# Smoke Detection System - Master Plan

## Phase 1 – High Impact / Low Effort (Performance + Stability + Self-Healing)

* GStreamer ingestion in rtsp\_worker.py with TCP transport, latency control, auto-reconnect.
* Auto hardware acceleration detection (NVIDIA / Intel / Apple / RPi / CPU fallback).
* Auto codec selection (H.264 / H.265).
* Downscale detection frames to target resolution; keep full-res only for event clips.
* FPS throttling to target\_fps from camera config.
* Enable Numba caching (cache=True), warm-up with real frame sizes.
* Remove redundant .astype() calls.
* Limit frame\_buffer to 1 per stage per camera (JPEG bytes only).
* Pre-allocate rolling buffers.
* Rate-limit logs per camera.
* Smart log rotation (size/date or error spike).
* Watchdog core (Tier 1: Restart camera thread; Tier 2: Restart detection loop/dashboard; Tier 3: Trigger systemd restart).
* Dashboard watchdog controls: /api/watchdog/status, /api/watchdog/restart/camera/<id>, /api/watchdog/restart/service.
* systemd integration with health checks, auto-restart, cleanup.
* Phone call alert integration (Twilio/SIP).
* Multi-channel escalation (call → second number → SMS → push).
* Mobile push snapshot in notifications.
* Camera feed fallback to substream on failure.
* Encrypted camera stream support (RTSPS or VPN).
* Per-camera performance dashboard (FPS, latency, decode method).

## Phase 2 – Medium Impact / Medium Effort (Dashboard + Memory Control)

* Replace MJPEG with WebSocket snapshot streaming (1–2 FPS).
* Lazy-load dashboard streams (only visible cameras/stages send data).
* Bound lists with deque(maxlen=...).
* Batch metrics updates every N seconds.
* Event context capture (30s before, heatmap, metrics snapshot).
* Mobile app acknowledgement to stop repeat alerts.
* Local + cloud event storage (recent locally, archive to S3).
* Debug snapshot mode for troubleshooting.
* Alert summary reports (daily/weekly PDF/HTML).
* Per-camera historical metrics (InfluxDB/TimescaleDB).
* API rate limiting + JWT authentication.
* Multi-language dashboard (EN/PT-BR).
* Advanced user roles (Viewer / Manager / Admin).

## Phase 3 – High Impact / High Effort (Scalability + Responsiveness)

* WebRTC live view (on demand, adaptive bitrate).
* Background encoding for event clips.
* Adaptive FPS based on motion/smoke activity.
* Async backend (uvicorn / hypercorn).
* Incident replay feature in dashboard.
* Remote firmware/software update hook (package server).
* Auto-test mode (scheduled end-to-end detection test).

## Phase 4 – Low Effort / Quality of Life

* Structured JSON logging.
* Dashboard FPS limit (client-side).
* Camera health checks + auto-restart.
* systemd health commands for admin.

## Mobile Apps – Minimal Version

* Built in React Native (one codebase for iOS + Android).
* Week 1: Auth → Camera list + health → Event list & snapshots.
* Week 2: Push notifications (with snapshot) → Event clip viewer → Settings → TestFlight / Play internal test.
* Estimated time: 7–10 working days with you + experienced dev + me + Cursor.

## Fast-Track Backend Plan (~10 Days)

* Days 1–3: GStreamer ingestion + GPU detection + FPS control + Numba caching.
* Days 4–5: Watchdog + systemd + alerts + feed fallback + log control.
* Days 6–7: WebSocket snapshots + lazy-load streams + performance dashboard.
* Days 8–10: Event context capture + mobile push snapshot + local/cloud storage.