walkthrough 2: Keyword Spotting and Project 3: Sensor Anomaly Detection

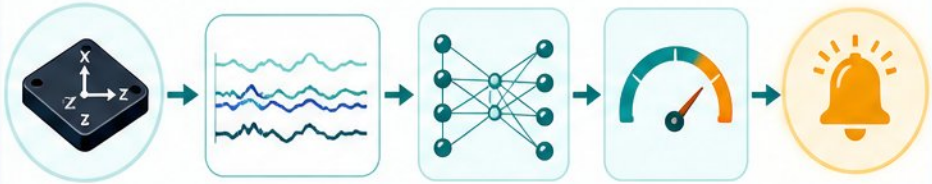
2 Project 2: Keyword Spotting



USB Mic → 1-Second Rolling Audio Buffer → Log-Mel Features → Tiny CNN/RNN → GPIO or MQTT Trigger

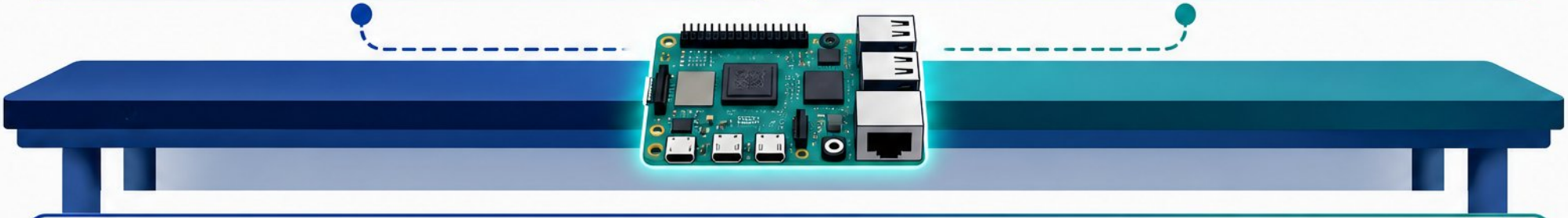
Rolling audio buffer + same preprocessing as training; GPIO or MQTT triggers

3 Project 3: Sensor Anomaly Detection



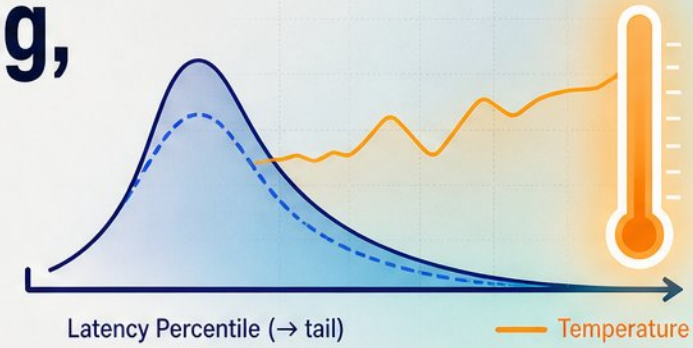
Accelerometer → Sliding Window of ~50 Samples → Small Autoencoder/LSTM → Anomaly Score → Alert if Score ≥ 0.75

Alert threshold e.g. 0.75; log inputs with scores to review false positives



Both run under 80–100 ms on Pi 5 — no camera, cheap classroom builds

Common Pitfalls, Troubleshooting, and Decision Checklist



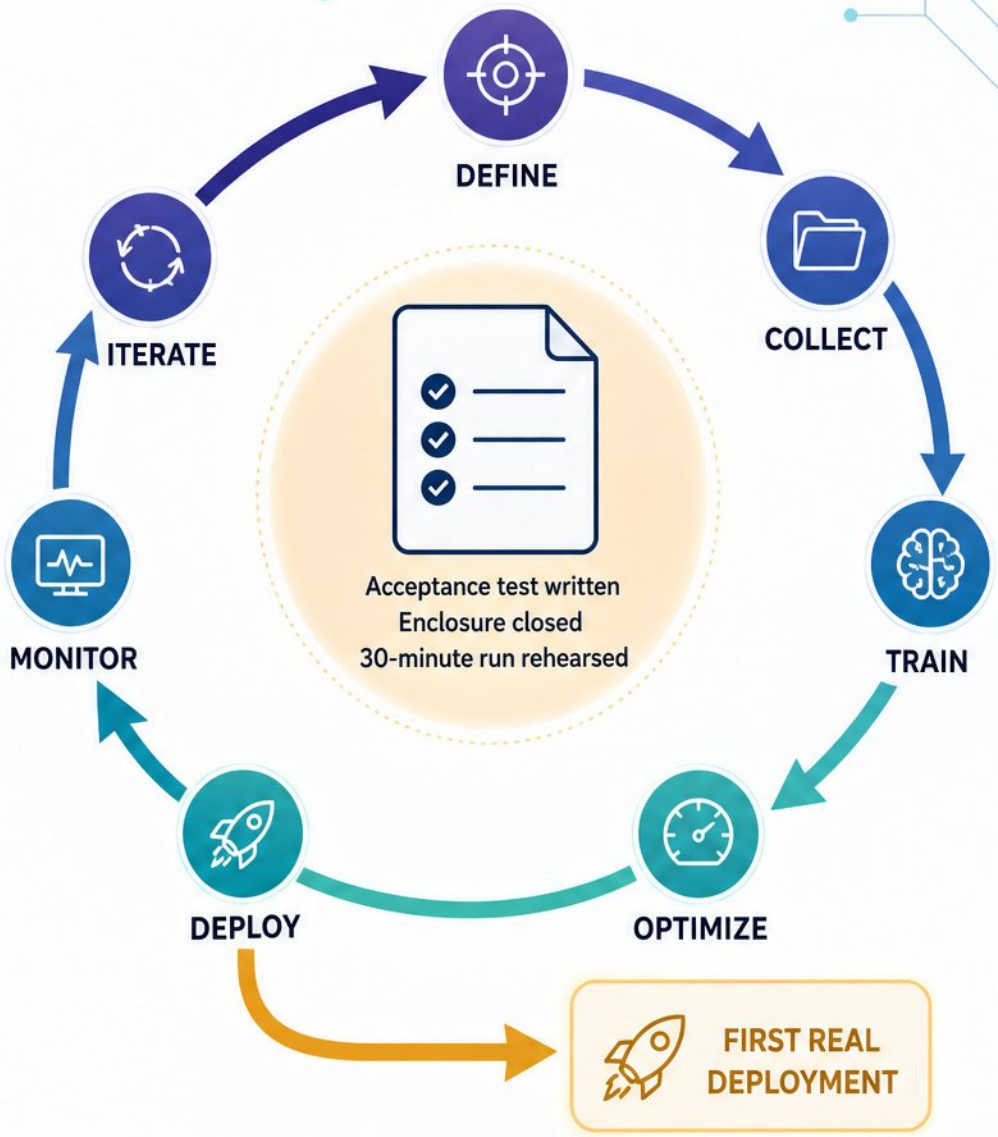
FIELD SYMPTOM	VERIFIED CAUSE
'No matching distribution' for tflite-runtime: reflash 64-bit, match your CPython wheel	- 'No matching distribution' for tflite-runtime: reflash 64-bit, match your CPython wheel
GLIBCXX and undefined-symbol import errors: uninstall full TensorFlow, keep tflite_runtime	- GLIBCXX and undefined-symbol import errors: uninstall full TensorFlow, keep tflite_runtime
'Regular TensorFlow ops are not supported': ops outside TFLITE_BUILTINS_INT8 — move to ONNX Runtime	- 'Regular TensorFlow ops are not supported': ops outside TFLITE_BUILTINS_INT8 — move to ONNX Runtime
40 ms jumping to 200 ms mid-run means thermal throttling: check cooling, PSU, governor	- 40 ms jumping to 200 ms mid-run means thermal throttling: check cooling, PSU, governor

DECISION CHECKLIST

- ✓ - Big post-quantization accuracy drop: calibration set not representative, or unfriendly architecture
- ✓ - Decision checklist: fits RAM/latency budget?
- ✓ - Decision checklist: quantizes acceptably?
- ✓ - Decision checklist: survives 30 min?
- ✓ - Decision checklist: rollback documented?

Where to Go Next: Resources, Community, and Your First Deployment

- **Official docs:** Raspberry Pi AI software, Picamera2 manual, OS release notes
- **Tooling:** TFLite/LiteRT and ONNX Runtime aarch64 guides
- **Models:** TensorFlow Hub, Hailo Model Zoo HEFs, Ultralytics YOLO export
- **Teaching:** Experience AI (ages 11–14) and the Computing Curriculum
- **Community:** Raspberry Pi and Hailo forums, r/LocalLLaMA, r/raspberry_pi
- **First deployment this month:** write the acceptance test, rehearse with enclosure closed, then scale
- **Workflow loop:** measure → reoptimize → retrain → change hardware (cheapest first)



PersonWise.

PersonWise • AI digital-human course platform

Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/35e0351d/public