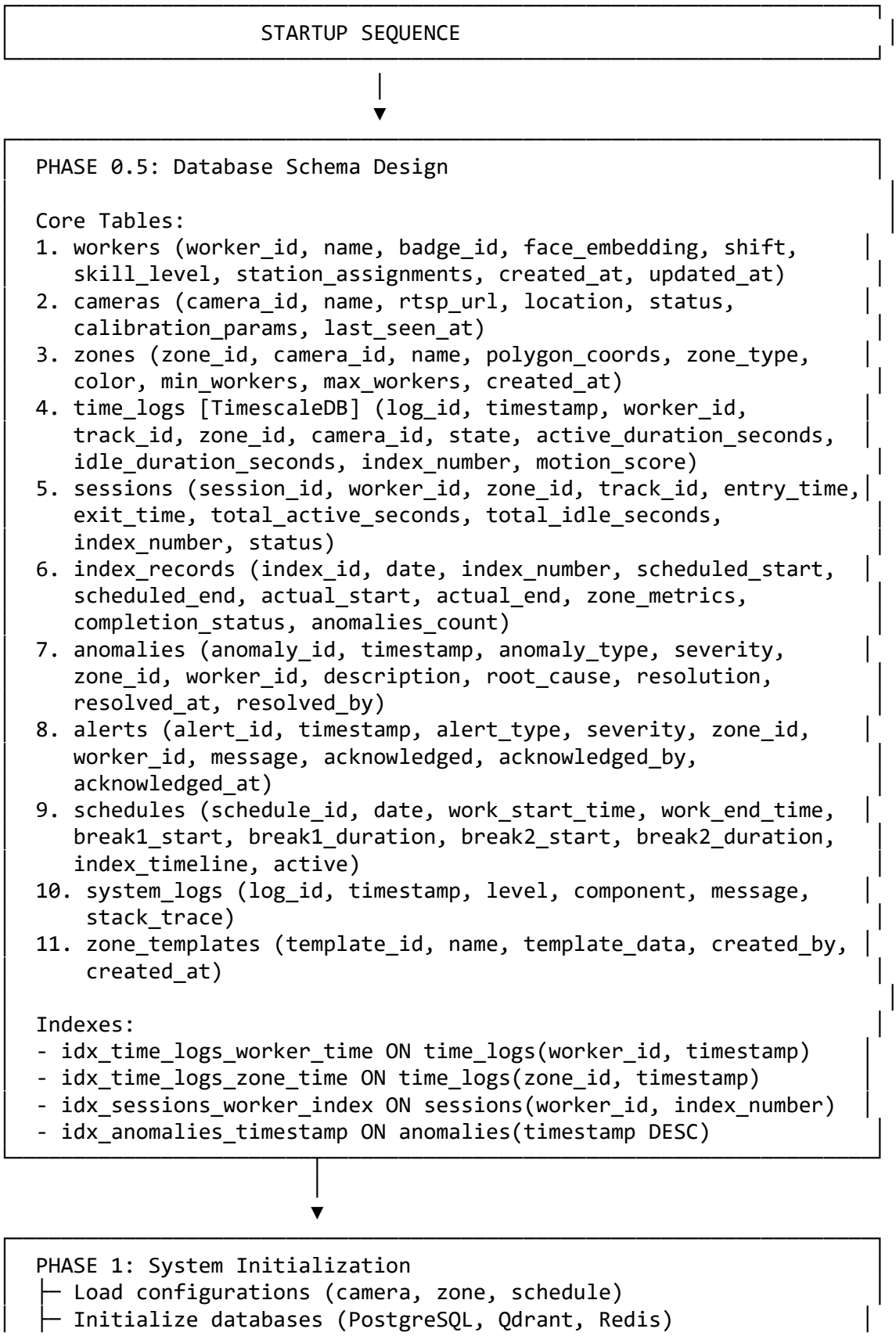
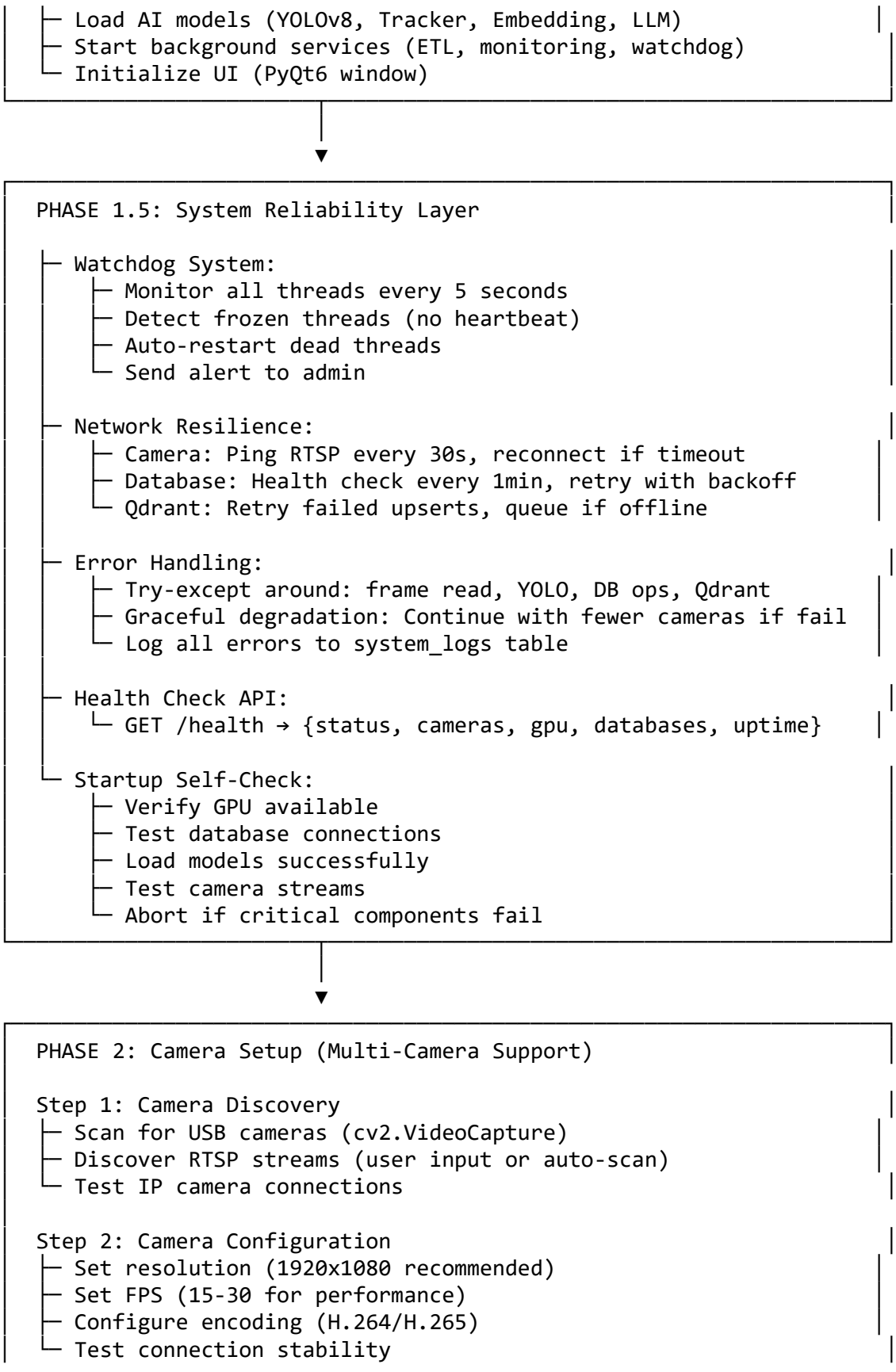
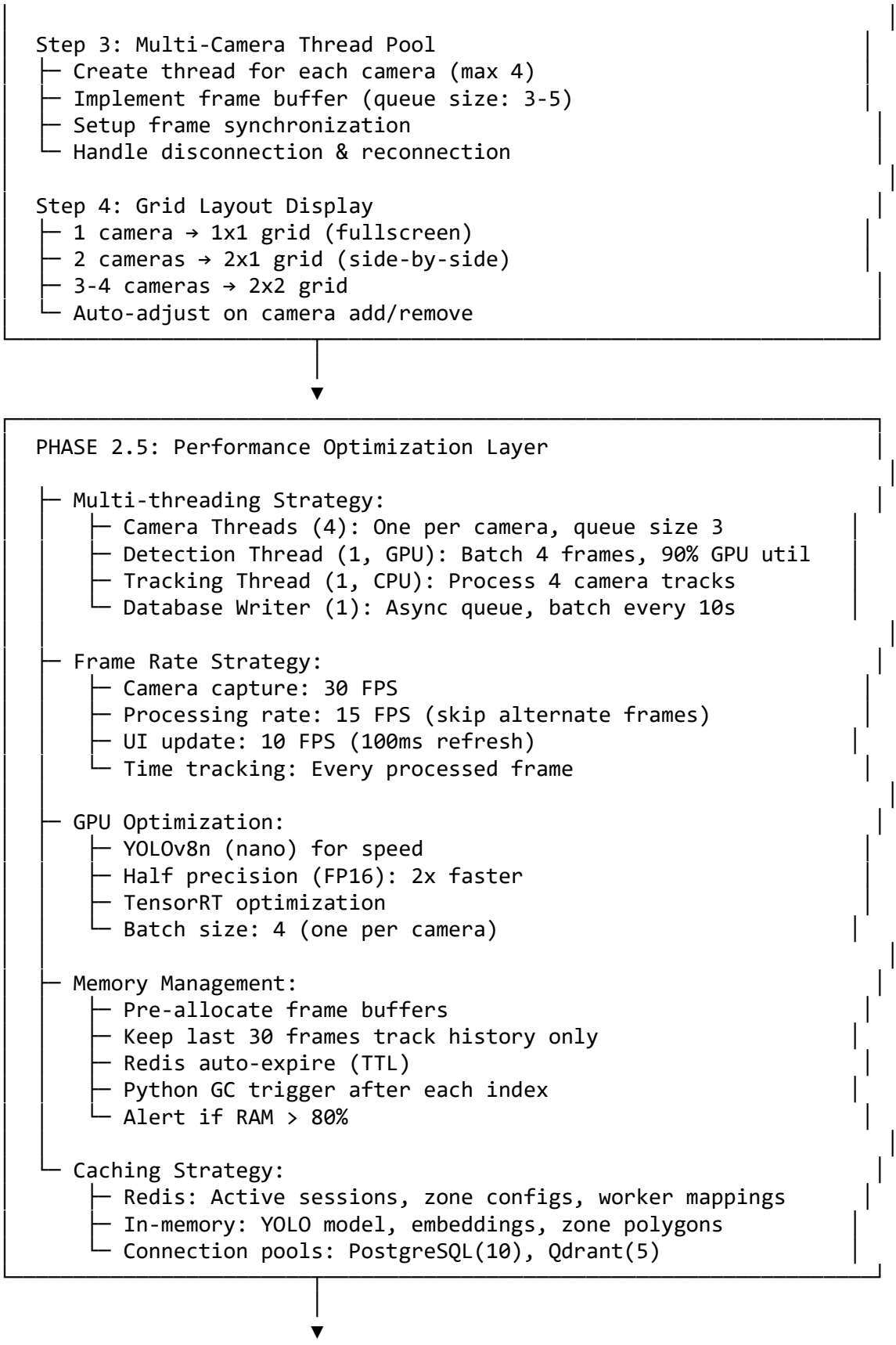
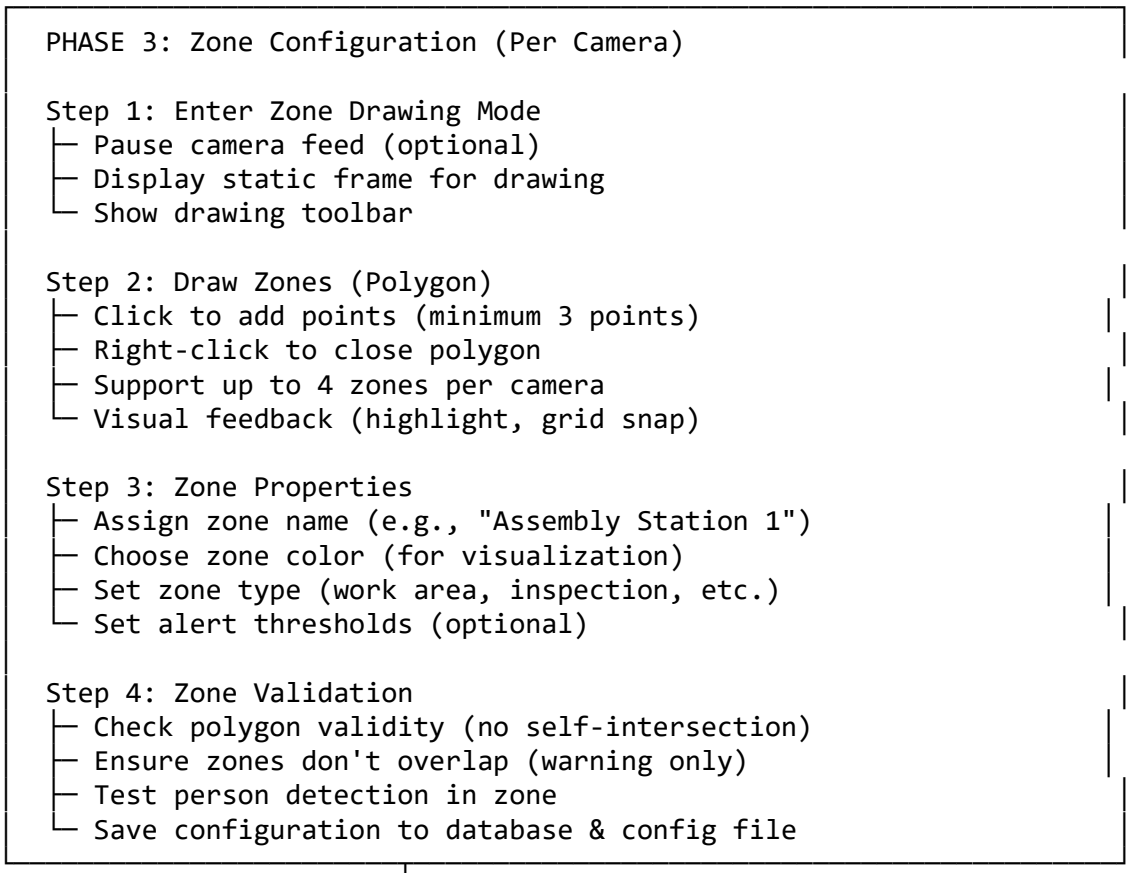
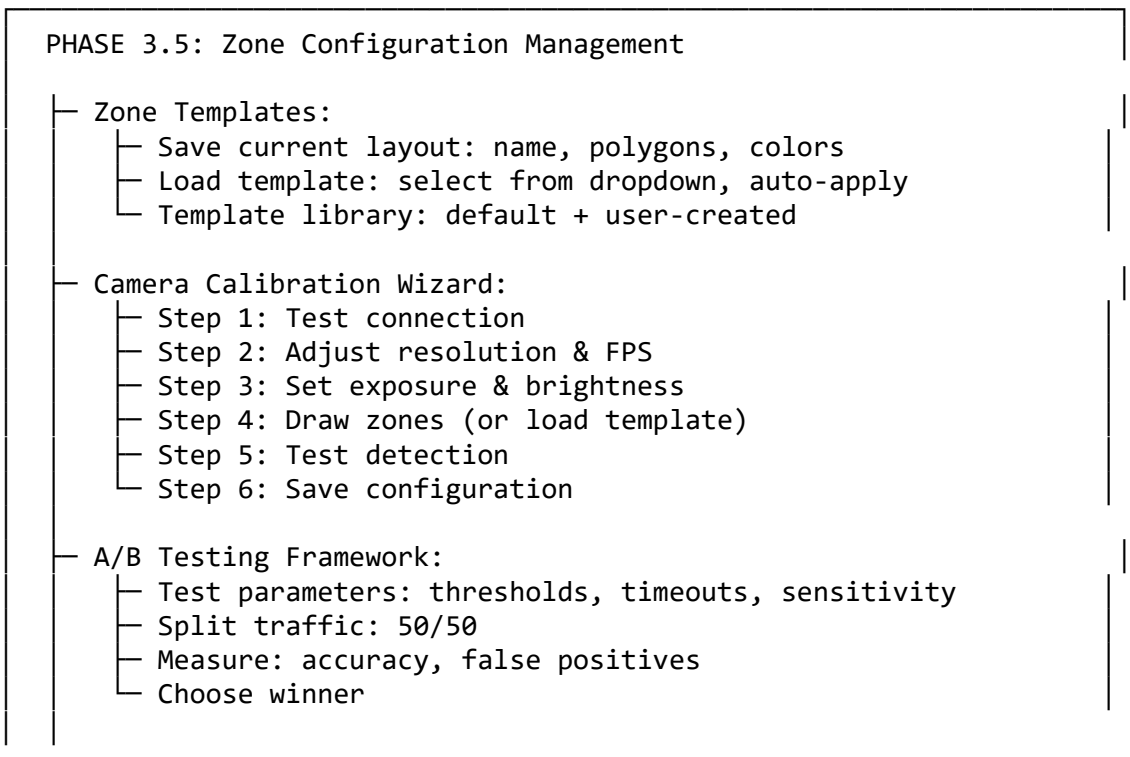

STARTUP SEQUENCE
PHASE 0.5: Database Schema Design
Core Tables:
1. workers (worker_id, name, badge_id, face_embedding, shift, skill_level, station_assignments, created_at, updated_at)
2. cameras (camera_id, name, rtsp_url, location, status, calibration_params, last_seen_at)
3. zones (zone_id, camera_id, name, polygon_coords, zone_type, color, min_workers, max_workers, created_at)
4. time_logs [TimescaleDB] (log_id, timestamp, worker_id, track_id, zone_id, camera_id, state, active_duration_seconds, idle_duration_seconds, index_number, motion_score)
5. sessions (session_id, worker_id, zone_id, track_id, entry_time, exit_time, total_active_seconds, total_idle_seconds, index_number, status)
6. index_records (index_id, date, index_number, scheduled_start, scheduled_end, actual_start, actual_end, zone_metrics, completion_status, anomalies_count)
7. anomalies (anomaly_id, timestamp, anomaly_type, severity, zone_id, worker_id, description, root_cause, resolution, resolved_at, resolved_by)
8. alerts (alert_id, timestamp, alert_type, severity, zone_id, worker_id, message, acknowledged, acknowledged_by, acknowledged_at)
9. schedules (schedule_id, date, work_start_time, work_end_time, break1_start, break1_duration, break2_start, break2_duration, index_timeline, active)
10. system_logs (log_id, timestamp, level, component, message, stack_trace)
11. zone_templates (template_id, name, template_data, created_by, created_at)
Indexes:
- idx_time_logs_worker_time ON time_logs(worker_id, timestamp)
- idx_time_logs_zone_time ON time_logs(zone_id, timestamp)
- idx_sessions_worker_index ON sessions(worker_id, index_number)
- idx_anomalies_timestamp ON anomalies(timestamp DESC)
PHASE 1: System Initialization
├─ Load configurations (camera, zone, schedule)
├─ Initialize databases (PostgreSQL, Qdrant, Redis)


├─ Load AI models (YOLOv8, Tracker, Embedding, LLM)
├─ Start background services (ETL, monitoring, watchdog)
└─ Initialize UI (PyQt6 window)
PHASE 1.5: System Reliability Layer
├─ Watchdog System:
├─ Monitor all threads every 5 seconds
├─ Detect frozen threads (no heartbeat)
├─ Auto-restart dead threads
└─ Send alert to admin
├─ Network Resilience:
├─ Camera: Ping RTSP every 30s, reconnect if timeout
├─ Database: Health check every 1min, retry with backoff
└─ Qdrant: Retry failed upserts, queue if offline
├─ Error Handling:
├─ Try-except around: frame read, YOLO, DB ops, Qdrant
├─ Graceful degradation: Continue with fewer cameras if fail
└─ Log all errors to system_logs table
├─ Health Check API:
└─ GET /health → {status, cameras, gpu, databases, uptime}
└─ Startup Self-Check:
├─ Verify GPU available
├─ Test database connections
├─ Load models successfully
├─ Test camera streams
└─ Abort if critical components fail
PHASE 2: Camera Setup (Multi-Camera Support)
Step 1: Camera Discovery
├─ Scan for USB cameras (cv2.VideoCapture)
├─ Discover RTSP streams (user input or auto-scan)
└─ Test IP camera connections
Step 2: Camera Configuration
├─ Set resolution (1920x1080 recommended)
├─ Set FPS (15-30 for performance)
├─ Configure encoding (H.264/H.265)
└─ Test connection stability


Step 3: Multi-Camera Thread Pool
├─ Create thread for each camera (max 4)
├─ Implement frame buffer (queue size: 3-5)
├─ Setup frame synchronization
└─ Handle disconnection & reconnection
Step 4: Grid Layout Display
├─ 1 camera → 1x1 grid (fullscreen)
├─ 2 cameras → 2x1 grid (side-by-side)
├─ 3-4 cameras → 2x2 grid
└─ Auto-adjust on camera add/remove
PHASE 2.5: Performance Optimization Layer
├─ Multi-threading Strategy:
│ ├─ Camera Threads (4): One per camera, queue size 3
│ ├─ Detection Thread (1, GPU): Batch 4 frames, 90% GPU util
│ ├─ Tracking Thread (1, CPU): Process 4 camera tracks
│ └─ Database Writer (1): Async queue, batch every 10s
├─ Frame Rate Strategy:
│ ├─ Camera capture: 30 FPS
│ ├─ Processing rate: 15 FPS (skip alternate frames)
│ ├─ UI update: 10 FPS (100ms refresh)
│ └─ Time tracking: Every processed frame
├─ GPU Optimization:
│ ├─ YOLOv8n (nano) for speed
│ ├─ Half precision (FP16): 2x faster
│ ├─ TensorRT optimization
│ └─ Batch size: 4 (one per camera)
├─ Memory Management:
│ ├─ Pre-allocate frame buffers
│ ├─ Keep last 30 frames track history only
│ ├─ Redis auto-expire (TTL)
│ ├─ Python GC trigger after each index
│ └─ Alert if RAM > 80%
└─ Caching Strategy:
├─ Redis: Active sessions, zone configs, worker mappings
├─ In-memory: YOLO model, embeddings, zone polygons
└─ Connection pools: PostgreSQL(10), Qdrant(5)


PHASE 3: Zone Configuration (Per Camera)
Step 1: Enter Zone Drawing Mode
├─ Pause camera feed (optional)
├─ Display static frame for drawing
└─ Show drawing toolbar
Step 2: Draw Zones (Polygon)
├─ Click to add points (minimum 3 points)
├─ Right-click to close polygon
├─ Support up to 4 zones per camera
└─ Visual feedback (highlight, grid snap)
Step 3: Zone Properties
├─ Assign zone name (e.g., "Assembly Station 1")
├─ Choose zone color (for visualization)
├─ Set zone type (work area, inspection, etc.)
└─ Set alert thresholds (optional)
Step 4: Zone Validation
├─ Check polygon validity (no self-intersection)
├─ Ensure zones don't overlap (warning only)
├─ Test person detection in zone
└─ Save configuration to database & config file


PHASE 3.5: Zone Configuration Management
├─ Zone Templates:
│ ├─ Save current layout: name, polygons, colors
│ ├─ Load template: select from dropdown, auto-apply
│ └─ Template library: default + user-created
├─ Camera Calibration Wizard:
│ ├─ Step 1: Test connection
│ ├─ Step 2: Adjust resolution & FPS
│ ├─ Step 3: Set exposure & brightness
│ ├─ Step 4: Draw zones (or load template)
│ ├─ Step 5: Test detection
│ └─ Step 6: Save configuration
├─ A/B Testing Framework:
│ ├─ Test parameters: thresholds, timeouts, sensitivity
│ ├─ Split traffic: 50/50
│ ├─ Measure: accuracy, false positives
│ └─ Choose winner

```
  └─ Configuration Versioning:
      ├─ Track changes
      ├─ Rollback capability
      └─ Audit log: who, what, when
```

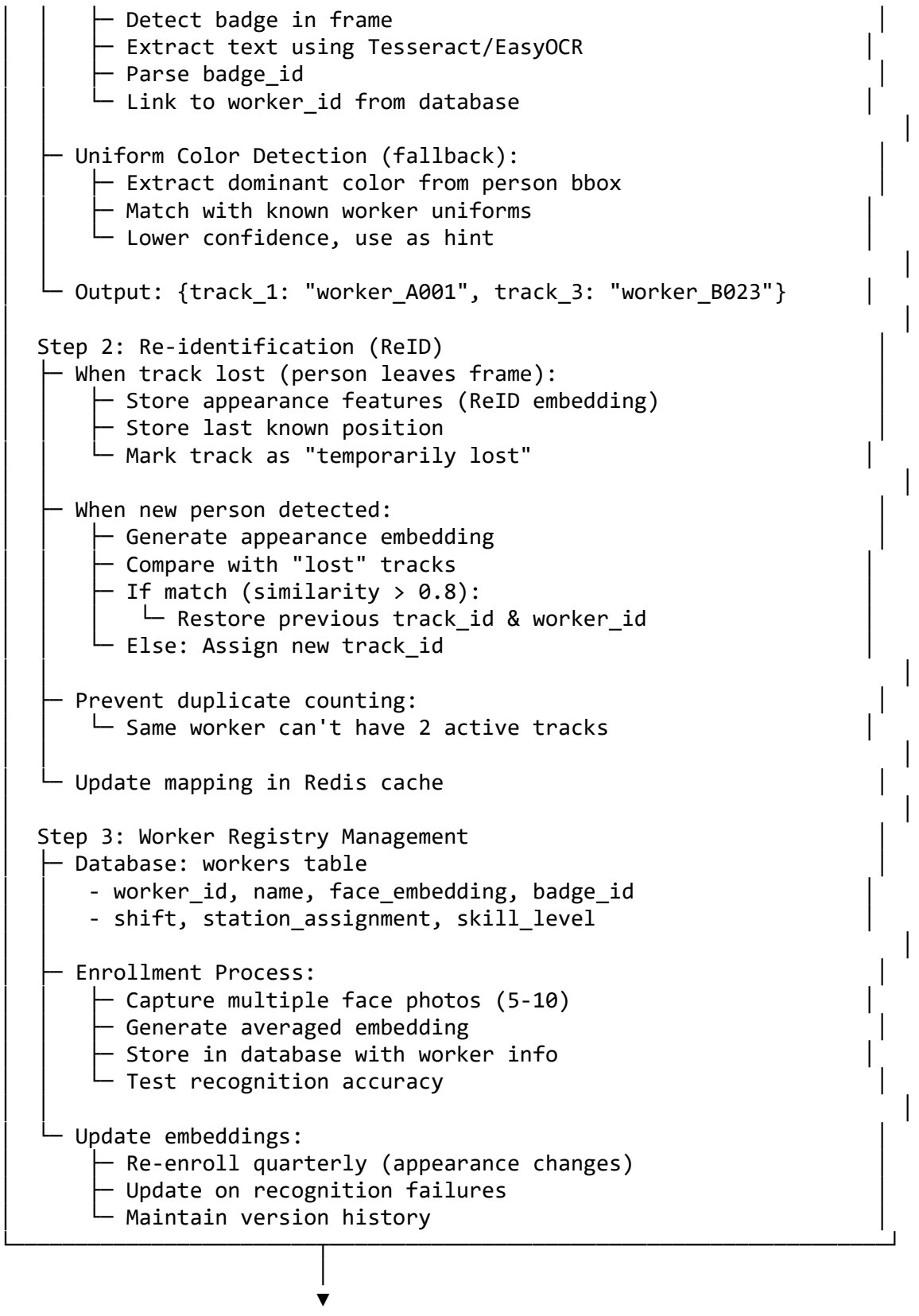


```
PHASE 4: Work Schedule Setup

Step 1: Define Daily Schedule
├─ Set work start time (e.g., 08:00)
├─ Set work end time (e.g., 17:00)
└─ Total work duration: 9 hours = 540 minutes

Step 2: Configure Break Times
├─ Break 1: Start time & duration (e.g., 10:00, 15 min)
├─ Break 2: Start time & duration (e.g., 15:00, 15 min)
└─ Total break time: 30 minutes

Step 3: Calculate Index Schedule (11 Indices)
├─ Net work time: 540 - 30 = 510 minutes
├─ Index duration: 510 ÷ 11 ≈ 46.36 minutes
├─ Rounded to: 57 minutes per index (with buffer)
└─ Generate timeline with break exclusions

Example Timeline:
Index 1:  08:00 - 08:57
Index 2:  08:57 - 09:54
Index 3:  09:54 - 10:00 → BREAK 1 (10:00-10:15)
          10:15 - 10:36  (complete remaining 21 min)
Index 4:  10:36 - 11:33
...
Index 8:  14:30 - 15:00 → BREAK 2 (15:00-15:15)
          15:15 - 15:42  (complete remaining 27 min)
...
Index 11: 16:03 - 17:00

Step 4: Store Schedule
└─ Save to database & Redis cache for quick access
```



```
PHASE 5: Real-time Detection & Tracking (Continuous Loop)

FOR EACH CAMERA (Parallel Processing):

Step 1: Frame Acquisition
├─ Read frame from camera stream
```

```
│  ├─ Check frame validity (not None, correct size)                    │
│  ├─ Handle dropped frames (skip or interpolate)                      │
│  └─ Add to processing queue                                          │
│                                                                      │
│  Step 2: Person Detection (YOLOv8)                                   │
│  ├─ Preprocess frame (resize, normalize)                             │
│  ├─ Run YOLO inference on GPU (batch of 4 frames)                    │
│  ├─ Filter detections (confidence > 0.5, class="person")             │
│  ├─ Extract bounding boxes [x, y, w, h]                              │
│  └─ Output: List of person detections with confidence                │
│                                                                      │
│  Step 3: Person Tracking (DeepSORT/ByteTrack)                        │
│  ├─ Update tracker with new detections                               │
│  ├─ Assign unique tracking ID to each person                         │
│  ├─ Handle occlusion (maintain ID when hidden)                       │
│  ├─ Re-identify after temporary disappearance                        │
│  └─ Remove stale tracks (disappeared > 30 frames)                    │
│                                                                      │
│  Step 4: Zone Matching                                               │
│  ├─ For each tracked person:                                         │
│  │   ├─ Calculate person center point (bbox center)                  │
│  │   ├─ Check if center is inside any zone polygon                   │
│  │   │   (use point-in-polygon algorithm)                            │
│  │   ├─ If inside: Record (track_id, zone_id, timestamp)             │
│  │   └─ If outside all zones: Record as "unassigned"                 │
│  └─ Handle multi-zone overlap (prioritize by area)                   │
│                                                                      │
│  Step 5: Motion Detection (Per Person)                               │
│  ├─ Store previous frame bbox for each track_id                      │
│  ├─ Calculate displacement: distance = sqrt((x2-x1)²+(y2-y1)²)       │
│  ├─ Determine motion state:                                          │
│  │   ├─ Moving: distance > threshold (e.g., 10 pixels)               │
│  │   └─ Idle: distance ≤ threshold                                   │
│  ├─ Track consecutive idle frames                                    │
│  └─ Trigger state change after 60s (1800 frames @ 30fps)             │
└──────────────────────────────────────────────────────────────────────┘
                              │
                              ▼
┌──────────────────────────────────────────────────────────────────────┐
│  PHASE 5.5: Worker Identification & Re-identification                │
│                                                                      │
│  Step 1: Identity Detection                                          │
│  ├─ Face Recognition (DeepFace/ArcFace/FaceNet):                     │
│  │   ├─ Detect faces in bbox                                         │
│  │   ├─ Generate face embedding (512-dim vector)                     │
│  │   ├─ Compare with workers database embeddings                     │
│  │   ├─ Match if similarity > 0.7                                    │
│  │   └─ Link: track_id → worker_id                                   │
│  │                                                                   │
│  ├─ ID Badge/QR Code Detection (OCR):                                │
```


├─ Detect badge in frame
├─ Extract text using Tesseract/EasyOCR
├─ Parse badge_id
└─ Link to worker_id from database
├─ Uniform Color Detection (fallback):
├─ Extract dominant color from person bbox
├─ Match with known worker uniforms
└─ Lower confidence, use as hint
└─ Output: {track_1: "worker_A001", track_3: "worker_B023"}
Step 2: Re-identification (ReID)
├─ When track lost (person leaves frame):
├─ Store appearance features (ReID embedding)
├─ Store last known position
└─ Mark track as "temporarily lost"
├─ When new person detected:
├─ Generate appearance embedding
├─ Compare with "lost" tracks
├─ If match (similarity > 0.8):
└─ Restore previous track_id & worker_id
└─ Else: Assign new track_id
├─ Prevent duplicate counting:
└─ Same worker can't have 2 active tracks
└─ Update mapping in Redis cache
Step 3: Worker Registry Management
├─ Database: workers table
- worker_id, name, face_embedding, badge_id
- shift, station_assignment, skill_level
├─ Enrollment Process:
├─ Capture multiple face photos (5-10)
├─ Generate averaged embedding
├─ Store in database with worker info
└─ Test recognition accuracy
└─ Update embeddings:
├─ Re-enroll quarterly (appearance changes)
├─ Update on recognition failures
└─ Maintain version history
PHASE 5.6: Data Quality Validation

```
├─ Detection Validation:
│   ├─ Confidence Threshold: Require conf > 0.6
│   ├─ Bounding Box Sanity Check:
│   │   - Min size: 30x50 pixels
│   │   - Max size: 500x800 pixels
│   │   - Aspect ratio: 0.3 to 0.8 (person-like)
│   └─ False Positive Filtering:
│       - Must appear in 3+ consecutive frames
│       - Filter shadows, reflections
│       - Remove static "persons" (objects)
│
├─ Occlusion Handling:
│   ├─ Detect occlusion: bbox shrinks, confidence drops
│   ├─ Maintain track: Kalman filter prediction, 2s alive
│   └─ Re-associate: Match by ReID embedding
│
├─ Data Integrity Checks:
│   ├─ Before insert: Validate FKs, timestamps, durations
│   ├─ Periodic validation: Daily at midnight
│   │   - Check orphaned records
│   │   - Fix inconsistencies
│   └─ Quality metrics: Track accuracy, false positive rate
│
└─ Zone-specific Tuning:
    ├─ Different thresholds per zone
    ├─ Store per-zone configuration
    └─ Adjust based on lighting, camera angle
```

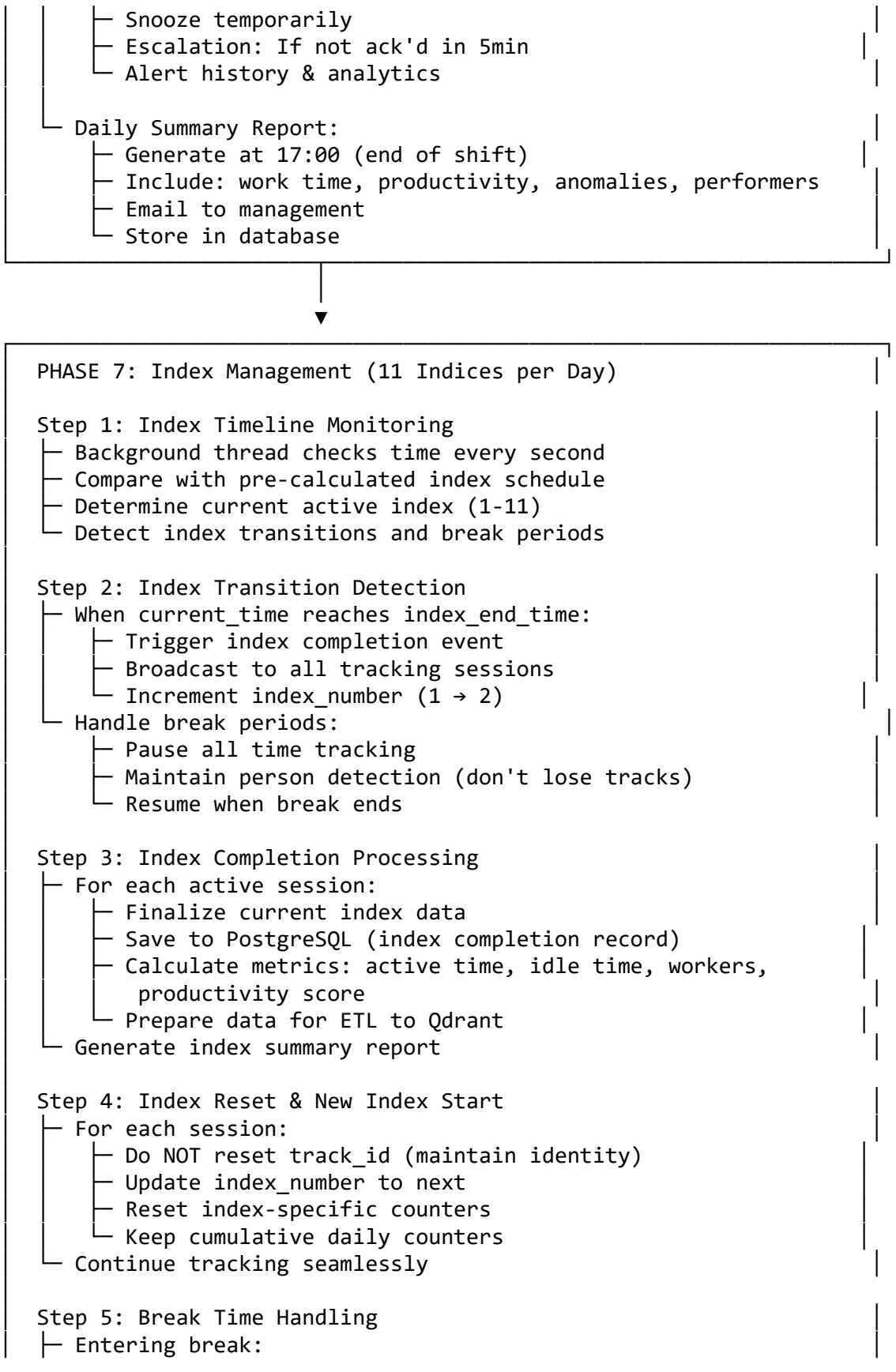


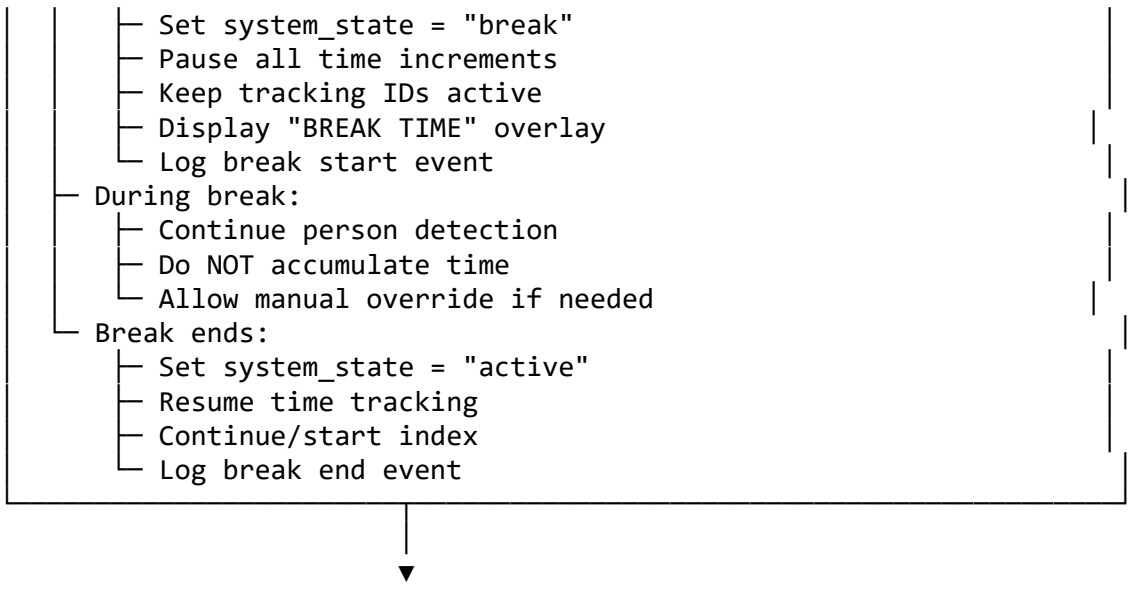
```
PHASE 6: Intelligent Time Tracking (Per Person, Per Zone)

Step 1: Initialize Tracking Session
├─ When person first detected in zone:
│   ├─ Create session record in Redis
│   ├─ session_id = f"{worker_id}_{zone_id}_{timestamp}"
│   ├─ Initialize: active_time=0, idle_time=0
│   └─ Set state = "active"
└─ Store: {worker_id, zone_id, start_time, state, timers}

Step 2: Active State Processing
├─ Every frame (while person moving):
│   ├─ Increment active_time += frame_duration (1/FPS)
│   ├─ Reset idle counter = 0
│   └─ Update last_active_timestamp
├─ If motion stops:
│   ├─ Start counting idle_frames++
│   └─ Continue incrementing active_time
```

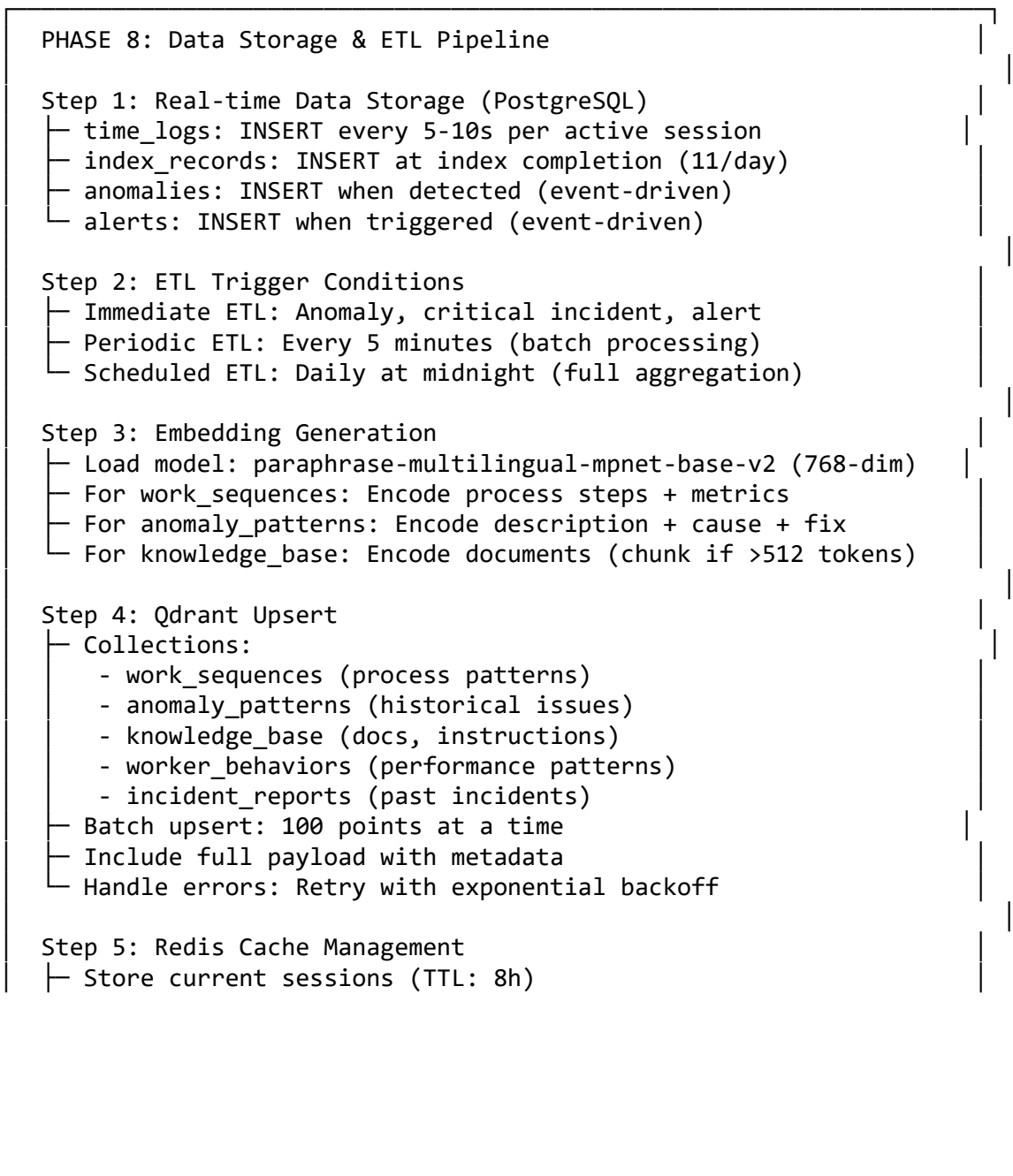
```
└─ Store real-time state in Redis

Step 3: Idle State Transition
├─ When idle_frames >= 1800 (60 seconds):
│   ├─ Change state to "idle"
│   ├─ Stop incrementing active_time
│   ├─ Record idle_start_timestamp
│   └─ Trigger alert (if configured)
└─ Continue monitoring for movement

Step 4: Resume from Idle
├─ When movement detected after idle:
│   ├─ Calculate idle_duration = now - idle_start_timestamp
│   ├─ Log idle period to database
│   ├─ Change state back to "active"
│   └─ Resume incrementing active_time
└─ Do NOT reset active_time (accumulate continuously)

Step 5: Zone Exit Handling
├─ When person leaves zone:
│   ├─ Save session to PostgreSQL:
│   │   - session_id, worker_id, zone_id
│   │   - entry_time, exit_time
│   │   - total_active_time, total_idle_time
│   │   - index_number
│   ├─ Clear Redis session
│   └─ Prepare for re-entry (new session)
└─ If re-enters: Create new session, continue for same index
```

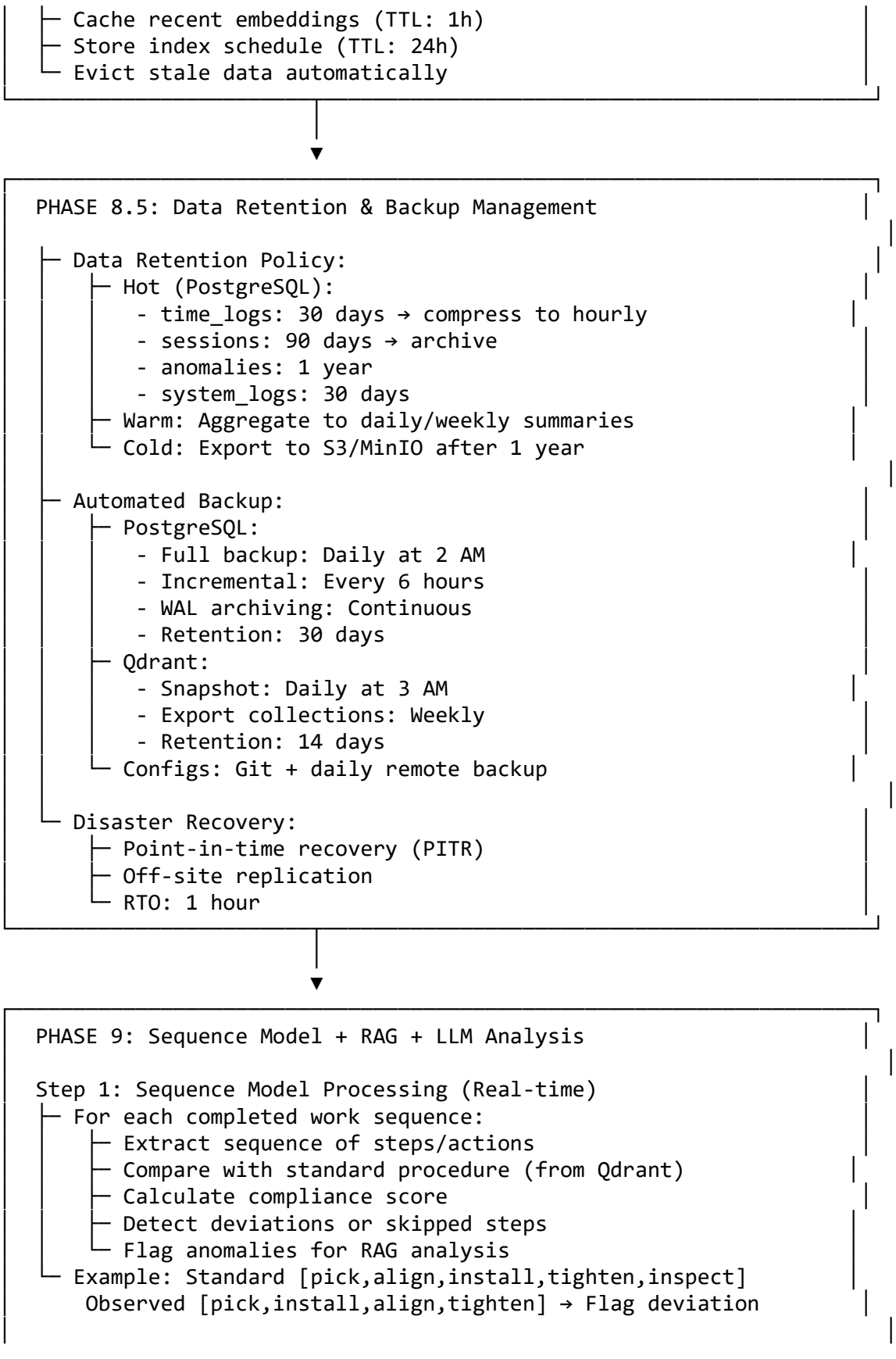
▼

```
PHASE 6.5: Alert & Notification System (Real-time)

├─ Alert Triggers:
│   ├─ Idle Threshold: IF idle_time > 300s → Alert
│   ├─ Zone Violation: IF worker in restricted zone → Alert
│   ├─ No Worker: IF critical_zone = 0 workers > 2min → Alert
│   ├─ Productivity Drop: IF efficiency < 70% → Alert
│   └─ Anomaly: IF sequence_model flags → Alert
│
├─ Notification Channels:
│   ├─ In-App (PyQt6): Toast, alert panel, sound
│   ├─ Email (SMTP): Immediate for critical, daily digest
│   ├─ LINE Notify: Push to supervisor with snapshot
│   ├─ Webhook: POST to external system
│   └─ SMS (Twilio): Critical only (safety, failures)
│
├─ Alert Management:
│   ├─ Acknowledge alerts
```


├─ Snooze temporarily
├─ Escalation: If not ack'd in 5min
└─ Alert history & analytics
└─ Daily Summary Report:
├─ Generate at 17:00 (end of shift)
├─ Include: work time, productivity, anomalies, performers
├─ Email to management
└─ Store in database
PHASE 7: Index Management (11 Indices per Day)
Step 1: Index Timeline Monitoring
├─ Background thread checks time every second
├─ Compare with pre-calculated index schedule
├─ Determine current active index (1-11)
└─ Detect index transitions and break periods
Step 2: Index Transition Detection
├─ When current_time reaches index_end_time:
├─ Trigger index completion event
├─ Broadcast to all tracking sessions
└─ Increment index_number (1 → 2)
└─ Handle break periods:
├─ Pause all time tracking
├─ Maintain person detection (don't lose tracks)
└─ Resume when break ends
Step 3: Index Completion Processing
├─ For each active session:
├─ Finalize current index data
├─ Save to PostgreSQL (index completion record)
├─ Calculate metrics: active time, idle time, workers,
productivity score
└─ Prepare data for ETL to Qdrant
└─ Generate index summary report
Step 4: Index Reset & New Index Start
├─ For each session:
├─ Do NOT reset track_id (maintain identity)
├─ Update index_number to next
├─ Reset index-specific counters
└─ Keep cumulative daily counters
└─ Continue tracking seamlessly
Step 5: Break Time Handling
├─ Entering break:


├─ Set system_state = "break"
├─ Pause all time increments
├─ Keep tracking IDs active
├─ Display "BREAK TIME" overlay
└─ Log break start event
├─ During break:
├─ Continue person detection
├─ Do NOT accumulate time
└─ Allow manual override if needed
└─ Break ends:
├─ Set system_state = "active"
├─ Resume time tracking
├─ Continue/start index
└─ Log break end event


PHASE 8: Data Storage & ETL Pipeline
Step 1: Real-time Data Storage (PostgreSQL)
├─ time_logs: INSERT every 5-10s per active session
├─ index_records: INSERT at index completion (11/day)
├─ anomalies: INSERT when detected (event-driven)
└─ alerts: INSERT when triggered (event-driven)
Step 2: ETL Trigger Conditions
├─ Immediate ETL: Anomaly, critical incident, alert
├─ Periodic ETL: Every 5 minutes (batch processing)
└─ Scheduled ETL: Daily at midnight (full aggregation)
Step 3: Embedding Generation
├─ Load model: paraphrase-multilingual-mpnet-base-v2 (768-dim)
├─ For work_sequences: Encode process steps + metrics
├─ For anomaly_patterns: Encode description + cause + fix
└─ For knowledge_base: Encode documents (chunk if >512 tokens)
Step 4: Qdrant Upsert
├─ Collections:
- work_sequences (process patterns)
- anomaly_patterns (historical issues)
- knowledge_base (docs, instructions)
- worker_behaviors (performance patterns)
- incident_reports (past incidents)
├─ Batch upsert: 100 points at a time
├─ Include full payload with metadata
└─ Handle errors: Retry with exponential backoff
Step 5: Redis Cache Management
├─ Store current sessions (TTL: 8h)

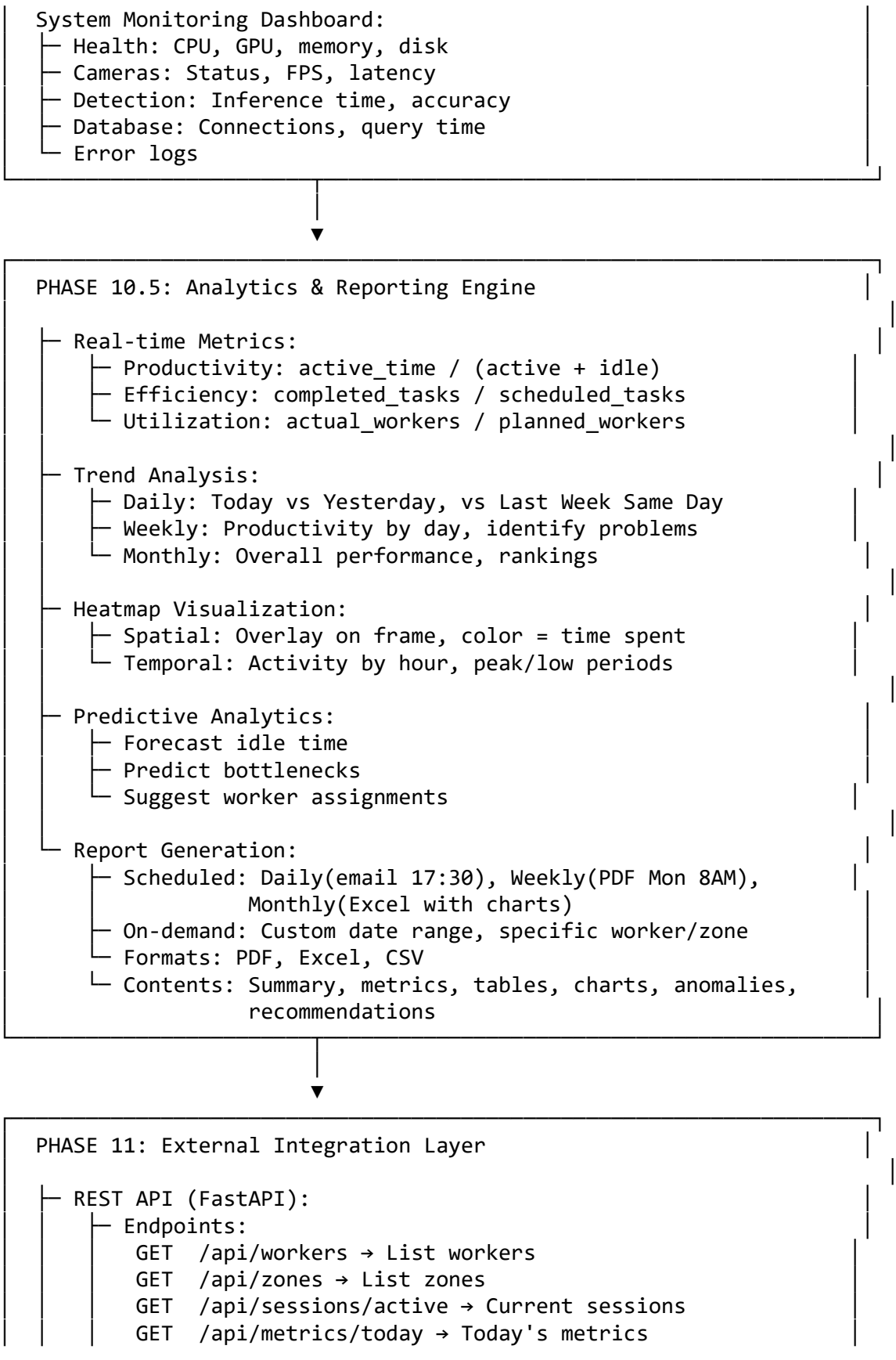

├─ Cache recent embeddings (TTL: 1h)
├─ Store index schedule (TTL: 24h)
└─ Evict stale data automatically
PHASE 8.5: Data Retention & Backup Management
├─ Data Retention Policy:
│ ├─ Hot (PostgreSQL):
│ - time_logs: 30 days → compress to hourly
│ - sessions: 90 days → archive
│ - anomalies: 1 year
│ - system_logs: 30 days
│ ├─ Warm: Aggregate to daily/weekly summaries
│ └─ Cold: Export to S3/MinIO after 1 year
├─ Automated Backup:
│ ├─ PostgreSQL:
│ - Full backup: Daily at 2 AM
│ - Incremental: Every 6 hours
│ - WAL archiving: Continuous
│ - Retention: 30 days
│ ├─ Qdrant:
│ - Snapshot: Daily at 3 AM
│ - Export collections: Weekly
│ - Retention: 14 days
│ └─ Configs: Git + daily remote backup
└─ Disaster Recovery:
├─ Point-in-time recovery (PITR)
├─ Off-site replication
└─ RTO: 1 hour
PHASE 9: Sequence Model + RAG + LLM Analysis
Step 1: Sequence Model Processing (Real-time)
├─ For each completed work sequence:
│ ├─ Extract sequence of steps/actions
│ ├─ Compare with standard procedure (from Qdrant)
│ ├─ Calculate compliance score
│ ├─ Detect deviations or skipped steps
│ └─ Flag anomalies for RAG analysis
└─ Example: Standard [pick,align,install,tighten,inspect]
Observed [pick,install,align,tighten] → Flag deviation

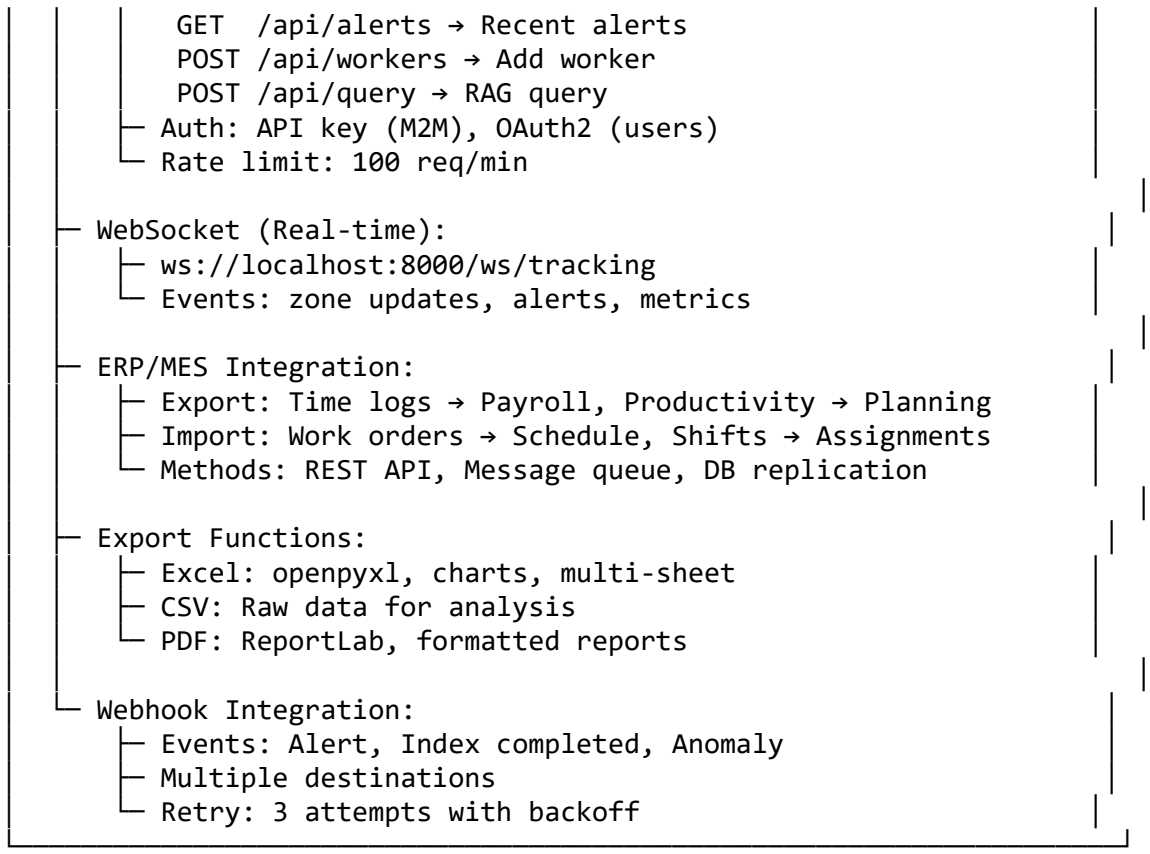
```
Step 2: RAG Query Interface
├─ User queries (NL): "ทำไม station 2 วันนี้ช้ากว่าปกติ"
└─ Automated queries: Anomaly detected → "Find similar incidents"

Step 3: Query Router
├─ Analyze query intent (keywords/NLP)
├─ Determine required sources:
│   - PostgreSQL: Current/recent data
│   - Qdrant: Historical patterns, knowledge
│   - Both: Comparative analysis
└─ Route to appropriate path

Step 4: Vector Search in Qdrant
├─ Generate query embedding
├─ Search collections (top-k=3-5):
│   - work_sequences, anomaly_patterns, knowledge_base,
│     worker_behaviors, incident_reports
├─ Apply filters (date, zone, severity)
└─ Return top matches with payloads

Step 5: SQL Query for Real-time Context
├─ Extract time-based data from PostgreSQL
├─ Aggregate metrics (hourly, daily, weekly)
└─ Combine with vector search results

Step 6: Context Assembly
├─ Combine: vector_results + sql_results + system_status
└─ Rank by relevance score

Step 7: Prompt Engineering
├─ Build structured prompt:
│   - System role: "คุณคือผู้เชี่ยวชาญด้านการผลิต"
│   - User question
│   - Current data (SQL)
│   - Historical context (Qdrant)
│   - Instructions: analyze, cite, recommend
└─ Optimize token usage

Step 8: LLM Inference (Ollama)
├─ Send to local LLM (Llama 3.1 8B / Qwen 2.5 7B)
├─ Stream response
├─ Parse structured output
└─ Handle errors (retry, fallback)

Step 9: Response Post-processing
├─ Extract key insights
├─ Add source citations
├─ Generate action items
└─ Store conversation for learning
```

```
Step 10: Feedback Loop
├─ User feedback (thumbs up/down)
├─ Log response quality
├─ Retrain/fine-tune periodically
└─ Update embeddings if needed
```

▼

```
PHASE 10: User Interface & Monitoring

UI Layout (PyQt6):

┌──────────────────────────────────────────────────────────┐
│ Top Menu Bar                                             │
│ [File][Camera][Zone][Schedule][Analytics][Settings]      │
└──────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────┐
│ Status Bar                                               │
│ Index: 3/11 | Active Workers: 8 | Alerts: 2 | GPU: 85%   │
└──────────────────────────────────────────────────────────┘

┌──────────────────────────────┬───────────────────────────┐
│ Camera Grid (2x2)            │ Right Sidebar             │
│                              │                           │
│ ┌──────────┬──────────┐      │ ┌─────────────────────┐   │
│ │ Cam 1    │ Cam 2    │      │ │ Zone Statistics     │   │
│ │ Zone 1   │ Zone 2   │      │ │ Zone 1: 2 workers,  │   │
│ ├──────────┼──────────┤      │ │ 95% active          │   │
│ │ Cam 3    │ Cam 4    │      │ │ Zone 2: 1 worker,   │   │
│ │ Zone 3   │ Zone 4   │      │ │ 75% active          │   │
│ └──────────┴──────────┘      │ └─────────────────────┘   │
│                              │                           │
│                              │ ┌─────────────────────┐   │
│                              │ │ Active Alerts       │   │
│                              │ │ • Zone 2: High idle │   │
│                              │ │   time              │   │
│                              │ │ • Worker 5: Out of  │   │
│                              │ │   zone              │   │
│                              │ └─────────────────────┘   │
│                              │                           │
│                              │ ┌─────────────────────┐   │
│                              │ │ Index Progress      │   │
│                              │ │ [=========>    ] 65%│   │
│                              │ │ Time left: 20       │   │
│                              │ │ minutes             │   │
│                              │ └─────────────────────┘   │
└──────────────────────────────┴───────────────────────────┘

┌──────────────────────────────────────────────────────────┐
│ RAG Chat Interface                                       │
│ [User]: ทำไม zone 2 วันนี้ช้า                            │
│ [Claude]: วิเคราะห์แล้ว idle time สูงกว่าปกติ...         │
└──────────────────────────────────────────────────────────┘

Real-time Updates (WebSocket):
├─ Push tracking updates every 2 seconds
├─ Push alerts immediately
├─ Update charts every 5 seconds
└─ Sync index transitions
```


System Monitoring Dashboard:
├─ Health: CPU, GPU, memory, disk
├─ Cameras: Status, FPS, latency
├─ Detection: Inference time, accuracy
├─ Database: Connections, query time
└─ Error logs
PHASE 10.5: Analytics & Reporting Engine
├─ Real-time Metrics:
│ ├─ Productivity: active_time / (active + idle)
│ ├─ Efficiency: completed_tasks / scheduled_tasks
│ └─ Utilization: actual_workers / planned_workers
├─ Trend Analysis:
│ ├─ Daily: Today vs Yesterday, vs Last Week Same Day
│ ├─ Weekly: Productivity by day, identify problems
│ └─ Monthly: Overall performance, rankings
├─ Heatmap Visualization:
│ ├─ Spatial: Overlay on frame, color = time spent
│ └─ Temporal: Activity by hour, peak/low periods
├─ Predictive Analytics:
│ ├─ Forecast idle time
│ ├─ Predict bottlenecks
│ └─ Suggest worker assignments
└─ Report Generation:
├─ Scheduled: Daily(email 17:30), Weekly(PDF Mon 8AM),
Monthly(Excel with charts)
├─ On-demand: Custom date range, specific worker/zone
├─ Formats: PDF, Excel, CSV
└─ Contents: Summary, metrics, tables, charts, anomalies,
recommendations
PHASE 11: External Integration Layer
├─ REST API (FastAPI):
│ ├─ Endpoints:
│ GET /api/workers → List workers
│ GET /api/zones → List zones
│ GET /api/sessions/active → Current sessions
│ GET /api/metrics/today → Today's metrics


GET /api/alerts → Recent alerts
POST /api/workers → Add worker
POST /api/query → RAG query
Auth: API key (M2M), OAuth2 (users)
Rate limit: 100 req/min
WebSocket (Real-time):
ws://localhost:8000/ws/tracking
Events: zone updates, alerts, metrics
ERP/MES Integration:
Export: Time logs → Payroll, Productivity → Planning
Import: Work orders → Schedule, Shifts → Assignments
Methods: REST API, Message queue, DB replication
Export Functions:
Excel: openpyxl, charts, multi-sheet
CSV: Raw data for analysis
PDF: ReportLab, formatted reports
Webhook Integration:
Events: Alert, Index completed, Anomaly
Multiple destinations
Retry: 3 attempts with backoff