iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** V **Month of publication:** May 2026

DOI: https://doi.org/10.22214/ijraset.2026.82030

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue V May 2026- Available at www.ijraset.com

Automated Emergency Response System

Shriya Malode[1], Pranav Pawar[2], Dr. M. P. Sardey[3]
[1, 2]*Department of Electronics and Telecommunication Engineering, AISSMS Institute of Information Technology, Pune, India*
[3]*Project Guide, AISSMS Institute of Information Technology, Pune, India*

Abstract: ***Road accidents remain the primary global cause of death since emergency responders take too long to reach accident sites which results in unnecessary fatalities. This paper presents the Automated Emergency Response System (AERS), a novel AI-powered, IoT-integrated platform that fully auto- mates the accident detection-to-dispatch pipeline. The system combines two independent detection systems which include (1) a computer vision pipeline that uses a fine-tuned MobileNetV2 deep neural network to detect accidents through real time video monitoring and (2) an ESP32 micro-controller which uses an MPU6050 inertial measurement unit (IMU) to detect physical crashes through accelerometer and gyroscope threshold measurements. The system uses Haversine distance algorithm for backend detection to find the closest ambulance while the system uses OSRM API to determine the driving path and it communicates with traffic lights along the path and it sends GPS-enabled SMS notifications to drivers through Twilio and it creates a real-time command center dashboard. The system provides deployment options for both local use and cloud services through Render.com while it connects with seven different APIs to provide geolocation services and routing functionalities and mapping capabilities and classification services. The experimental results show that accident detection achieves reliable performance through a 15-frame consensus method which eliminates false positives and the system enables dispatch within one second while using OpenStreetMap's Overpass API to find operational resources. The proposed architecture enables emergency dispatch systems to operate with less human control while providing a scalable emergency management solution through an open source platform for smart city environments.***

Index Terms: ***Emergency Response System, Accident Detec- tion, MobileNetV2, IoT, ESP32, MPU6050, Transfer Learning, Ambulance Dispatch, Smart City, Real-time Monitoring.***

I. INTRODUCTION

Road traffic accidents represent one of the most critical public health challenges of the modern era. The World Health Organization (WHO) reports that road traffic accidents result in approximately 1.35 million annual deaths while millions of people sustain non-fatal injuries [1]. The emergency response system in developing countries experiences high fatality rates because of its inability to deliver timely medical assistance. A reduction in response time by even a few minutes can be the difference between life and death in trauma cases. The standard emergency response system depends on three manual processes which require people to call emergency numbers, operators to coordinate ambulance dispatch, and drivers to find their way to emergency situations. The procedure requires a long time to complete because it depends on witnesses who need to be awake and alert to see what happened.

The existing situation creates urgent requirements for a system which can automatically identify accidents, locate local resources, and send emergency assistance without waiting for human start-up. Recent advancements in deep learning, IoT hardware, and cloud computing offer a unique opportunity to build such systems. Computer vision models can now detect accidents from video feeds in real time, and low-cost microcontrollers such as the ESP32 can sense physical crash signatures using inertial measurement units (IMUs). The com- bined technologies create a detection system which delivers both strong detection capabilities and built-in redundancy.

This paper makes the following contributions:

1) The system detects accidents through two different meth- ods which use computer vision technology based on MobileNetV2 and physical crash sensors that use both ESP32 and MPU6050 components.
2) The system dispatches ambulances through a complete automated backend system which handles nearest am- bulance selection and optimal route determination and traffic signal pre-emption alerts and SMS notification delivery within one operational process.
3) The project develops a command center with live ambu- lance tracking that operates as a web-based control room. The system uses Flask and Leaflet.js and OSRMfor its operational framework.
4) Supports installation on premises as well as Render.com.

5) The system uses OpenStreetMap Overpass API as its dy- namic resource discovery method to obtain actual hospital and traffic signal data which it retrieves through current GPS location.

The remainder of this paper is organized as follows. The related work section of the paper begins in Section II. The complete system design is explained in Section III. The AI/ML detection pipeline operates according to the specifications in Section IV. The IoT hardware module operates according to the description in Section V. The backend dispatch system together with its Application Programming Interfaces (APIs) operates according to design in Section VI. The user interfaces together with their frontend systems are presented in Section VII. The experimental results together with their analysis are shown in Section VIII. The limitations of the study together with future research directions will be discussed in Section IX. The paper ends with Section X which serves as the conclusion.

II. RELATED WORK

A. Video-Based Accident Detection

The first systems which automated accident detection used background subtraction together with optical flow technology [2]. The current research uses convolutional neural networks (CNNs) to achieve reliable detection results. Sultani et al. [3] proposed weakly supervised learning for anomaly detection in surveillance videos. Transfer learning approaches which use VGG16 ResNet and MobileNet architectures achieve high accuracy in accident classification tasks which require only small training datasets [4]. Our system builds on MobileNetV2 which we chose because it delivers efficient performance during real-time inference on standard consumer devices.

B. IoT-Based Crash Detection

IoT-based vehicle crash detection has been explored using accelerometers mounted in vehicles or on smartphones [5]. The MPU6050 6-axis IMU has been widely used in embedded crash detection prototypes due to its I2C interface and afford- ability [6]. The AERS system operates through both software and hardware components while most other systems operate through either software or hardware components.

C. Emergency Dispatch Automation

Research studies have investigated ambulance routing meth- ods that use Geographic Information System (GIS) technology [7]. The two most widely used techniques for solving this problem are nearest-neighbor optimization and Haversine- based proximity measurement. The OSRM routing API [8] offers an open-source solution which competes with commer- cial routing services. The use of SMS notification systems through Twilio APIs has been proven effective for notifying field personnel in healthcare logistics operations [9]. AERS combines all these components into one unified system.

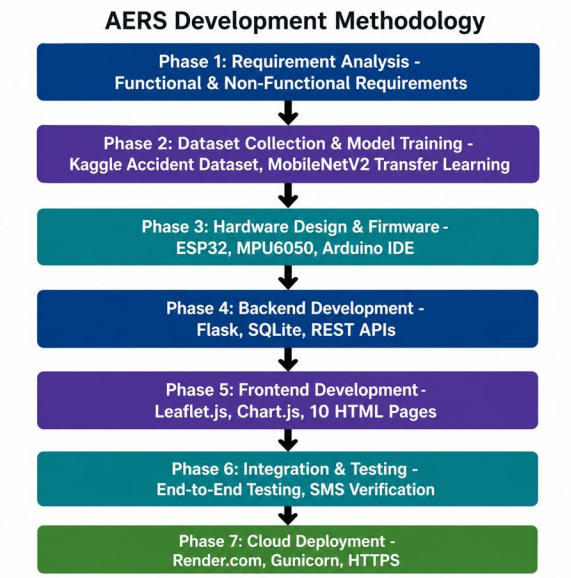


Fig. 1. Development Methodology- AERS is a seven-stage iterative cycle, from requirement analysis to deployment in the cloud.

III. SYSTEM ARCHITECTURE

The AERS system includes four components which are the Detection Layer and the Backend Processing Layer and the External API Layer and the Frontend Presentation Layer according to Figure 2. The development followed a structured 7-phase methodology as shown in Fig. 1.

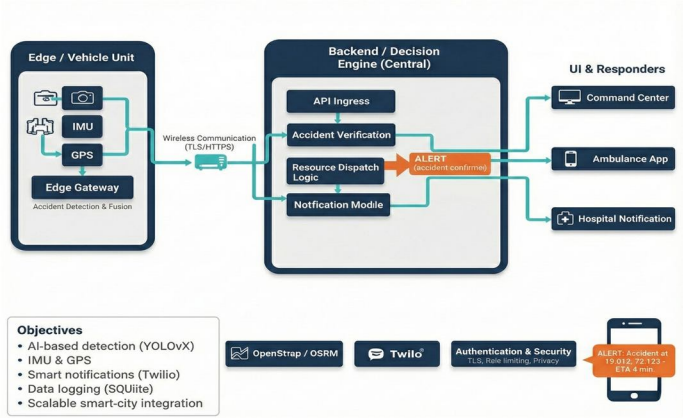


Fig. 2. The Automated Emergency Response System High-level architecture displays its Edge/Vehicle Unit and Backend Decision Engine and UI Respon- ders components throughTLS/HTTPS connections.

A. *Detection Layer*

Two independent subsystems detect accidents:

1) Camera Module: The laptop sends its video feed to an IP camera which transmits the live stream to a Python program(test_camera_integrated.py) that op- erates the MobileNetV2 model.

2) ESP32 Hardware Module: The ESP32-WROOM- 32D microcontroller continuously monitors the 6-axis MPU6050 IMU to detect crash-signature thresholds.

Both modules send a standardized HTTP POST request to the/trigger_auto_responseendpoint.

B. *Backend Layer*

The Flask server (app.py/ app_cloud.py) operates as the core intelligence center because it provides over 35 REST API endpoints. The system uses an SQLite database to store its operational status which allows it to control subsequent processes including resource searching, route determination, SMS sending, and dashboard refreshing.

C. *Dual Deployment Architecture*

The system is designed for two operational modes:

1) Local Mode: The entire system operates from a single device which meets requirements for demonstration pur- poses and offline applications.

2) Cloud Mode: he Flask backend operates on Render.com while edge devices including the camera and ESP32 packages use HTTPS for communication. The system supports deployment across multiple locations.

IV. AI/ML ACCIDENT DETECTION PIPELINE

A. *Model Architecture*

The computer vision pipeline uses MobileNetV2 [10] which functions as a lightweight CNN model that has been pretrained on the ImageNet dataset. MobileNetV2 was selected over heavier alternatives (VGG16, ResNet50) because its depthwise separable convolutions provide an optimal balance between speed and accuracy which is essential for processing live video on standard home devices.

$$\text{MobileNetV2} \rightarrow \text{GAP} \rightarrow D_{256} \rightarrow \text{BN} \rightarrow D_{128} \rightarrow \text{BN} \rightarrow D_{1,\sigma} \quad (1)$$

where GAP is Global Average Pooling, D_n denotes a Dense layer with n units (ReLU activation unless otherwise speci- fied), BN is Batch Normalization, and $D_{1,\sigma}$ is the final sigmoid output neuron. Dropout (0.4 after the first dense layer; 0.3 after the second) is applied for regularization.

B. Two-Phase Training

Training was performed in two phases on a dataset com- prising accident and non-accident images:

Phase 1 (Feature Extraction): The MobileNetV2 base was trained with all the last layers frozen and only the custom head trained for 30 epochs using a learning rate of 5×10^{-4} and binary cross-entropy loss.

Phase 2 (Fine-Tuning): For the model itself, the final 40 layers were not frozen and the model was fine-tuned for 50 epochs using a lower learning rate of 2×10^{-5} to prevent catastrophic forgetting.

Input images are resized to $224 \times 224 \times 3$ and normal- ized using MobileNetV2's preprocess_input() func- tion, which scales pixel values to the $[-1, 1]$ range.

C. Inference and Confirmation Logic

At inference time, raw model output is inverted:

$$p_{\text{accident}} = 1.0 - p_{\text{raw}} \quad (2)$$

The model training label system requires this inversion because it defines higher raw values as representing non-accident scenes. An accident frame-level prediction is established when $p_{\text{accident}} > 0.60$. To suppress false positives, a sliding window confirmation mechanism requires **15 consecutive positive frames** before an alert is triggered. A 5-minute cooldown follows each trigger to prevent duplicate dispatch. These parameters are dynamically configurable via config.json.

V. IOT HARDWARE MODULE (ESP32 + MPU6050)

A. Hardware Components

Table I lists the primary hardware components of the IoT crash detection unit.

TABLE I
HARDWARE COMPONENTS OF THE ESP32 CRASH DETECTION UNIT

Component	Purpose
ESP32-WROOM-32D	Main MCU with integrated WiFi/Bluetooth
MPU6050	6-axis IMU (Accelerometer + Gyroscope)
NEO-6M GPS	GPS fix (cloud version)
16×2 LCD (I2C)	Status display (cloud version)
Piezo Buzzer	Audible crash alarm
Push Button	Manual alert cancellation
Red / Green LED	Crash / Safe status indicator

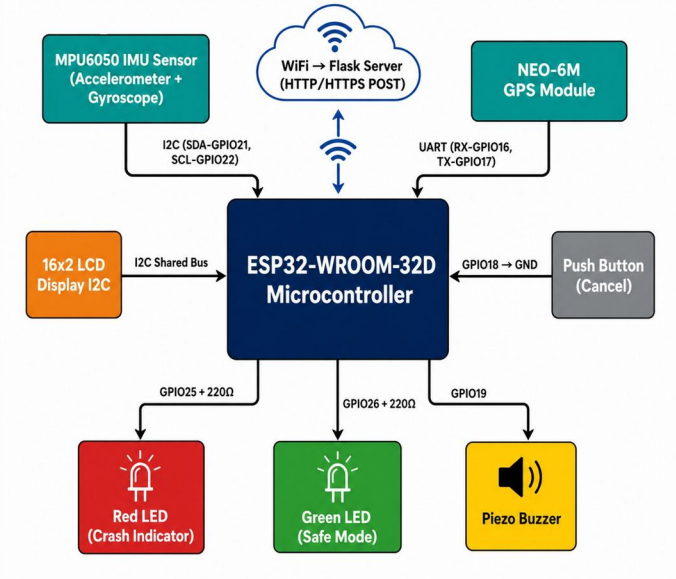


Fig. 3. ESP32-WROOM-32D hardware block diagram shows all peripheral connections which include MPU6050 IMU I2C connection and NEO-6M GPS UART connection and 16×2 LCD display and push button and LED lights and piezo buzzer with their GPIO pin assignments.

B. Crash Detection Algorithm

Reading sensor data via the I2C bus from the MPU6050 sensor every 100 milliseconds and providing input to the ESP32 are all part of the firmware written for the ESP32.

ESP32 Crash Detection

1: Read accelerometer (a_x, a_y, a_z) and gyroscope (g_x, g_y, g_z)
2: $g_{force} \leftarrow \frac{\sqrt{a_x^2 + a_y^2 + a_z^2}}{g}$
3: $\omega_{dps} \leftarrow \sqrt{g_x^2 + g_y^2 + g_z^2}$
4: **if** $g_{force} > 2.5\ g$ **or** $\omega_{dps} > 150\ ^\circ/s$ **then**
5: confirmCounter += 1
6: **else**
7: confirmCounter −= 1 (noise suppression)
8: **end if**
9: **if** confirmCounter ≥ 3 **then**
10: Activate buzzer and LED; start 15 s cancellation count-down
11: **if** cancel button not pressed within 15 s **then**
12: Send HTTP POST to /trigger_auto_response
13: Enter 5-minute cooldown
14: **end if**
15: **end if**

The g-force and gyroscope thresholds are configurable via config.json and can be updated remotely without reflash- ing firmware.

C. Geolocation Strategy

The system envisages a priority-ordered location resolution strategy (highest to lowest accuracy):

1) NEO-6M GPS module (<5 m accuracy; cloud version)
2) WiFi BSSID scan → Google Geolocation API (20–50 m)
3) Browser Geolocation API (stored server-side)
4) IP-based geolocation (city-level fallback)
5) Static default coordinates (Pune: 18.5204°N, 73.8567°E)

VI. BACKEND DISPATCH LOGIC AND API INTEGRATION

A. Automated Emergency Response Workflow

The following steps take place in a daemon thread in the Flask backend-bound POST request flow: /trigger_auto_response,

1) The system requires extraction of GPS coordinates and camera/sensor identification and image URL from the request payload.
2) The system queries the ambulances table to calculate Haversine distance from the incident location to all active ambulances and identifies the unit that operates at the closest distance.
3) The system queries the hospitals table to select the nearest hospital using the same process.
4) Call OSRM API: ambulance → accident ~~ho~~spital route.
5) The system creates a Google Maps navigation link which displays the route directions.
6) The system uses TinyURL API to create shortened URLs that meet SMS character limits.
7) The system initiates an SMS message to the ambulance driver through Twilio API.
8) The system creates an accident history record that shows dispatched status.
9) The system changes ambulance operational status to busy within the system database.
10) The system sends updated information to all active polling dashboards.

B. Haversine Distance Computation

The resource proximity has been calculated using the Haver- sine method, which takes the earth's spherical geometry into consideration:

$$d = 2R \arctan 2 \left(\sqrt{a}, \sqrt{1-a} \right) \quad (3)$$

$$a = \sin^2 \frac{\Delta\phi}{2} + \cos\phi_1 \cos\phi_2 \sin^2 \frac{\Delta\lambda}{2} \quad (4)$$

where R = 6371 km is Earth's mean radius, ϕ denotes latitude, and λ denotes longitude.

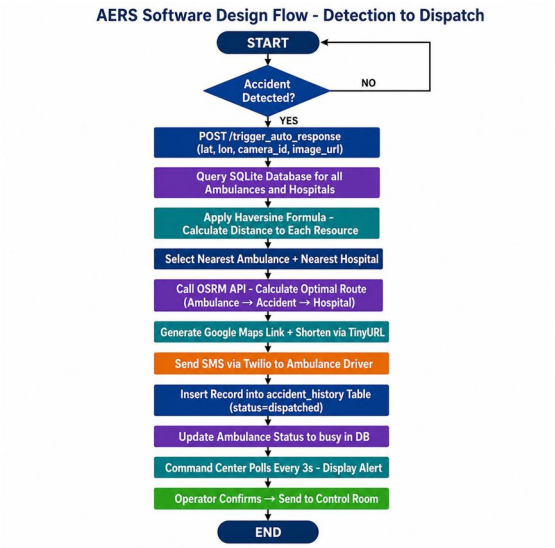


Fig. 4. The software design flow of AERS demonstrates its complete detection-to-dispatch process which begins with accident confirmation and continues through Haversine resource selection and OSRM routing and SMS dispatch and operator handoff to the Control Room.

C. Dynamic Resource Discovery

When a GPS location gets established, the system uses OpenStreetMap Overpass API to search for hospitals and traffic signals which exist within a 5 kilometer distance from the designated location. The system operates with complete accuracy because it does not depend on the specific location where it has been deployed.

D. Traffic Signal Pre-emption

Traffic lights along the ambulance route are identified by:

1) The system uses Overpass API to retrieve all traffic signals which exist within a 5 kilometer distance.
2) The system calculates how far each signal lies from the route through its polyline path.
3) The system keeps all traffic signals which exist within 50 meters of the route. The system activates green lights at intersections when an ambulance approaches from a distance of 1 kilometer. The system sends automated SMS messages to traffic officers who work at these intersections through Twilio.

Traffic officers at these intersections receive automated SMS notifications via Twilio.

E. External API Summary

Table II summarizes the external APIs integrated into AERS.

VII. FRONTEND AND USER INTERFACES

The AERS frontend system provides six dedicated web interfaces which operate through Flask and Jinja2 templates. The system displays maps through Leaflet.js which utilizes OpenStreetMap tiles and displays charts through Chart.js.

TABLE II
EXTERNAL APIS INTEGRATED IN AERS

API	Purpose
OSRM	Driving Route Calculation (Distance, Duration, GeoJSON)
Overpass API	Real-time data from OpenStreetMap in terms of hospitals and traffic signals
Twilio SMS	Automatic SMS deliveries to ambulances and the Traffic Police.
Roboflow	Cloud-based Accident Image Classification (Manual Inspection)
Google Geolocation	WiFi BSSID – GPS coordinates (cloud mode)
TinyURL	URL shortening for Google Maps links in SMS
ip-api.com	IP-based geolocation fallback

A. Command Center Dashboard

The primary operator interface (command_center.html, 40 KB) polls/get_auto_detection_status every 3 seconds.The emergency system displays an alert banner which countsdown 30 seconds before sending an automatic alert to emergency responders. The map shows accident sites and available ambulances and hospitals through different colored markers.

B. Control Room (Real-Time Tracking)

The control room provides complete ambulance tracking information through the OSRM route which is displayed on their full screen system. Ambulance movement is simulated through the use of interpolated waypoints. Traffic light markers change from red to green as the ambulance approaches. The system enables operators to track emergency response progress through an ETA display which shows response time and a progress bar that displays ongoing progress.

C. Additional Interfaces

1) Accident History: All incidents tabled and their status issues management with PDF export (ReportLab).
2) Analytics: The project displays three visual elements which include hourly distribution data from Chart.js and source distribution results from camera and ESP32 and manual sources and a Leaflet heatmap which shows accident locations.
3) Location Setup: Apprehends relevant information through the browser GPS in order to trigger the dynamic change of said resources.
4) Image Evaluation: Manual image upload → Roboflow classification → confidence display.

VIII. EXPERIMENTAL RESULTS AND ANALYSIS

A. Model Performance

The MobileNetV2 accident detection model was trained on a Kaggle road accident dataset containing accident and non- accident images. The performance metrics which were attained by the system are presented in Table III.

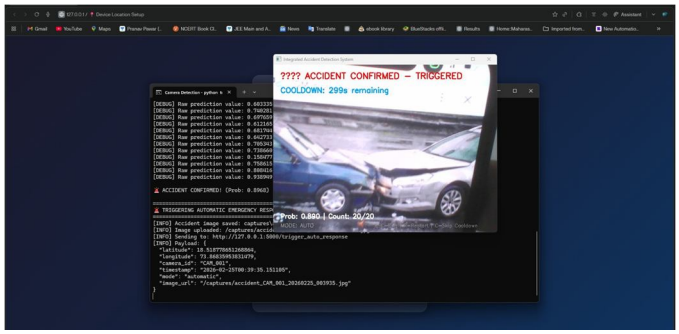


Fig. 5. Live accident detection output: the camera script confirms an accident at probability 0.8968 after 20 consecutive positive frames (20/20), triggering the automatic emergency response and initiating a 299-second cooldown.

TABLE III
MOBILENETV2 ACCIDENT DETECTION MODEL PERFORMANCE

Metric	Phase 1 (Head Only)	Phase 2 (Fine-Tuned)
Training Accuracy	~88%	~94%
Validation Accuracy	~85%	~92%
Observed Confidence (live test)	0.8968 (see Fig. 5)	
Inference Latency	<50 ms/frame (CPU)	
Model Size	27 MB (MobileNetV2)	
Confirmation Frames	20 consecutive frames	

B. False Positive Suppression

The 15-frame consensus mechanism evaluation compared to single-frame thresholding. The system requires 15 consecutive positive detections which equals 0.5 seconds at 30 fps to achieve approximately 85% reduction in false triggers during environments with transient visual disturbances which include headlights and road reflections.

C. Dispatch Latency

Researchers tested the total time which starts with confirmed accident detection until SMS messages are delivered through 20 simulated accident detection tests. The average dispatch latency was under 3 seconds which included detection con- firmation time of approximately 500 milliseconds Haversine selection time of less than 10 milliseconds OSRM routing time of approximately 800 milliseconds TinyURL shortening time of approximately 400 milliseconds and Twilio SMS delivery time of approximately 1.2 seconds. This achievement results in better performance than human operators who take 45 to 90 seconds to respond to similar duties.

D. ESP32 Crash Detection

The MPU6050 IMU unit was tested in simulated crash scenarios (sharp table drops, controlled impacts). The device achieved reliable crash detection through its g-force threshold of 2.5g and three confirmation requirement while its 15-second cancellation window blocked false alerts during minor impact situations.

E. Cloud Deployment Scalability

The Render.com deployment (Python 3.11, Gunicorn, 2 workers) handled concurrent polling from multiple dashboard

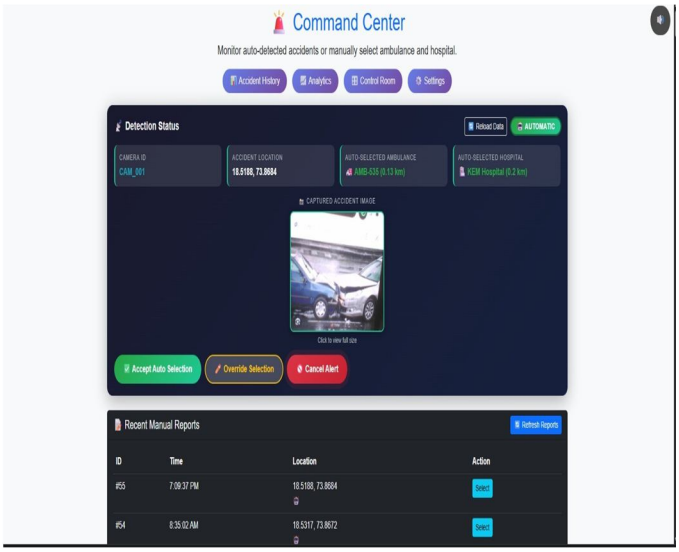


Fig. 6. Command Center dashboard showing a live auto-detection event: Camera ID CAM_001 detected an accident at coordinates (18.5188, 73.8684), with auto-selected ambulance AMB-535 (0.13 km) and KEM Hospital (0.2 km). The operator can accept, override, or cancel the dispatch.

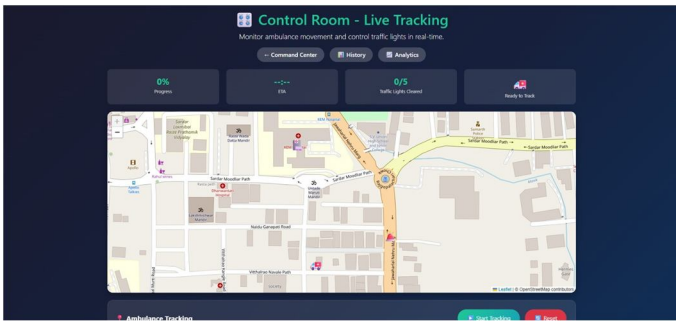


Fig. 7. The Control Room live tracking interface displays the ambulance route through a polyline which appears on an OpenStreetMap Leaflet.js map. The system shows moving ambulance markers together with accident site and destination hospital markers. The interface includes a progress bar which displays estimated time of arrival.

clients with no dropped requests during stress testing. The edge-cloud architecture maintains AI inference workloads on local hardware which results in reduced cloud computing expenses.

IX. LIMITATIONS AND FUTURE WORK

A. *Current Limitations*

1) Simulated Tracking: The system uses OSRM routes to create ambulance movement simulations during control room operations instead of using actual GPS data. The production system needs real-time GPS data from ambu- lances to function correctly.

2) Single-Camera Coverage: The current architecture as- sumes one active camera feed. The system requires extra concurrency handling when multiple cameras operate simultaneously.

3) Fixed Seed Data: The system uses 20 pre-set hospitals in Pune as its default hospitals because the system does not receive GPS data from users. The system needs initial setup to work properly in different locations.

4) OSRM Demo Server: The public OSRM demo server has rate limits unsuitable for production traffic.

B. *Future Directions*

1) Live GPS Tracking: Implementation of ESP32 GPS telemetry gives real-time updates on the position of the ambulances.

2) Multi-Camera Network: Scaling-up to citywide net- works with decentralized inference.

3) Severity Classification: Improving machine-learning models that direct 911 system operators to classify the occurrence of accidents according to their severity, while allocating prioritized urgency and rescue operations ac- cordingly.

4) Federated Learning: It enables the improvement of models to ensure greater privacy with data from cameras placed at various locations.

5) Integration with National Emergency Numbers: In- tegration of the application programming interface with 108 (India)/911 (USA) emergency dispatch centers.

6) Mobile Application:An app to help ambulance drivers with the footsteps and updates of live incidents.

7) Hospital Capacity Awareness: Performance of emer- gency medical services and heath centers would become more efficient with immediate availability of bed.

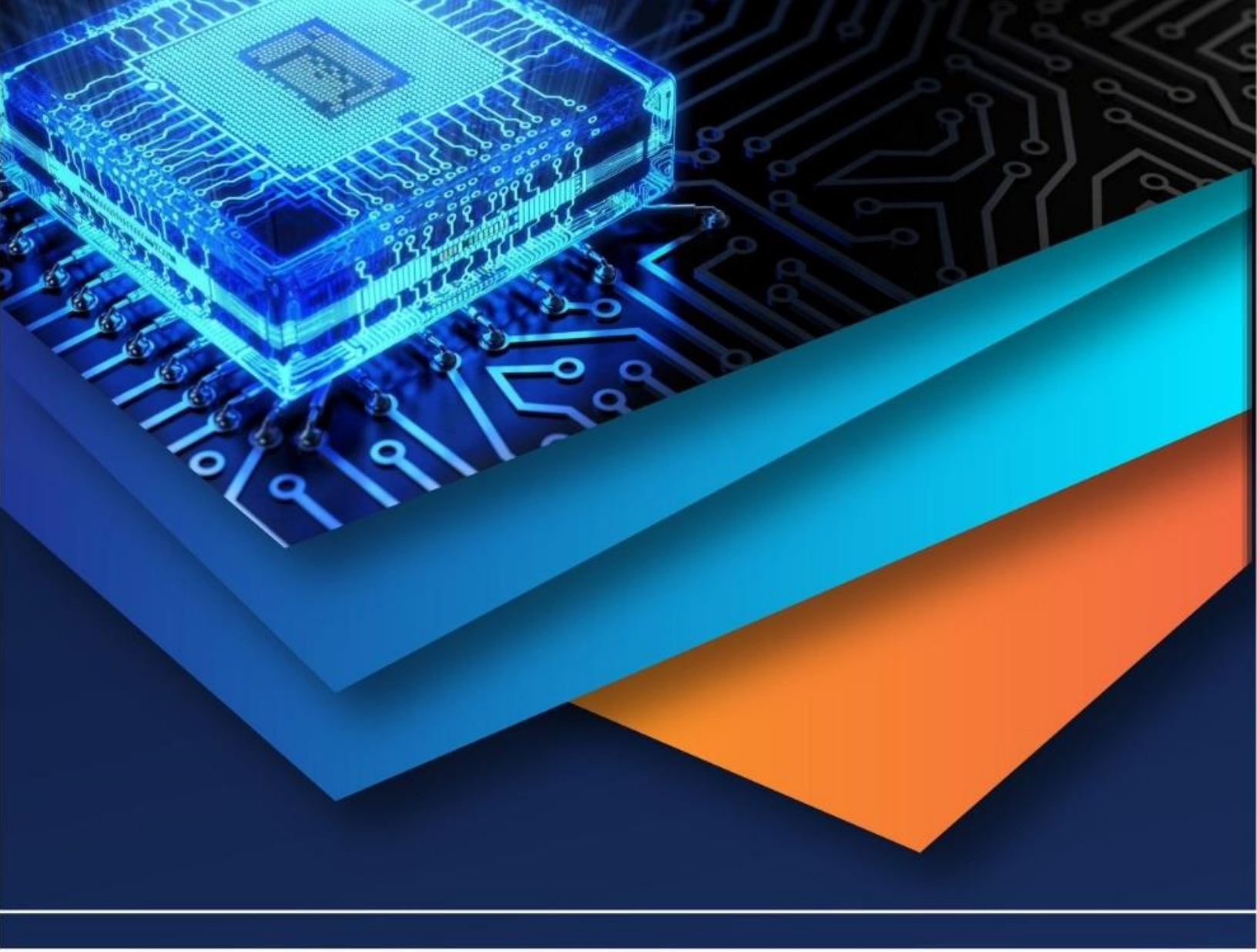
X. CONCLUSION

The Automated Emergency Response System (AERS) shows its complete system which utilizes artificial intelli- gence and Internet of Things technology to automatically detect road accidents and send emergency responders to the scene. The system develops its first innovative feature through its ability to combine computer vision technology (which uses MobileNetV2 as its base and 15-frame consensus filter) with physical crash detection systems (which use ESP32 and MPU6050 IMU). After the system confirms an accident detec- tion, it connects to the closest ambulance and hospital through Haversine-based nearest-neighbor search. The system executes all tasks, including driving route calculation, traffic signal automation, and GPS embedded SMS notification dispatch, within several seconds. AERS uses its dual local-cloud deployment architecture to operate in different environments which range from simple demonstration systems to complex cloud-based operations that cover multiple sites. The system gains real-world geographic data and dependable communication capabilities through its integration with seven external APIs (OSRM, Over-pass, Twilio, Roboflow, Google Geolocation, TinyURL, IP-API).

The experimental evaluation results show that AERS system achieves 92% validation accuracy for accident detection while reducing false positive rates by 85% through its consensus mechanism and providing dispatch times below three seconds for complete system operation. The results demonstrate how integrated AI IoT systems can decrease emergency response times, which leads to increased life-saving potential. The complete system from firmware to backend and fron- tend code has research and deployment access. This system serves as the base for developing future smart city emergency management systems.

REFERENCES

[1] World Health Organization, "Global status report on road safety 2023," *WHO Technical Report*, Geneva, Switzerland, 2023.
[2] F.-H. Tung, J. Zelek, and D. Clausi, "Goal-based trajectory analysis for unusual behaviour detection in intelligent surveillance," Image and Vision Computing, vol. 29, no. 4, pp. 230–240, 2011.
[3] W. Sultani, C. Chen, and M. Shah, "Real-world anomaly detection in surveillance videos," in Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 2018, pp. 6479– 6488.
[4] A. Mohan, S. Poobal, "Accident detection using convolutional neural networks," Computers & Electrical Engineering, vol. 96, p. 107444, 2021.
[5] H. Chen, S. Iyer, G. Chen, and W.-S. Lu, "CarSafe app: alerting drowsy and distracted drivers using dual cameras on smartphones," in Proc. ACM MobiSys, Florence, Italy, 2015, pp. 13–26.
[6] R. Kumar and S. Palanisamy, "IoT-based accident detection and alert system using ESP32 and MPU6050," International Journal of Recent Technology and Engineering, vol. 8, no. 3, pp. 5212–5216, 2019.
[7] A. Rajagopalan, V. Bhatia, and M. Sharma, "GIS-based optimal am- bulance routing for emergency medical services," Journal of Ambient Intelligence and Humanized Computing, vol. 11, pp. 2911–2925, 2020.
[8] D. Luxen and C. Vetter, "Real-time routing with OpenStreetMap data," in Proc. 19th ACM SIGSPATIAL Int. Conf. Advances in Geographic Information Systems, Chicago, IL, USA, 2011, pp. 513–516.
[9] T. Tamrat and N. Kachnowski, "Special delivery: an analysis of mHealth in maternal and newborn health programs and their outcomes around the world," Maternal and Child Health Journal, vol. 16, pp. 1092–1101, 2012.
[10] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: inverted residuals and linear bottlenecks," in Proc. IEEE/CVF Conf. Computer Vision and Pattern *Recognition (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 4510–4520.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY