OMNIWATCH

Simulation Architecture & Methodology Plan

Real-Time AI Emergency Detection System

Google DevFest AI Hackathon · Washington University in St. Louis · 2026

February 25, 2026

"OmniWatch watches your cameras and listens to your building — and the moment someone is in danger, it finds the fastest way to get them help."

Contents

1 CCTV Dataset Sources 3
1.1 Fall Detection Datasets (Primary Scenario) . . . 3
1.2 Fight / Assault Detection Datasets (Secondary Scenario) . . . 3
1.3 General Surveillance Datasets (Background Feeds) . . . 3

2 Gemini API as the Detection Brain 4
2.1 The Gemini Vision Loop . . . 4
2.2 Prompt Structure (Conceptual) . . . 4
2.3 Audio Analysis via Gemini . . . 4
2.4 Why Live Gemini Detection Beats Pre-Scripted Boxes . . . 4
2.5 Rate Limiting Strategy . . . 5

3 Full Simulation Architecture 5
3.1 The State Machine . . . 5
3.2 Video Streaming Architecture . . . 5
3.2.1 Backend: FastAPI + WebSocket . . . 6
3.2.2 Why Canvas Instead of a Video Element . . . 6
3.2.3 Multiple Camera Feed Strategy . . . 6
3.3 Bounding Box Visual Design . . . 6

4 Dispatcher & Hospital Selection — Visual Flow 7
4.1 Step 1 — Scoring Animation (2 seconds) . . . 7
4.2 Step 2 — Selection Flash (0.5 seconds) . . . 7
4.3 Step 3 — Dispatcher Detail Card . . . 7
4.4 Step 4 — Map Activation . . . 7

5 Map Animation Methodology 7
5.1 Pre-stored Route Waypoints . . . 7

5.2 Dispatcher Vehicle Movement 7
5.3 Simultaneous ETA Visualisation 8
5.4 Distance Ring Animation 8

6 UI Layout for the Full Simulation **8**
6.1 Main Layout Structure 8
6.2 The Alert Moment — Full-Screen Response 8

7 Recommended Technology Stack **9**

8 Recommended Implementation Sequence **9**

9 What Makes This Demo Win **10**
9.1 The 10-Second Moment Is Everything 10
9.2 Gemini's Output Should Be Readable 10
9.3 The Map Tells the Story 10

1. CCTV Dataset Sources

1.1. Fall Detection Datasets (Primary Scenario)

- **URFD — University of Rochester Fall Detection Dataset**
  Available at `fenix.ur.edu.pl/mkepski/ds/uf.html`. Real RGB fall sequences comprising 70 clips, specifically designed for fall detection use cases. Free download, no registration required. This is the primary recommended dataset for the fall detection demo scenario.

- **Le2i Fall Detection Dataset**
  Available on Kaggle. Indoor CCTV-style footage shot from ceiling-mounted cameras, providing a very authentic surveillance aesthetic. Ground-truth fall labels are included.

- **OCCU Dataset**
  Covers elderly person fall scenarios. Importantly, it includes the moment of collapse and the motionless aftermath — precisely what the 8-second stillness detector (Equation 5 in the project paper) requires to validate.

1.2. Fight / Assault Detection Datasets (Secondary Scenario)

- **RWF-2000**
  Available on GitHub at `mchengny/RWF2000-Video-Database-for-Violence-Detection`. Contains 2,000 real CCTV clips in an exact 50/50 split between violent and normal footage. The low-quality surveillance aesthetic is highly realistic and visually convincing for a demo.

- **UCF-Crime Dataset**
  Available at `crcv.ucf.edu`. Contains 1,900 real surveillance videos spanning 13 anomaly categories including fighting and assault. This is considered the gold standard dataset for anomaly detection research.

1.3. General Surveillance Datasets (Background Feeds)

- **VIRAT Video Dataset**
  Available at `viratdata.org`. Outdoor surveillance footage with annotated pedestrians. Authentic parking lot and building lobby aesthetic, ideal for the "all-clear" background camera feeds.

- **MOT Benchmark**
  Available at `motchallenge.net`. Multi-person tracking sequences in urban environments matching the standard computer-vision detection aesthetic (coloured bounding boxes per class).

> **Recommended combination for the demo:** One URFD fall clip as the primary scenario trigger on CAM-01, one RWF-2000 clip on CAM-02 as a secondary feed, and VIRAT clips looping on CAM-03 and CAM-04. This produces visual variety across the four camera feeds and covers two distinct incident types.

2. Gemini API as the Detection Brain

This is the most important architectural decision: using the hackathon-provided Gemini API key as the live detection engine rather than pre-scripted bounding boxes. This makes the demo genuinely impressive and defensible to judges.

2.1. The Gemini Vision Loop

Every 2–3 seconds, the backend extracts a frame from the video stream and sends it to the Gemini 2.0 Flash multimodal API with a structured prompt. The prompt instructs Gemini to analyse the frame and return a structured JSON object containing:

- Whether any person is in a dangerous position
- The approximate bounding box coordinates of that person (expressed as percentages of frame width/height, making them resolution-independent)
- A confidence score between 0 and 1
- A plain-English description of what it observes, suitable for the incident report

The backend parses this JSON response and forwards it to the frontend via WebSocket. The frontend draws the bounding box on the canvas at the returned coordinates and updates the confidence meter accordingly.

2.2. Prompt Structure (Conceptual)

The prompt sent to Gemini consists of two parts: a system context establishing the AI as an emergency detection system, and a user message containing the base64-encoded frame image. The system context instructs Gemini to return only valid JSON with the following fields: `person_detected` (boolean), `danger_detected` (boolean), `danger_type` (string: fall / fight / motionless / none), `bounding_box` (object with x, y, width, height as percentage floats), `confidence` (float 0–1), and `description` (string, one sentence).

The JSON output mode available in Gemini 2.0 Flash makes this response format reliable and directly parseable without regex post-processing.

2.3. Audio Analysis via Gemini

If the CCTV footage has an audio track, short audio segments (approximately 2-second windows) can be sent to Gemini simultaneously with the visual frames. Gemini analyses the audio for distress signals including screams, calls for help, and impact sounds. The fused confidence score is then computed using Equation 7 from the project paper:

$$C_{\text{fused}} = \alpha \cdot C_{\text{vision}} + (1 - \alpha) \cdot C_{\text{audio}} \quad \text{where } \alpha = 0.65 \tag{1}$$

2.4. Why Live Gemini Detection Beats Pre-Scripted Boxes

When a judge asks “is this real AI or is it pre-programmed?” the team can answer with complete honesty: every bounding box coordinate and every confidence score is live output from Gemini 2.0 Flash analysing actual video frames in real time. That is a fundamentally stronger story than pre-computed detections, and it is verifiable on the spot.

2.5. Rate Limiting Strategy

At 2-second intervals across 4 camera feeds, the system would make approximately 2 API calls per second if all feeds ran at full resolution. The practical strategy is:

- Run active Gemini vision analysis only on the primary feed (CAM-01)
- Use lightweight local motion detection (frame differencing via OpenCV) on the three background feeds to decide when to escalate to a Gemini call
- Only escalate a background feed to Gemini if local motion detection flags unusual movement — this keeps API usage minimal while maintaining genuine coverage

3. Full Simulation Architecture

3.1. The State Machine

The entire simulation runs as a finite state machine. Every state transition has a specific, distinct visual representation in the UI so that anyone watching can track exactly where the system is in the detection-to-dispatch pipeline.

State	Description & UI Behaviour
IDLE	All 4 feeds playing normally. Metrics showing low baseline confidence scores. Map displays the SF hospital network with no active route. System status indicator: **GREEN**.
ANOMALY BUILDING (˜3 seconds)	Primary feed acquires a yellow border glow. Confidence meter begins rising. Gemini returns first elevated confidence score. Bounding box appears around the person in yellow.
DANGER CONFIRMED (˜7 seconds)	Bounding box turns **red** with pulsing glow animation. Confidence crosses threshold ($C_{\text{fused}} \geq 0.88$). Alert banner fires across top of UI. Audio alert plays. Priority badge switches to **CRITICAL**.
DISPATCHER CALLED (˜10 seconds)	Right panel animates in with selected dispatcher details. Hospital routing algorithm runs visibly. Map zooms to incident and hospital. Route polyline draws itself animated. Dispatcher vehicle icon appears and begins moving. ETA countdown starts.
IN TRANSIT	Dispatcher icon moves along route in real time. ETA counts down second by second. Incident panel shows live status updates. Other feeds continue playing.
ARRIVED	Dispatcher icon reaches incident location. Incident marked **RESOLVED**. Entry appended to incident history log. System returns to IDLE after 15 seconds.

Table 1: OmniWatch simulation state machine with UI behaviour per state.

3.2. Video Streaming Architecture

3.2.1. Backend: FastAPI + WebSocket

The backend stores video files locally (downloaded CCTV dataset clips). A FastAPI WebSocket endpoint opens when the frontend connects. The backend runs a continuous loop that:

1. Reads the video frame by frame using OpenCV
2. Encodes each frame as a base64 JPEG string
3. Bundles it with the latest detection metadata received from Gemini
4. Sends this combined JSON payload over the WebSocket at approximately 10 frames per second

Each WebSocket message payload contains conceptually: the base64 image string, an array of bounding box objects (each with pixel coordinates, class label, confidence score, and colour), the current system state name, and the latest metric values (C_v, C_a, C_{fused}).

3.2.2. Why Canvas Instead of a Video Element

A standard HTML `<video>` tag plays the file but provides no mechanism to: inject bounding boxes synchronised to specific frames, pause on a detection event, blend overlay graphics, or switch between camera feeds programmatically. An HTML5 `<canvas>` element provides complete pixel-level control. Every frame is a fresh draw operation, allowing bounding boxes, scan lines, confidence halos, and detection labels to be composited on top with arbitrary styling.

3.2.3. Multiple Camera Feed Strategy

The UI displays one large primary feed and three smaller thumbnail feeds. Each runs as either a separate WebSocket channel or a multiplexed sub-channel. The three background feeds play their dataset clips on a loop at a lower frame rate (5 fps is sufficient for thumbnails). Only the primary feed runs at full quality with active Gemini analysis.

3.3. Bounding Box Visual Design

The bounding box styling communicates system state without any text being necessary — a key advantage when presenting to an audience who may be watching from distance.

- **Normal state:** Thin magenta rectangles around all detected persons, matching the standard YOLO aesthetic. Corner tick marks rather than a full rectangle (higher-tech appearance). Small label above: `PERSON 0.94`.
- **Anomaly building:** The bounding box of the suspicious person turns yellow. A small animated icon appears in the corner. Label changes to: `ANOMALY DETECTED 0.71`.
- **Danger confirmed:** The box turns red with a CSS glow animation that pulses outward. The corners animate. Label changes to: ▷ `CRITICAL --- FALL`. A semi-transparent red overlay floods the region inside the bounding box.
- **Other objects:** Vehicles receive yellow boxes, traffic lights receive cyan boxes. This makes the feed visually resemble a full computer-vision system analysing the scene broadly, not a single-purpose fall detector.

4. Dispatcher & Hospital Selection — Visual Flow

This is one of the most important demo moments. When the DANGER CONFIRMED state triggers, the system does not immediately jump to "hospital selected." It shows the algorithm working step by step, which is considerably more impressive to judges than a result appearing instantaneously.

4.1. Step 1 — Scoring Animation (2 seconds)

The right panel shows all candidate hospitals appearing one by one, with their Equation 9 composite scores being calculated live. Each hospital entry appears with its distance, trauma level, and ETA populating in sequence:

$$h^* = \arg\min_{h_i \in H} \left[w_1 \cdot \mathrm{ETA}(h_i) + w_2 \cdot \frac{1}{\mathrm{TraumaLevel}(h_i)} + w_3 \cdot d(h_i) \right] \tag{2}$$

with defaults $w_1 = 0.5$, $w_2 = 0.3$, $w_3 = 0.2$.

4.2. Step 2 — Selection Flash (0.5 seconds)

The optimal hospital entry flashes green and expands. The others dim out. A ⋆ `OPTIMAL` badge appears next to the selected entry.

4.3. Step 3 — Dispatcher Detail Card

The full dispatcher card slides in showing: unit ID and dispatcher name, vehicle type (ambulance / fire / police depending on incident type), the selected hospital name and trauma level designation, and the initial ETA in minutes and seconds.

4.4. Step 4 — Map Activation

The Leaflet map zooms from the SF city overview down to the incident neighbourhood. The incident marker pulses red. The route polyline animates drawing itself from the hospital to the incident location over approximately 1.5 seconds, providing a visually satisfying confirmation of the routing decision.

5. Map Animation Methodology

5.1. Pre-stored Route Waypoints

Before the demo, the Google Maps Directions API (or OpenRouteService, which is free) is queried for the route from each major SF hospital to the chosen incident location. The returned polyline waypoints are saved as a static JSON file. During the live demo, no runtime API call is required — the system loads the pre-stored waypoints and animates along them. This eliminates any risk of API failure during the presentation.

5.2. Dispatcher Vehicle Movement

The dispatcher icon (an ambulance SVG marker) starts at the hospital location. Every second, the system calculates how far along the polyline the dispatcher should be, based on elapsed time divided by total ETA. It interpolates the marker position between waypoints and updates the Leaflet marker's latitude and longitude. This produces smooth, realistic movement along the actual road network rather than a straight-line approximation.

5.3. Simultaneous ETA Visualisation

Three elements update in parallel:

1. The dispatcher marker moves along the route polyline on the map
2. The ETA number in the sidebar counts down second by second
3. A progress bar in the incident card fills from left to right proportional to elapsed time

When ETA reaches zero, the marker animates onto the incident location and the state machine transitions to **ARRIVED**.

5.4. Distance Ring Animation

As the dispatcher closes in, an animated circle drawn on the map and centred on the incident location shrinks proportionally as the ETA decreases. This provides a visual representation of help approaching that reads clearly even from a distance. The circle changes colour from orange to green as it shrinks.

6. UI Layout for the Full Simulation

6.1. Main Layout Structure

```
+---------------------------------------------------------------+
|  HEADER: OMNIWATCH logo  |  SYSTEM ACTIVE  |  Clock | Status  |
+--------------------------------------+------------------------+
|                                      |  INCIDENT PANEL        |
|   PRIMARY CAMERA FEED (Canvas)       |  --------------------  |
|                                      |  Priority badge        |
|   [Bounding boxes drawn on top       |  Dispatcher details    |
|    of live video frames from         |  Hospital selected     |
|    WebSocket stream]                 |  ETA countdown timer   |
|                                      |  --------------------  |
|   +----------------------------+     |  LIVE MAP              |
|   | C_fused: [=====>    ] 0.87 |     |                        |
|   | Priority: [HIGH]           |     |  [Leaflet map with     |
|   +----------------------------+     |   route polyline +     |
|                                      |   moving dispatcher    |
|   +--------+--------+----------+     |   vehicle icon]        |
|   | CAM-02 | CAM-03 | CAM-04   |     |  --------------------  |
|   | ELEV   | PARK   | STAIRWELL|     |  METRICS               |
|   +--------+--------+----------+     |  Cv  |  Ca  | Cfused   |
+--------------------------------------+------------------------+
|  INCIDENT LOG -- scrolling event history feed                  |
+---------------------------------------------------------------+
```

6.2. The Alert Moment — Full-Screen Response

When CRITICAL fires, the entire UI responds simultaneously within 0.3 seconds:

- A full-screen **red flash** lasting 0.3 seconds
- The header bar turns red and remains red for the duration of the incident
- An alert banner slides down from the top containing the Gemini-generated incident summary in plain English
- The primary camera feed acquires a persistent red border glow
- An audio alert sound plays once

All of this happens simultaneously in under half a second, creating a visceral signal that reads clearly to judges watching from across the room.

7. Recommended Technology Stack

Layer	Tool	Rationale
AI Core	Google Gemini 2.0 Flash	Multimodal vision + audio + agentic reasoning; hackathon API key
Video Storage	MP4 files via FastAPI static dir	Simple, no cloud dependency, works offline
Frame Streaming	FastAPI WebSocket	Bidirectional channel, low latency, carries both frames and metadata
Detection	Gemini vision loop	Live, genuine AI output; defensible to judges
Frontend Video	HTML5 Canvas	Full overlay control; bounding boxes, scan lines, alert halos
Map	Leaflet.js + CartoDB dark tiles	No API key required at runtime; already in project
Route Animation	Pre-stored JSON waypoints + JS interpolation	Zero runtime API dependency; guaranteed reliability
State Management	JavaScript state machine	Lightweight; no React needed for this layer

Table 2: Recommended technology stack for the OmniWatch simulation demo.

8. Recommended Implementation Sequence

1. **Phase 1 — Video playing via WebSocket.**
   This is the entire foundation. Nothing else matters until a video frame is rendering in a canvas element in the browser via a backend WebSocket connection.

2. **Phase 2 — Gemini detection loop.**
   Wire frame extraction to Gemini API calls. Receive bounding box JSON. Draw boxes on canvas. Verify coordinate accuracy against the visible frame content.

3. **Phase 3 — State machine.**
   Implement the full IDLE → ANOMALY → DANGER → DISPATCHED → ARRIVED → RESET cycle. Trigger transitions based on Gemini confidence scores crossing thresholds defined in Equation 8.

4. **Phase 4 — Dispatcher UI.**
   Build the incident panel, hospital scoring animation, dispatcher details card, and ETA countdown. The scoring animation (Step 1 in Section 4) should be built and timed carefully — this is a key judge-facing moment.

5. **Phase 5 — Map animation.**
   Load pre-stored route waypoints. Animate the dispatcher marker. Synchronise the marker position to the ETA countdown timer.

6. **Phase 6 — UI polish.**
   Apply the alert flash, bounding box animations, confidence bar transitions, and incident log styling. This is where the visual design quality is established.

7. **Phase 7 — End-to-end rehearsal.**
   Run the complete simulation from IDLE to RESOLVED ten consecutive times. Identify every timing issue, visual glitch, and race condition. The demo must be deterministic and reliable under presentation conditions.

9. What Makes This Demo Win

Three things separate a hackathon demo that wins from one that merely works.

9.1. The 10-Second Moment Is Everything

From the frame where the fall is detected to the moment the dispatcher card appears with ETA should be exactly 10 seconds visible on screen. Time it precisely. Make it feel urgent. This is the entire value proposition of OmniWatch compressed into one visible, measurable moment. The survival probability equation makes the stakes explicit:

$$\Delta P_{\text{survival}} = P_0 \left(e^{-\lambda t_{\text{OmniWatch}}} - e^{-\lambda t_{\text{traditional}}} \right) \approx +41 \text{ percentage points} \tag{3}$$

9.2. Gemini's Output Should Be Readable

Show the raw Gemini response briefly on screen — the plain-English incident description it generates (e.g., *"Elderly person appears to have fallen near elevator bank, motionless for 8+ seconds, possible cardiac event"*) — before converting it to structured data and updating the metrics. Judges want to see the AI thinking, not just the output of a pipeline.

9.3. The Map Tells the Story

When the ambulance icon begins moving from UCSF Medical Center toward the incident marker, every person in the room understands what OmniWatch does without any narration required. That moving icon is the entire pitch made visual. Invest the time to make the movement smooth, the route accurate to real SF streets, and the ETA countdown credible.

Core principle: The demo should function as a standalone pitch. If a judge watches the simulation from start to finish without hearing a single word of explanation, they should be able to describe OmniWatch's purpose, its AI pipeline, and its impact on emergency response time. Design every UI element toward that goal.