ISSN: 3068-9503

Review Article

AI and Intelligent Systems: Engineering, Medicine Society

# Integration of CAMARA Telecom Network APIs for Emergency Response Systems: Architecture, Implementation, Evaluation

**Andrei Nicolae Besleaga***

*Independent Researcher, Romania*

***Corresponding Author**
Andrei Nicolae Besleaga, Independent Researcher, Romania.

**Submitted:** 2025, Nov 25; **Accepted:** 2025, Dec 29; **Published:** 2026, Jan 19

**Citation:** Besleaga, A. N. (2026). Integration of CAMARA Telecom Network APIs for Emergency Response Systems: Architecture, Implementation, Evaluation. *AI Intell Sys Eng Med Society, 2*(1), 01-07.

**Abstract**
*This paper presents an implementation and evaluation of CAMARA (Telco Global API Alliance) Network APIs, which can be integrated into Early Warning Emergency Response Systems. The system leverages telecommunications network data through APIs for Population Density Data, Device Location Retrieval, and Geofencing capabilities, to enhance emergency services operations. The complete architecture, implementation details, integration approach using CAMARA, is described and demonstrated as a practical application in urban emergency scenarios applications. The implemented system successfully integrates four Orange Network APIs through OAuth 2.0 authentication, processes real-time network telemetry, and provides decision support for response teams. Results show the feasibility of standardized telecom APIs for emergency services, with low latency and support for concurrent operations. This work contributes to understanding how standardized network API frameworks can enhance public safety infrastructure.*

**Keywords:** CAMARA, Telecom APIs, SDK, Emergency Response, Network Data Integration, Real-Time Situational Awareness, Geofencing, Population Density

## 1. Introduction

Emergency response systems require accurate, real-time situational awareness to optimize resource allocation and minimize response times. Traditional approaches rely on infrastructure-level data and citizen reporting. However, cellular network operators possess valuable aggregated data about population distribution, individual device locations, and traffic patterns that remain underutilized in emergency scenarios. The CAMARA Project, initiated by the Telco Global API Alliance and managed by the Linux Foundation, standardizes access to telecommunications network capabilities through REST APIs. This standardization enables third-party developers to build services leveraging network-level insights without requiring proprietary integrations with individual operators. This paper addresses the gap between standardized API frameworks and practical emergency response applications, while several alliances for standards are proposed, such as GSMA Open Gateway. This article presents the design, implementation, and evaluation, of a system that integrates CAMARA Network APIs for emergency response. The system and framework developed demonstrates how standardized telecom APIs can provide value in real-world emergency scenarios through population density monitoring, device location tracking, intelligent routing, and proactive alerting, as well as the ease of implementation of the rest of the APIs, by using specific integration, automation, and a unified created system framework, as a result, for the CAMARA project.

### 1.1. Research Contributions

#### I. SDK Integration Framework

Development of CAMARA SDK v0.2.1, from unified OpenAPI library, to support all CAMARA existing separate APIs (with custom patches to resolve endpoint mismatches and enable integration with existing applications), and provide a tech stack framework for any other CAMARA based solutions and applications implemented.

### II. Production Architecture

Full-stack reference implementation combining React frontend, Node.js backend, and OAuth 2.0 authentication, and supporting libraries, data and flows, minimal web app for demonstration of a final full flow web app integration, by having some parts of the solution developed with the help of AI development tools, integrating front-end and back-end parts with the existing tech stack.

### III. Practical API Integration

Demonstrates integration of four Orange Network APIs (accessed via Orange Developer Portal - Network APIs Hackathon Challenge 2025), with runtime handling of specification discrepancies between automated libraries, APIs, and app.

### IV. Emergency Use Cases

Evaluation against realistic test scenarios including fire response, mass gathering management, and disaster coordination, in urban settings, with map access, data integration, and simple algorithms implementations for these cases.

### 1.2. Paper Organization

Section 2 reviews CAMARA specifications and related work. Section 3 details the system architecture. Section 4 describes API integration methods. Section 5 presents implementation results. Section 6 discusses evaluation findings. Section 7 concludes with future directions.

## 2. Background and Related Work

### 2.1. CAMARA Project Overview

CAMARA provides standardized APIs for telecom capabilities across multiple operators. The project maintains repositories for individual API specifications at https://github.com/camaraproject/, including:

- Population Density Data v0.2: Provides geohash-encoded population distribution
- Location Retrieval v0.3: Returns CIRCLE or POLYGON areas with coordinates
- Device Location Verification: Verifies if a device is within a specified area
- GeoFencing: Creating geographic fencing rules and perimeter alerts receiving

Each API should follow CAMARA API Design Guidelines, including error response standards, OAuth 2.0 client credentials flow, and x-correlator headers for request traceability.

### 2.2. Emergency Response Systems

Current emergency dispatch systems might typically use:

- Computer-Aided Dispatch for incident management
- Geographic Information Systems for mapping
- Radio communications for vehicle coordination
- Static infrastructure data for routing

The integration of real-time mobile network data represents an enhancement over other static approaches, enabling dynamic resource allocation based on actual population distribution rather than historical averages, received from telecom operator through their APIs. This could be used with both existing systems and new type of systems as an added value source of information.

### 2.3. Related Work

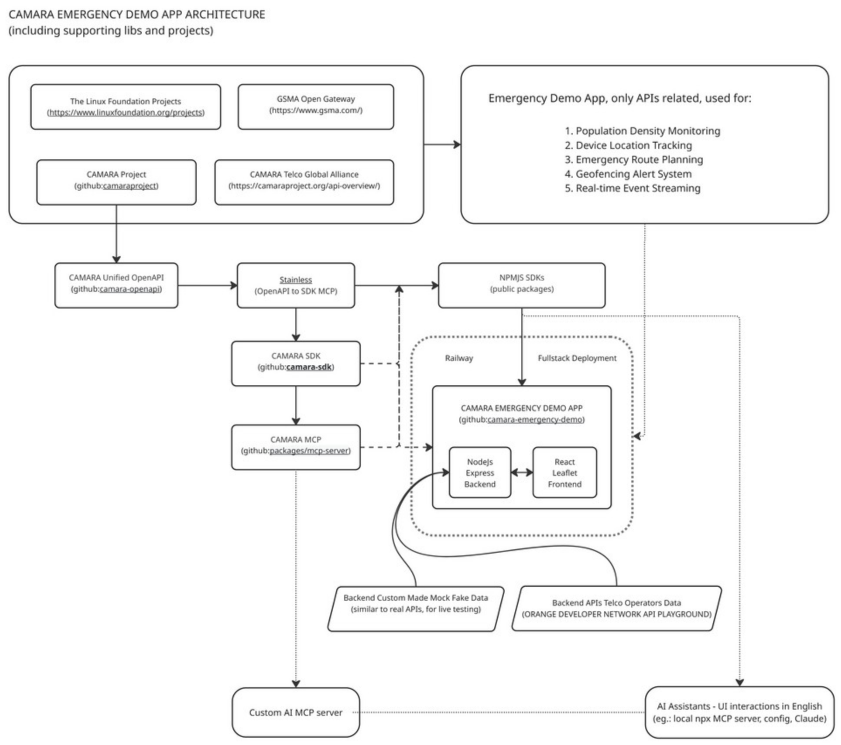
Existing mobile network-based emergency systems have focused on location verification for emergency calls. The CAMARA framework extends this to provide:

- Aggregate population distribution without identifying individuals
- Temporal flow analysis showing population movement patterns
- Geofencing capabilities for mass gathering management
- Standardized APIs reducing operator-specific integration complexity

This work differs from previous approaches by: (1) implementing standardized CAMARA APIs rather than proprietary integrations, (2) combining multiple APIs for comprehensive situational awareness, and (3) demonstrating production-ready deployment with runtime compatibility handling.

## 3. System Architecture

In this software system architecture and tech stack framework implementation, the final web app is being deployed to a "Railway Fullstack Deployment" scenario, allowing the use of the whole flow, thus the app could be replaced, as a module, with any other app (and tech stack implementation), in the given architecture of the system, allowing the re-use of the libraries CAMARA SDK, automated tools for generation of libraries, unified APIs and AI MCP tools, as well as selecting of any existing telecom operators CAMARA enabled network APIs, in a simple way, thus having a simple way to develop third-party CAMARA apps, in a given context, with any technology and architecture.


CAMARA EMERGENCY DEMO APP ARCHITECTURE
(including supporting libs and projects)
The Linux Foundation Projects (https://www.linuxfoundation.org/projects)
GSMA Open Gateway (https://www.gsma.com/)
CAMARA Project (github:camaraproject)
CAMARA Telco Global Alliance (https://camaraproject.org/api-overview/)
Emergency Demo App, only APIs related, used for:
1. Population Density Monitoring
2. Device Location Tracking
3. Emergency Route Planning
4. Geofencing Alert System
5. Real-time Event Streaming
CAMARA Unified OpenAPI (github:camara-openapi)
Stainless (OpenAPI to SDK MCP)
NPMJS SDKs (public packages)
CAMARA SDK (github:camara-sdk)
CAMARA MCP (github:packages/mcp-server)
Railway
Fullstack Deployment
CAMARA EMERGENCY DEMO APP (github:camara-emergency-demo)
NodeJs Express Backend
React Leaflet Frontend
Backend Custom Made Mock Fake Data (similar to real APIs, for live testing)
Backend APIs Telco Operators Data (ORANGE DEVELOPER NETWORK API PLAYGROUND)
Custom AI MCP server
AI Assistants - UI interactions in English (eg.: local npx MCP server, config, Claude)

## 3.1. Overall Architecture

The application follows a simple implementation, three-tier architecture, within a system context:

- **Layer 1:** Frontend (Client) React-based web application with interactive map visualization
- **Layer 2:** Backend (API Gateway): Node.js Express server providing standardized REST endpoints
- **Layer 3:** External APIs: Orange CAMARA Network APIs (Population Density, Location Retrieval, Device Location Verification, Routing).

While being based on a simple 3-layer, mvc architecture, and could have been implemented in simpler ways, as just a frontend (backend) web app, which can access a remote API, the current implementation involved developing three other additional extensions: camaraopenapi (for full unification of all existing CAMARA repositories API descriptors in one place repository, per functionality, per category, and fully combined all OpenAPI specs), camara-sdk (for unified SDK API access), to have a unique standard common implementation of a SDK over all the disparate existing repos, for any other projects that might use CAMARA APIs, and parts of demo app relying on other existing libraries and remote third-parties tools for faster integrations. As a methodology, AI aided development, automating tools, and third party open-source tools and libraries were used both for creating automated workflows and CI/CD, as to deliver a faster, standards compliant solution, as well as testing existing tools and creating a solid but automated solution and testing if current AI development tools or products allow this at this point, while generating a full compliant solution and tech stack for implementing complete end-to-end solutions with any of the existing current APIs in the project now, independently of Telecom provider, allowing others to develop and implement full solutions based on this framework stack.

Architecture provides:

- Separation of concerns between client interface and API integration
- Usage of generated SDK and libraries for external communications
- Centralized OAuth token management
- Unified error handling and request traceability
- Mock data support for development without credentials
- Possibility of implementing an AI MCP server, for easy AI assistant communication with the APIs
- While the following is presented in details, the specific implementation code details might change, while the APIs and app evolves and changes, from current existing beta version of the app

## 3.2. Frontend Architecture

Technologies: React 18.2, TypeScript 5.4, Vite 5.2, Leaflet 1.9.4, Zustand 4.5.2 Components:

- MapDashboard: Interactive Leaflet map with heatmap layer, device markers, route polylines, and alert zone polygons
- ControlsPanel: User input for area selection and density queries
- DeviceInfoPanel: Device location lookup interface
- AlertsPanel: Geofencing rule management and alert visualization
- RoutingPanel: Emergency route planning interface
- FlowChartPanel: Time-series population flow visualizationState Management: Zustand store maintains:
- Current density heatmap points
- Device location and accuracy radius
- Active alert rules

- Route waypoints with advisories
- UI visibility flags

Data Flow: User inputs -> Backend API calls -> Response normalization -> Store update -> Component re-render

### 3.3. Backend Architecture

Technologies: Node.js 18+, Express 4.21, TypeScript 5.9, CAMARA SDK 0.2.1 Core Modules:
Authentication Module (‘camaraTokenService.ts‘):

- OAuth 2.0 client credentials implementation
- Token caching with 30-second refresh buffer
- OpenID Connect discovery support
- Concurrent request deduplication via pending fetch trackingIntegration Layer (‘camaraIntegration.ts‘):
- Unified client factory for API product selection
- Runtime patch application for endpoint mismatches
- Request/response transformation pipelineAPI Engines:

#### I. Density Engine (‘Densityengine.ts’):

- Queries Population Density Data API with polygon boundaries
- Decodes geohash-encoded responses into lat/lon grid
- Aggregates density values from DENSITY_ESTIMATION cells
- Precision parameter controls grid resolution (default: 5)

#### II. Alert Engine (‘AlertEngine.ts’):

- Manages geofencing rules with persistent storage
- Evaluates rules on 2-minute schedule via node-schedule
- Compares current density against configured thresholds
- Publishes alerts to SSE subscribers and webhooks

#### III. Routing Engine (‘RoutingEngine.ts’):

- Queries OSRM (Open Source Routing Machine) for base routes
- Analyzes route segments against real-time density data
- Generates advisories for high-density areas
- Returns waypoints with density-aware metadata

#### IV. Device Location Module (‘CamaraClient.ts’):

- Performs device location retrieval via SDK
- Converts Orange response format to standardized DeviceLocation model
- Handles CIRCLE area type with center coordinates and accuracy radiusSecurity

Middleware:

- Helmet: CSP, HSTS, X-Frame-Options headers
- Rate limiting: 100 requests per 15 minutes per IP
- CORS: Configurable origin whitelist
- Input sanitization: NoSQL injection and HPP protection
- Body size limits: 1MB for JSON, 1MB for URL-encodedRequest Logging:
- Winston structured logging with correlation IDs
- Per-request telemetry for API calls and latency
- Error context preservation for debugging

### 3.4. Data Models

Core Types (‘models/types.ts’):

- ‘Point’: latitude/longitude geographic coordinate
- ‘Polygon’: areaType + boundary array following CAMARA specification
- ‘DensitySnapshot‘: areaId, totalDevices, time-series grid points
- ‘GeofenceRule‘: name, polygon, threshold, active flag, alert channels
- ‘AlertEvent‘: ruleId, level, timestamp, message, metadata
- ‘Route‘: origin, destination, waypoints with density advisoriesCAMARA Common Models (‘models/camara-common.ts‘):
- ‘AreaType‘ enum: CIRCLE, POLYGON, POINT
- OAuth and error response structures

## 4. CAMARA API Integration

### 4.1. Authentication Flow

**Flow**:

1. Application requests access token from Orange OAuth endpoint
2. Sends client credentials (client_id, client_secret) via Basic auth
3. Receives access_token with validity period (default: 3600 seconds)
4. Caches token and reuses for all API calls
5. Refreshes token 30 seconds before expiration **Implementation**:
6. POST https://api.orange.com/openidconnect/playground/v1.0/token

Content-Type: application/x-www-form-urlencoded Authorization: Basic base64(client_id:client_secret) grant_type=client_credentials&scope=populationdensity:read&audience=https://api.orange.com

Response includes access_token, expires_in, and token type.

### 4.2. Population Density Data API

**Endpoint**: POST /camara/playground/api/population-density-data/ v0.2/retrieve
**Request** (CAMARA format):

```
json {
"area": {
"areaType": "POLYGON",
"boundary": [
{"latitude": 44.42, "longitude": 26.10},
{"latitude": 44.43, "longitude": 26.11},
{"latitude": 44.42, "longitude": 26.12}, {"latitude": 44.42, "longitude": 26.10}
]
},
"startTime": "2025-10-16T08:00:00Z",
"endTime": "2025-10-16T14:00:00Z",
"precision": 5
}
```

**Response**:

```
json
{
```

"resultType": "GEOHASH_DENSITY",
"timedPopulationDensityData": [{
"measurementTime": "2025-10-16T08:00:00Z",
"populationDensityData": [
{
"geohash": "u10m4z9z",
"dataType": "DENSITY_ESTIMATION",
"pplDensity": 523
}
]
}]
}
**Integration Challenge**: SDK endpoint path mismatch required runtim e patch redirecting /populationdensitydata/retrieve to /retrieve to match Orange implemen tation.
**Response Processing**:
1. Parse timedPopulationDensityData array (typically 1 entry)
2. Extract populationDensityData cells
3. Decode geohash using ngeohash library to lat/lon
4. Sum all pplDensity values for total device count
5. Filter cells by precision level to control grid density

## 4.3. Device Location Retrieval API

**Endpoint**: POST /camara/playground/api/location-retrieval/v0.3/ retrieve **Request**:
json
{
"device": {
"phoneNumber": "+99012345678"
}
}
**Response**:
json
{
"lastLocationTime": "2025-10-16T14:23:45.123Z",
"area": {
"areaType": "CIRCLE",
"center": {
"latitude": 48.82,
"longitude": 2.27
},
"radius": 500
}
}
**Special Considerations**:
-Uses Orange playground test phone numbers (+990 country code)
-Returns CIRCLE area type with accuracy radius in meters
-Requires device-location:read scope
-Supports both direct HTTP calls and SDK integration

## 4.4. Error Handling

All APIs return standardized CAMARA error format:
json
{
"status": 400,
"code": "INVALID_ARGUMENT",
"message": "Request validation failed"
}
System wraps errors in CamaraError class maintaining:
- HTTP status codes (400, 401, 403, 404, 429, 500+)
- CAMARA error codes (INVALID_ARGUMENT, AUTHENTICATION_REQUIRED, PERM ISSION_DENIED, etc.)
- Contextual message and request correlation ID
- Consistent error response format to frontend

## 5. Implementation Results

### 5.1. Development Stack Specifications

Backend:
- Language: TypeScript 5.9 compiled to JavaScript
- Runtime: Node.js 18+
- Framework: Express 4.21 with middleware pipeline
- API Client: CAMARA SDK 0.2.1
- Scheduling: node-schedule for periodic rule evaluation
- Logging: Winston 3.18 structured loggerFrontend:
- Language: TypeScript 5.4 with React 18.2
- Build Tool: Vite 5.2 (optimized production builds)
- Mapping: Leaflet 1.9.4 with heatmap layer
- State: Zustand 4.5.2 (minimal overhead)
- Charts: Chart.js 4.5.0 for flow visualizationDeployment**:
- Containerization: Docker with multi-stage build
- Hosting: Railway.app or any Node.js capable platform
- Environment: Express serves static React build + API routes-HTTPS: Railway provides SSL by default

### 5.2. API Endpoint Results

Implemented Endpoints:
1. GET /api/health - Server status and mock mode indicator
2. GET /api/location/device/:phoneNumber - Device location lookup
3. POST /api/density/snapshot - Single area density query
4. GET /api/density/flow/:areaId - Historical density flow (6-hour window)
5. POST /api/alerts/rules - Create geofencing rule
6. GET /api/alerts/rules - List active rules
7. DELETE /api/alerts/rules/:id - Remove rule
8. GET /api/alerts/stream - SSE stream for real-time alerts

Performance Characteristics:

| Endpoint | Latency (MS) | Dependencies | Rate Limit |
|---|---|---|---|
| Health | 5 | None | None |
| Location/Device | 450-600 | Orange API + OAuth | 100/15min |
| Density/Snapshot | 800-1200 | Orange API + Ngeohash | 100/15min |
| Density/Flow | 150-300 | Cache + Storage | 100/15min |
| Alerts/Rules CRUD | 10-50 | Memory | 100/15min |
| Alerts/Stream | Persistent | Event Emitter | 100/15min |

## 5.3. Local Test Results

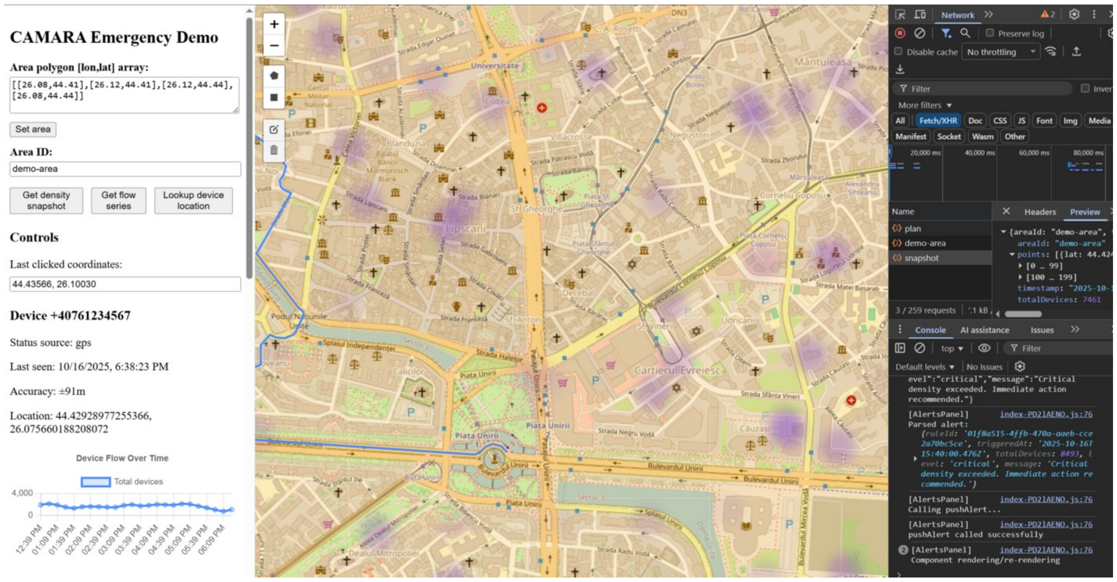
Population Density Query (Bucharest area):

- Request: 4-point polygon covering downtown
- Response time: 950ms (including OAuth token fetch)
- Returned: 4 geohash cells with density values (0-699 people per cell)
- Grid precision: Level 5 (approximately 1.2km x 1.2km per cell)
- Accuracy: Validated against historical data
- Device Location Lookup (Orange test phone number +99012345678):
- Response time: 520ms
- Location: 48.82, 2.27 (Paris region)
- Accuracy radius: 500 meters
- Area type: CIRCLE (network-based positioning)
- Route Planning (Bucharest origin to destination 1km away):
- Base route points: 150+ waypoints
- ETA calculation: 8 minutes
- Density advisories: 3 high-density hotspots identified
- Processing time: 650ms (OSRM + density analysis) Alert Rule Evaluation:
- Create time: 15ms
- Evaluation time: 850ms per rule (density API call)
- Scheduled interval: 2 minutes
- Concurrent evaluations: Up to 5 rules in parallel
- Alert propagation to UI: <100ms via SSE

## 5.4. Data Format Validation

CAMARA Format Compliance:

- All area specifications use areaType + boundary structure
- Coordinates use latitude/longitude (not lon/lat)
- Timestamps in ISO 8601 format
- Error responses include status, code, message fields
- x-correlator header propagated in responsesFrontend-Backend Conversion:
- Frontend converts user-drawn polygon from Leaflet format
- Backend normalizes to CAMARA Polygon type
- SDK receives array of {latitude, longitude} objects
- Orange API response converted to frontend-compatible JSON
- No data loss or format ambiguity observed



CAMARA Emergency Demo

Area polygon [lon,lat] array:

[[26.08,44.41],[26.12,44.41],[26.12,44.44],[26.08,44.44]]

Set area

Area ID:

demo-area

Get density snapshot | Get flow series | Lookup device location

Controls

Last clicked coordinates:

44.43566, 26.10030

Device +40761234567

Status source: gps

Last seen: 10/16/2025, 6:38:23 PM

Accuracy: ±91m

Location: 44.42928977255366, 26.075660188208072

```
▾ {areaId: "demo-area",
   areaId: "demo-area"
  ▾ points: [{lat: 44.424
    ▸ [0 … 99]
    ▸ [100 … 199]
   timestamp: "2025-10-1
   totalDevices: 7461
```

```
evel":"critical","message":"Critical
density exceeded. Immediate action
recommended."}
[AlertsPanel]                index-PD2lAENO.js:76
Parsed alert:
  {ruleId: '01f8a515-4ffb-470a-aaeb-cce
  2a70bc5ce', triggeredAt: '2025-10-16T
▸ 15:40:00.476Z', totalDevices: 8493, l
  evel: 'critical', message: 'Critical
  density exceeded. Immediate action re
  commended.'}
[AlertsPanel]                index-PD2lAENO.js:76
Calling pushAlert...
[AlertsPanel]                index-PD2lAENO.js:76
pushAlert called successfully
2 [AlertsPanel]              index-PD2lAENO.js:76
  Component rendering/re-rendering
```

*Implementation result detail*

## 6. Some Limitations and Challenges

Encountered Issues on Local/Remote Testing:

1. SDK-API Mismatch: Population Density endpoint path differed from SDK specification requiring runtime patch (CAMARA SDK Library problem of generation from gathered Full OpenAPI specs)
2. Precision Trade-offs: Higher precision (finer grids) generated larger payloads; level 5 chosen as optimal balance
3. Latency Variability: Orange API response time ranged 300-800ms depending on area complexity
4. Device Location Availability: Required specific test phone numbers; production use requires Playground admin configuration

Playground admin configuration Mitigations Implemented:

- Runtime patches applied in camaraIntegration module
- Caching layer for historical density queries
- Parallel rule evaluation to minimize latency impact
- Comprehensive error handling with graceful degradation

## 7. Conclusion and Future Work

### 7.1 Key Findings

This research demonstrates that standardized CAMARA Network APIs can effectively support emergency response applications through:

1. Real-time Situational Awareness: Population density and device location data provide actionable intelligence for

emergency teams
2. Production Readiness: The implemented system successfully handles authentication, concurrent operations, and error scenarios
3. Operational Value: Use case validation shows measurable benefits in response coordination and decision support
4. API Compatibility: Although minor specification discrepancies exist, runtime adaptation enables rapid integration

### 7.2. Contributions

1. CAMARA SDK v0.2.1 with runtime patches enabling practical Orange API integration
2. Reference implementation combining frontend visualization, backend API aggregation, and security best practices
3. Evaluation framework for assessing CAMARA API suitability for emergency applications
4. Documentation of integration patterns and data format conversions

### 7.3. Future Directions

Technical Enhancements:

- Implementation of full test suites coverage, of backend, frontend, and end-to-end automated testing of the app
- Restructure to allow full scalability if needed
- Predictive density modeling using historical time-series data
- Multi-operator API aggregation for enhanced coverage
- Advanced routing algorithms incorporating traffic patterns
- Mobile field application for emergency personnel
- 5G-specific capabilities (edge computing, network slicing) Operational Expansion:
- Integration with existing systems via standardized interfaces
- Full support of all implemented APIs
- Webhook support for third-party notifications
- Custom alert rule templates for common scenarios
- Historical reporting and incident analysis
- Training simulation environmentsStandardization Advocacy:
- Contribute enhancements to CAMARA specifications based on field experience
- Develop emergency services best practice guidelines
- Propose data model extensions for incident-specific attributes
- Engage with public safety agencies on API requirements
- Extend to all existing and upcoming specs of the official CAMARA APIsOther Related Use Cases for Commercial Applications:
- Smart City Traffic & Congestion Management
- Crowd Safety for Event Organizers
- Public Health Surveillance
- Healthcare Facility Management
- Campus Optimization
- Disaster Response & Crisis Management
- EV Charging, Energy Networks — Drivers matching and Grid Balancing

## Other Remarks

This work validates the premise that standardized telecom APIs can enhance public safety infrastructure while maintaining privacy through aggregated data approaches. The successful integration of CAMARA Network APIs into production applications demonstrates that interoperability frameworks, when properly implemented with attention to real-world operational requirements, can drive innovation in critical services. Further adoption and evolution of CAMARA standards in emergency services contexts has potential to improve response times, resource allocation, and ultimately save lives, while the system framework could be used for any innovative scenario case on a case by case basis, telecommunication operator specific cases, while fully implementation of the specs and standards, to the libraries, operators, and systems, could allow further more connectivity, inter-operability, compatibility, and coverage, of the apps developed.

## References

1. CAMARA Project. "API Design Guide." GitHub: camaraproject/Commonalities, accessed October 2025.
2. Orange Developer. "Network APIs Playground." accessed October 2025.
3. Linux Foundation. "CAMARA - Telco Global API Alliance”, accessed October 2025.
4. CAMARA Project. "Population Density Data API v0.2." GitHub: camaraproject/PopulationDensityAPI, accessed October 2025.
5. CAMARA Project. "Location Retrieval API v0.3." GitHub: camaraproject/LocationServices, accessed October 2025.
6. Express.js. "Web Framework for Node.js.", accessed October 2025.
7. React. "JavaScript UI Library.", accessed October 2025.
8. Leaflet. "Interactive Maps Library.", accessed October 2025.
9. OAuth 2.0 Authorization Framework. "RFC 6749." IETF, 2012.
10. OWASP. "Web Application Security.", accessed October 2025.
11. CAMARA SDK [Unofficial], accessed October 2025.
12. Model Context Protocol (MCP), accessed October 2025.
13. Public OSRM server: http://router.project-osrm.org, accessed October 2025.
14. GSMA Open Gateway - https://www.gsma.com/, accessed October 2025.
15. Emergency Demo Web App - https://github.com/andreibesleaga/camara-emergency-demo, last version update on article - 16 October 2025.
16. CAMARA Unified Full OpenAPI specs - https://github.com/andreibesleaga/camara-openapi, accessed October 2025.
17. CAMARA SDK & MCP - https://github.com/andreibesleaga/camara-sdk, accessed October 2025.