IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

"SmartResQ – AI-Based Emergency Response System"

Mr. Bhushan Sunil Magare[1], Prof. Shivam Limbhare[2], Prof. Manoj Vasant Nikum[*3]

Research Scholar, Master of Computer Applications, Shri JRIT, Dondaicha, KBC NMU Jalgaon, Maharashtra, India[1]

Assistant Professor, Master of Computer Applications, Shri JRIT, Dondaicha, KBC NMU Jalgaon, Maharashtra, India[2]

HOD, Master of Computer Applications, Shri JRIT, Dondaicha, KBC NMU Jalgaon, Maharashtra, India[*3]

Abstract: When critical incidents occur in urban or semi-urban environments, every second counts. Traditional emergency dispatch frameworks continue to rely heavily on manual phone calls and voice descriptions, which frequently lead to location ambiguity, delayed response times, and fragmented coordination. To bridge this gap, we developed SmartResQ, a practical web-based command platform built specifically to streamline emergency reporting and hazard monitoring. The system unites citizen SOS transmissions, a keyword-scoring natural language classifier, deterministic computer-vision heuristics, and simulated Internet of Things (IoT) sensor telemetry into a single cohesive interface. Running on a React and TypeScript frontend backed by a Python Flask 3.0 REST service and an SQLite database, SmartResQ allows users to submit geolocated incident reports while simultaneously monitoring environmental hazards like high temperatures, smoke, and gas leaks. Our implementation successfully tackles common integration pitfalls, such as database alert duplication caused by concurrent frontend-backend triggers. Through local testing and database verification, we demonstrate that combining lightweight heuristics with a structured relational backend offers a robust, responsive foundation for modern smart city emergency management.

Keywords: SmartResQ, Emergency Dispatch, Natural Language Processing, Rule-Based Vision Heuristic, IoT Simulation, Flask API, SQLite.

I. INTRODUCTION

In any public safety scenario—whether it is a highway collision, a residential fire, or an acute medical emergency—the speed and accuracy of the initial response dictate the outcome. For decades, emergency services have relied on telephone dispatch centers where operators manually gather caller details, transcribe notes, and estimate locations. While indispensable, this traditional model is prone to human error, cognitive overload under high call volumes, and significant delays when callers are panicked or unable to describe their surroundings.

With the proliferation of connected devices, browser geolocation APIs, and sensor networks, there is a clear opportunity to modernize emergency workflows. However, many proposed software solutions tend to treat reporting, image recognition, and hardware monitoring as isolated silos rather than a unified operational pipeline.

Furthermore, heavy deep-learning models often introduce excessive setup complexity and hardware overhead that hinder practical deployment in resource-constrained academic settings.

To explore a more practical and integrated approach, we built SmartResQ. Our primary objective was to design a functional, end-to-end prototype that connects citizen-generated SOS requests with automated hardware telemetry in a single Command Center dashboard. By pairing a responsive React/TypeScript client with a lightweight Python Flask REST API and a structured SQLite database, SmartResQ demonstrates how lightweight Natural Language Processing (NLP) and rule-based computer-vision heuristics can organize emergency data efficiently without sacrificing transparency or maintainability.

II. LITERATURE SURVEY

Our review of contemporary literature in emergency management and smart city frameworks points to several key areas of development:

1. **Evolution of Computer-Aided Dispatch (CAD):** Early dispatch systems focused primarily on database logging for police and fire units. Recent studies emphasize the transition toward citizen-centric web portals capable of capturing structured digital inputs, such as precise GPS coordinates and photographic evidence, directly at the source.

 This work is licensed under a Creative Commons Attribution 4.0 International License

2. **Role of IoT in Hazard Monitoring:** The integration of physical sensors—such as thermistors, smoke detectors, and MQ-series gas sensors—allows automated systems to detect environmental anomalies (e.g., rising room temperatures or LPG leaks) instantaneously, bypassing the need for human observation.

3. **Balancing Deep Learning and Heuristics:** While Convolutional Neural Networks (CNNs) deliver impressive image classification results, their training requirements and inference latency can be prohibitive. Academic prototypes frequently adopt deterministic, feature-based heuristics (evaluating color histograms and edge density) to achieve reliable, interpretable image sorting.

4. **Text Classification and Triage:** Keyword scoring and weighted token matching techniques are applied to emergency text descriptions to automatically categorize incidents into operational categories (such as medical crises versus vehicular accidents) before dispatchers review the logs.

III. RESEARCH METHODOLOGY

SmartResQ was engineered following an iterative, applied development methodology. Rather than attempting to build an unverified enterprise-scale deployment from day one, we focused on establishing a stable data loop: user input or sensor telemetry flows through the frontend, hits the Flask REST API, persists into SQLite, and updates the Command Center display in real time.

3.1 Architectural Design

The system adopts a three-tier client–server–database model. The frontend client is written in TypeScript using React and bundled with Vite, offering dedicated views for citizen reporting, Vision Lab preset testing, IoT simulation, and map visualization. The backend server utilizes Python and Flask 3.0, organizing endpoints into modular blueprints for emergencies, alerts, sensors, and AI detections. All operational data is stored locally in backend/smartresq.db.

3.2 Core Analytical Modules

To ensure our algorithms remain transparent and fully inspectable during academic evaluation, we implemented specific deterministic logic rather than black-box models:

1. **NLP Keyword Scoring:** When a citizen submits an SOS description, a lightweight keyword scanner evaluates the text against domain dictionaries to categorize the event into ACCIDENT, FIRE, or MEDICAL_EMERGENCY, logging a confidence score.

2. **Rule-Based Vision Inspection:** The vision module (visionAnalyzer.ts) inspects image features such as fire-color ratio (fireRatio), smoke ratio (smokeRatio), and edge density (edgeDensity). If fire pixels exceed defined thresholds, it flags a FIRE hazard; if edge density and smoke indicators align, it identifies a vehicular ACCIDENT; otherwise, it defaults to NORMAL traffic.

3. **Automated IoT Telemetry Ingestion:** The backend sensor route (sensor_routes.py) accepts simulated telemetry readings. If temperature reaches or exceeds 60°C, smoke exceeds 400 PPM, or flame detection is active, the backend automatically generates an active database alert for a fire hazard. Similarly, gas levels exceeding 500 PPM trigger a gas-leak alert.

3.3 Operational Workflow & Database Schema

Table 1. SQLite Database Schema

SQLite Table	Core Schema Elements	Functional Role in SmartResQ
users	id, name, phone/contact info	Stores contact details of citizens submitting SOS reports.
emergencies	id, user_id, emergency_type, description, image_path, location/GPS, status, priority	Persists active citizen incidents displayed on the Command Center map.
alerts	id, emergency_id, alert_type, message, status, created_at	Logs active system alerts (IoT alerts maintain emergency_id = NULL).
iot_sensor_data	id, temperature, smoke_level, gas_level, fire_detected, created_at	Stores raw simulated environmental telemetry.
ai_detections	id, emergency_id, detected_type, confidence, model_name, created_at	Records NLP and vision classification outputs tied to incident IDs.

IV. RESULTS

The system was thoroughly tested in a local Windows development environment (VS Code, Python Flask 3.0, Node.js/Vite, Brave browser), with database states verified using a custom inspection script (inspect_db.py).

 This work is licensed under a Creative Commons Attribution 4.0 International License

4.1 Summary of Experimental Findings

Table 2. Summary of Experimental Findings

Test Scenario	Tested Component	Observed Outcome
GPS Auto-Detect	HTML5 Geolocation API	Successfully retrieved accurate local coordinates, resolving earlier coordinate inaccuracies.
Citizen SOS Triage	UserReportingView & NLP Scorer	Created road accident (EMG-9) and medical emergency (EMG-10) records, rendering as Critical priority.
Vision Lab Evaluation	visionAnalyzer.ts & Preset Assets	Correctly discriminated between industrial fire (fire.png), collision (accident.png), and normal street traffic (normal.png).
IoT Duplicate-Alert Fix	sensor_routes.py & emergencyStore.ts	Eliminated duplicate alert creation. A simulated Fire Outbreak correctly inserted exactly 1 new alert row (ID 19).

4.2 Addressing Technical Challenges

During integration testing, we encountered a notable architectural challenge: triggering a simulated fire outbreak resulted in two duplicate alert entries within the SQLite database. Debugging revealed that both the backend sensor ingestion handler and the frontend zone-transition store were independently issuing POST requests to create alerts. By refactoring the frontend store to handle state transitions and browser notifications without sending redundant HTTP alert requests, we achieved clean, one-to-one alert persistence.

V. DISCUSSION AND ANALYSIS

Our practical experience developing SmartResQ yields several important insights for academic prototyping in computer science:

1. **Integration Complexity:** Building an end-to-end system forces a developer to confront real-world integration issues—such as race conditions in REST APIs and database state synchronization—that are often glossed over in purely theoretical designs.
2. **Value of Inspectable Heuristics:** Using rule-based heuristics for image analysis and text classification allowed us to debug and fine-tune system behavior immediately, without waiting for lengthy model retraining cycles.
3. **Limitations:** Because SmartResQ relies on simulated IoT telemetry and preset image assets for testing, its performance under live, chaotic real-world sensor streams and unstructured CCTV feeds remains an area for future hardware expansion.

Data Flow & System Architecture Diagram

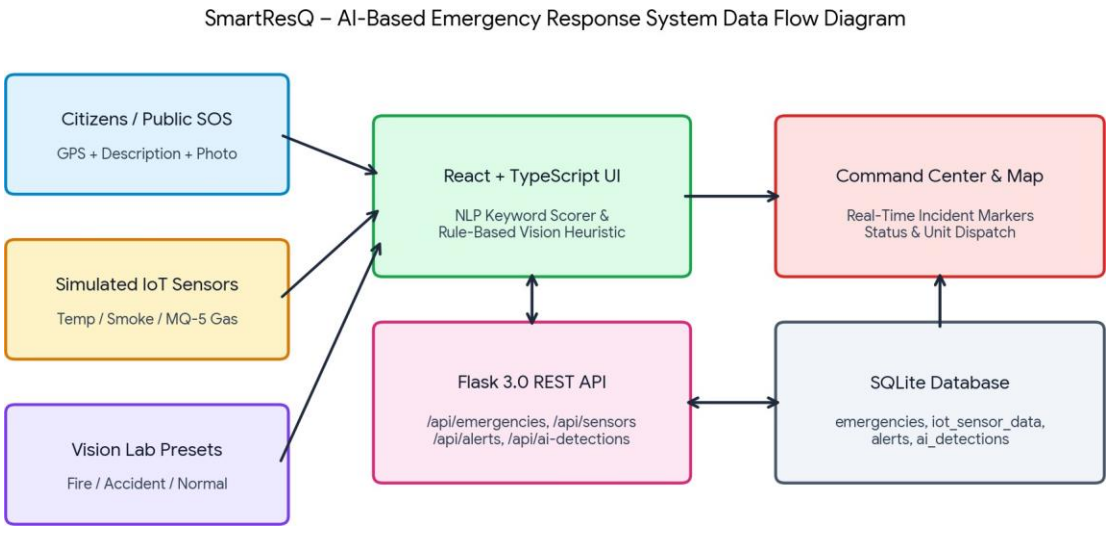


Fig. 1 SmartResQ – AI-Based Emergency Response System Architecture & Data Flow

 This work is licensed under a Creative Commons Attribution 4.0 International License

VI. CONCLUSION

SmartResQ demonstrates that effective emergency response platforms can be constructed by combining accessible web technologies with transparent, lightweight analytical heuristics. By integrating citizen reporting, automated IoT telemetry monitoring, and a centralized relational database, the project successfully achieves its primary design goals. Future extensions of this work will involve deploying physical microcontroller hardware nodes (such as ESP32 sensors) to feed live environmental data into the Flask backend, alongside exploring transfer learning with lightweight pre-trained vision models for enhanced field testing.

REFERENCES

[1]. Flask 3.0 Documentation, Pallets Projects, Python Web Framework – https://flask.palletsprojects.com/
[2]. React and TypeScript Documentation, Meta Open Source – https://react.dev/
[3]. SQLite Relational Database Engine Documentation, Serverless SQL Storage – https://www.sqlite.org/docs.html
[4]. MDN Web Docs, HTML5 Geolocation API Specification – https://developer.mozilla.org/
[5]. A. K. R. Avula, "AI for Environmental Resilience: Innovations in Disaster Detection and Response," International Journal of Computer Engineering and Technology (IJCET), vol. 15, no. 6, pp. 1531-1538, 2024.
[6]. K. Karunakaran & L. Luxshi, "AI-Driven Disaster Prediction and Early Warning Systems: A Systematic Literature Review," International Journal of Research in Computing (IJRC), vol. 4, no. 2, pp. 44-55, 2025.

 This work is licensed under a Creative Commons Attribution 4.0 International License