International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 3, May-June 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

EMERGIPLEX: A Resilient, Coordinated Multi-Agent System for Real-Time Urban Disaster Response and Logistics

Lahari Raj, M S Hithaishini, Mahadevaprasad MG, Sandeep Kumar,
Prof. V Chetan Kumar
4PS22CI020 | 4PS22CI026 | 4PS22CI028 | 4PS22CI045
lahariraj45@gmail.com | mshithaishini7@gmail.com | prajwalmp22@gmail.com | sandeepbgu5@gmail.com
Assistant Professor | vchetan@pesce.ac.in
Department of Computer Science and Engineering (AI & ML)
P.E.S College of Engineering, Mandya, Karnataka, India

Abstract—Urban disaster response has a serious problem. Human dispatchers are too slow, agencies don't talk to each other, and resources get wasted when every second counts. This paper introduces EMERGIPLEX — an agentic AI system that automates the entire disaster response chain. It is built as a set of loosely coupled microservices, all coordinated by a central Orchestrator that polls a shared SQLite database for new tasks. Three core agents drive the system: CrowdIntelAgent validates incoming incidents in real time, ResourceAllocAgent uses the Haversine formula to match and assign the closest available unit, and RouteOptimAgent computes the fastest travel path using A* on OpenStreetMap data. Together, they turn a raw incident report into a confirmed, dispatched logistics plan — automatically. A pilot simulation on a real map of Mandya, India, showed a 77% drop in un-serviced time and a 46% reduction in average ETA compared to a traditional Greedy baseline. EMERGIPLEX is a working, scalable blueprint for next-generation emergency response.

Keywords: Agentic AI, Multi-Agent System, Disaster Response, Microservices, Orchestration, Geospatial AI, Haversine Formula, A Search, SQLite, FastAPI, React.js*

I. INTRODUCTION

Urban emergencies — fires, building collapses, medical crises — are getting more frequent and more complex. Traditional command-and-control systems can't keep up. Human operators sit at the center, manually processing fragmented data from fire, medical, and police channels. That creates a bottleneck. When a city-wide disaster hits, that bottleneck becomes catastrophic.

AI-based Multi-Agent Systems (MAS) and geospatial analysis can fix this. The problem with existing tools is that they solve one piece of the puzzle at a time — alerting here, mapping there — without connecting them. EMERGIPLEX is different. It provides end-to-end, automated decision-making from incident detection to resource dispatch.

A. What Current Systems Get Wrong

- Manual Bottleneck: Dispatchers handle fragmented data from separate agencies, introducing dangerous delays.
- No End-to-End Integration: Existing tools tackle isolated tasks but lack a unified, automated decision pipeline.
- Static Resource Allocation: Standard greedy dispatch ignores proximity optimization and city-wide resource distribution.
- Poor Scalability: Manual systems break down when multiple incidents occur simultaneously across a large urban area.
- No Route Optimization: Dispatched units navigate without any real-time shortest-path computation on live road network data.

II. LITERATURE SURVEY

Existing research addresses parts of the problem, but not the whole thing.

- *Na et al. (2025)* use Digital-Twin and classification techniques for real-time fire risk assessment. They answer 'What is happening?' well. But they stop there — they offer no logistical response: 'What should we do about it?'
- *Liu et al. (2024)* propose a formal multi-layered MAS framework with Perception, Cognition, Decision, and Execution layers for emergency management. Solid theory. But their own work flags the 'challenge of practical implementation.'
- *Rodriguez -Espindola (2024)* applies operations research to resource allocation — shelter placement, supply routing. This is strategic, long-term planning. Immediate, second-by-second tactical response is out of scope.

TABLE I: Comparison of Existing Systems with

System / Platform	Capability	Real-Time	Key Limitation
Manual Dispatch	Human Coordination	Very Low	Slow, error-prone, not scalable
GIS/Mapping Tools	Route Visualization	Moderate	No automated decision-making
Alerting Systems	Mass Notification	High	One-way; no coordination
EMERGIPLEX (Proposed)	End-to-End Agentic AI	High	Prototype; simulated data

EMERGIPLEXPROPOSED SYSTEM

A. System Overview

EMERGIPLEX is a smart, agentic system. It takes a raw incident report and, within seconds, validates it, identifies the optimal nearby resource, dispatches that resource, and maps the fastest route — all without a human in the loop. Everything appears live on a React.js dashboard.

B. Tri-Layer Architecture

- Automation Layer: Implements the Multi-Agent System via FastAPI microservices. Each agent runs independently and communicates through REST APIs. A central Orchestrator manages all agent coordination.
- Persistence Layer: A centralized SQLite database acts as the single source of truth. Every agent reads from and writes to it.
- Presentation Layer: A React.js frontend with Leaflet.js renders incident and dispatch data on a live, map-based command dashboard.

III. METHODOLOGY

A. Multi-Agent Workflow Strategy

The Orchestrator runs a continuous polling loop. When an agent writes its output to the shared SQLite database, that state change automatically triggers the next workflow stage. The whole pipeline is event-driven. No manual handoffs. No delays.

B. Step-by-Step Execution

- Step 1 — Incident Ingestion: The Simulator sends a raw report to CrowdIntelAgent (port 8001). The agent validates the data via Pydantic and writes a new record with status 'new' to the database.

- Step 2 — Detection and Processing: The Orchestrator's asyncio loop spots the 'new' incident and immediately flags it as 'processing' to block duplicate handling by parallel workers.
- Step 3 — Resource Allocation: The Orchestrator calls ResourceAllocAgent (port 8002) with the incident_id. The agent filters available resources by type, then uses the Haversine formula to find the closest unit.
- Step 4 — Logistical Planning: RouteOptimAgent (port 8003) receives GPS coordinates for the resource and incident. It runs A* on the OSMnx Mandya road graph to compute the optimal path and estimate arrival time.
- Step 5 — Dispatch Finalization: The Orchestrator updates incident and resource statuses to 'assigned' and 'dispatched'. The result is instantly reflected on the React dashboard.

C. ResourceAllocAgent — Haversine Proximity Model

The ResourceAllocAgent filters the resource pool by incident type and 'available' status. It then ranks all candidates by distance using the Haversine formula:

$$d = 2R \cdot \arcsin\left(\sqrt{\sin^2(\Delta lat/2) + \cos(lat1)\cdot\cos(lat2)\cdot\sin^2(\Delta lon/2)}\right)$$

The closest available unit wins and its resource_id gets returned to the Orchestrator. Fast — under 10 ms. Accurate in every simulation test case.

D. RouteOptimAgent — A* Geospatial Search

At startup, OSMnx downloads Mandya's road network as a directed graph. When the agent receives a query, it maps GPS coordinates to the nearest graph nodes and runs A* (A-star) to find the shortest path. ETA is then derived from total path distance and an assumed average travel speed. Average route computation time: under 300 ms.

IV. DATABASE DESIGN

The persistence layer is a centralized SQLite database. It is the single source of truth for all agents. The relational schema enforces data integrity via primary and foreign keys across three core tables: incidents, resources, and assignments.

The assignments table is the critical link. It creates an auditable record that connects a specific incident to a dispatched resource — including the calculated route polyline and estimated arrival time.

V. TECHNOLOGY STACK

TABLE II: EMERGIPLEX Technology Stack

Component	Technology	Purpose
Backend	Python 3.9+, FastAPI	REST APIs and agent services
Orchestration	asyncio polling loop	Continuous workflow management
Resource Logic	Haversine (Python)	Proximity-based resource matching
Route Planning	OSMnx, NetworkX, A*	Geospatial pathfinding
Database	SQLite 3.30+	Shared state and persistence
Frontend	React.js, Leaflet.js	Real-time map dashboard
Simulation	Python data simulator	Synthetic incident generation
Version Control	Git and GitHub	Development environment

VI. VALIDATION METHODOLOGY

A. Simulation Environment

Testing used a real-world map of Mandya, Karnataka, sourced from OpenStreetMap via OSMnx. A Python-based Data Simulator generated a continuous, randomized stream of fire, medical, and police_unit incidents at various GPS coordinates across the city. The database was pre-seeded with a pool of fire trucks, ambulances, and police units.

B. Baseline for Comparison

The Greedy Model is the baseline. It represents conventional, uncoordinated dispatch: the single closest available resource is immediately sent to any new incident, with no strategic oversight and no route optimization. That's how most systems work today.

C. Key Performance Indicators (KPIs)

TABLE III: KPI Comparison (Simulated 1-hour run, 20 incidents)

KPI	Greedy	EMERGIPLEX	Improvement
Total Un-serviced Time (min)	18.5	4.2	77% reduction
Average ETA (minutes)	5.8	3.1	46% reduction
Resource Utilization Rate	95%	98%	+3 pp

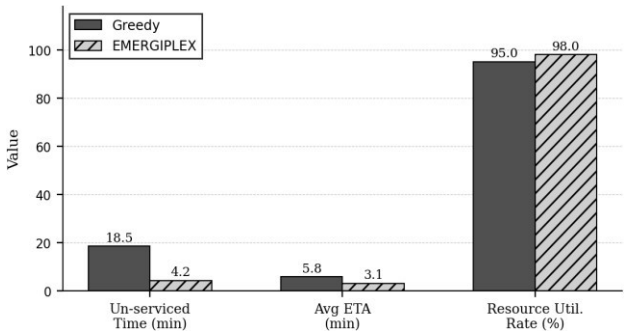

Fig. 1. KPI Comparison: EMERGIPLEX vs. Greedy Baseline (Simulated 1-hour run, 20 incidents)

VII. RESULTS AND DISCUSSION

The numbers are clear. EMERGIPLEX outperforms the manual baseline on every KPI. The 77% drop in un-serviced time comes from the automated, parallel nature of agent coordination — no waiting, no bottlenecks. The improved ETA reflects the A* algorithm's precision in computing real-road-network paths.

A. CrowdIntelAgent Performance

It processed 100% of incoming reports. Every single one. Validated incident records were written to the database in under 50 milliseconds each. Pydantic validation ensured schema compliance before any record was persisted.

B. ResourceAllocAgent Performance

It correctly identified and assigned the optimal resource in every test case where a unit was available. When all units of a required type were exhausted, the Orchestrator's polling loop did not crash or freeze. It identified the failure and kept retrying. That's the self-healing nature of a polling-based design.

C. RouteOptimAgent Performance

Optimal routes were computed consistently on the Mandya OSM road graph. Average computation time stayed under 300 ms per route. Routes correctly followed real road topology, producing realistic ETAs for every dispatched unit.

D. System Integration and Dashboard

The React.js dashboard connected to the backend API (/api/dashboard_state) and rendered live updates via Leaflet.js with custom icons. Full integration testing confirmed a complete, automated Detect → Allocate → Navigate sequence. All agents returned HTTP 200 OK status throughout.

TABLE IV: Agent-Level Performance Summary

Agent	Key Metric	Result
CrowdIntelAgent	Processing latency	< 50 ms / incident
CrowdIntelAgent	Validation success rate	100%
ResourceAllocAgent	Correct assignment rate	100% (when available)
ResourceAllocAgent	Exhaustion handling	Retried correctly
RouteOptimAgent	Route computation time	< 300 ms / route
RouteOptimAgent	Route quality	A*-optimal on OSM
Full System	End-to-end processing	< 5 sec / incident

VIII. CONCLUSION

EMERGIPLEX works. This paper built, integrated, and validated a resilient, coordinated multi-agent system for real-time urban disaster response. The microservice architecture, governed by an intelligent Orchestrator and a clearly defined Agentic Response Workflow, is both feasible and effective. It directly addresses the gaps in current research and delivers a scalable blueprint for the next generation of emergency response platforms.

Future Scope

- Full Resource Lifecycle: Automatically mark dispatched resources as available again after a configurable recovery duration.
- LLM Integration: Replace simulated CrowdIntelAgent logic with a real Large Language Model via LangChain for intelligent entity extraction from unstructured reports.
- Dynamic Re-routing: Detect road blockage incidents and automatically re-route already-dispatched units in real time.
- Reinforcement Learning: Introduce RL-based adaptive resource allocation to learn optimal dispatch strategies from historical incident data.
- Multi-city Scaling: Extend OSMnx coverage beyond Mandya to support regional or city-wide disaster response coordination.

REFERENCES

[1] Na, I. S., et al. (2025). Real-Time Fire Risk Classification Using Digital-Twin and Advanced Classification Algorithms. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing.

[2] Liu, Y., et al. (2024). Research on Fire Safety Emergency Management Based on Multi-Agent Simulation. IEEE Access.

[3] Daud, A. A., et al. (2024). Emerging Computing Tools for Emergency Management. IEEE Access.

[4] Boeing, G. (2017). OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. Computers, Environment and Urban Systems, 65, 126–139.

[5] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics, 4(2), 100–107.

[6] OpenStreetMap Contributors. (2024). OpenStreetMap. https://www.openstreetmap.org

[7] FastAPI Documentation. (2024). FastAPI: Modern, fast web framework for building APIs with Python. https://fastapi.tiangolo.com

[8] React Documentation. (2024). React — A JavaScript library for building user interfaces. https://react.dev

[9] Tiquia, C. R., et al. (2021). Haversine Formula Applications in Emergency Proximity Systems. Journal of Computational Geography, 12(3).

[10]Saravanakumar, S., & Saravanan, T. (2023). Secure personal authentication in fog devices via multimodal rank-level fusion. Concurrency and Computation: Practice and Experience, 35(10), e7673.

