Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 12 , December - 2025

Cloud-based Emergency Ambulance System

Dr. Vanita S. Buradkar
Professor
RCERT Chandrapur

Mr. Piyushchandra S. Vyas, Ms. Shravani P. Trikolwar, Ms. Radhika S. Sahu
Ms. Shreya B. Wankhede, Ms. Vaishnavi Jivtode
Student
RCERT Chandrapur

1. INTRODUCTION

Efficient ambulance dispatch is critical for saving lives in emergencies, yet traditional systems often rely on manual calls and static maps, leading to delays and suboptimal routing. For example, delays in ambulance arrival due to manual location handling can worsen patient outcomes. Modern solutions leverage digital technologies (smartphones, cloud computing, GIS) to accelerate dispatch. Khalique *et al.* (2017) developed a one-click smartphone app that automatically signals nearby ambulances with the patient's GPS location, greatly reducing call-handling overhead [3]. Likewise, Sulthan *et al.* (2025) built a cloud-based system where a mobile app requests an ambulance and the backend immediately assigns the nearest unit via geofencing and smart routing [1].

In response to these advances and remaining limitations, we propose a web-based real-time ambulance dispatch simulator. The system automatically generates incident events and assigns ambulances via a Flask backend, displays live ambulance, hospital, and incident markers on an interactive

LeafletJS map, and provides role-based dashboards for drivers and hospital administrators. Each phase of the incident lifecycle (dispatch, en route, arrival, patient drop-off) is logged for later analysis. By combining automated assignment, real-time tracking, and comprehensive logging, the simulator provides an integrated platform for EMS training and decision support. The system's key features include automated incident generation, dynamic routing, interactive mapping, and distinct user interfaces – collectively advancing the state-of-the-art in web-based EMS simulation.

Keywords: Emergency Response, Ambulance Dispatch, Real-Time Tracking, GIS, Leaflet, Flask, Smart Healthcare

1.1 Theoretical and Technological Foundations

Ambulance dispatch systems are grounded in operations research and information technology. The dispatch decision is a dynamic optimization task: assign available vehicles to calls to minimize response time or maximize coverage.

Early systems used simple heuristics (e.g. send the closest available unit), but modern approaches apply sophisticated algorithms to improve efficiency. For example, Li *et al.* (2014) integrated Information and Communication Technology (ICT) and Geographic Information Systems (GIS) on a cloud platform to track ambulance locations and automatically allocate the nearest available unit, demonstrating reduced transport times. In practical terms, this means that dispatch decisions can be made using real-time location data rather than relying on manual estimation.

Key enabling technologies include GIS and mapping libraries, GPS tracking, and cloud/web frameworks. GIS tools allow spatial visualization of ambulance and incident locations. For instance, Oghenekaro and Adepoju (2023) built a LeafletJS-based dispatch system where ambulance locations are tracked and response time is optimized through interactive mapping. GPS devices in ambulances provide continuous location feeds, which modern dispatch software uses to update maps and compute routes dynamically. Cloud computing and service-oriented architectures support scalability: by running core dispatch logic on cloud servers, the system can flexibly add resources (e.g. new ambulances or geographic regions) without heavy on-site hardware.

From a software perspective, web frameworks and APIs form the backbone. The Python Flask framework (Ronacher, 2010) is an example of a lightweight, flexible environment for building such applications. The frontend uses LeafletJS, an open-source JavaScript library for interactive maps, which is well suited for responsive, browser-based GIS visualizations. To support real-time updates, the system employs WebSocket or AJAX polling so that map markers and dashboards update seamlessly as ambulance states change. Finally, role-based interfaces (for drivers, dispatchers, and hospitals) draw on principles of software engineering (separation of concerns, RESTful API design) to present relevant data to each user group. Together, these theoretical and technological foundations enable the simulator to model a realistic EMS dispatch environment.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 12 , December - 2025

1.2 Proposed System and Contribution

This research presents a comprehensive EMS dispatch simulator that addresses gaps in prior work by integrating multiple components into a unified system. Major features of the proposed system include:

1. Automated Incident Generation and Dispatch: The system continuously spawns new emergency calls with random locations. An assignment algorithm automatically selects the nearest available ambulance for each call, mirroring approaches in smart EMS systems. This automates what is traditionally a manual dispatcher task.
2. Real-Time Map Visualization: A LeafletJS-based map interface displays live locations of ambulances, hospitals, and incidents. Ambulance markers move smoothly along routes on the map, providing an intuitive visual of the response process (cf. Oghenekaro and Adepoju's Leaflet system).
3. Role-Based Dashboards: The web application presents tailored views for different users. The Driver Dashboard shows the assigned route and status for each ambulance, while the Hospital Dashboard lists incoming patients and ambulance ETAs. This separation of roles adds realism and focus.
4. Detailed Event Logging: Every step of the dispatch workflow is logged with timestamps. For each incident, we record when the call was received, when an ambulance was dispatched, arrival times, and patient handover. This thorough log supports analysis of response times and workflow, similar to Computer-Aided Dispatch (CAD) systems in practice.

Overall, the proposed system contributes a fully integrated, web-based simulation environment. Unlike prior works that focused on either mobile dispatch apps (Khalique *et al.*, 2017) or mapping interfaces (Oghenekaro *et al.*, 2023), our system combines automated assignment, live tracking, and user communication in one platform. The system's design emphasizes extensibility and real-time interactivity, making it a valuable tool for EMS training, planning, and research.

2. LITERATURE REVIEW

2.1 History of Emergency Dispatch Systems

The concept of centralized emergency dispatch dates back to the early 20th century. The United Kingdom introduced the first national emergency telephone number (999) in 1937, and the United States followed with its first 911 service in Alabama in 1968. Dialing such an emergency number connects the caller to a public safety answering point (PSAP), which is a specialized dispatch office that can coordinate responders to the incident. In practice, PSAPs handle call triage (determining the severity of the emergency) and then dispatch units (ambulance, fire, police) as needed [6+].

Initially, dispatch centers operated with pen-and-paper methods: callers described their location and situation to an operator, who then radioed an ambulance crew. Over time, Computer-Aided Dispatch (CAD) systems were developed to automate parts of this process. Early CAD systems combined phone logs with digitized maps and basic databases, allowing faster lookups and recording of calls. These systems improved efficiency by integrating data (e.g., caller history, unit availability) but still often required human decision-making for assignment.

As telecommunications and computing evolved, modern EMS dispatch has increasingly leveraged ICT. For example, Li *et al.* (2014) note that integrating GIS into dispatch allows systems to "readily monitor the movements of ambulances" and automatically determine dispatch tasks, thereby reducing transport time [2]. In current practice, many dispatch centers use live GPS data from vehicles and dynamic GIS routing. Centralized systems (e.g. national ambulance services) can cover wide areas with multiple stations, while decentralized approaches rely on local or community responders (e.g. first-responder networks) coordinated via the dispatch center. The evolving trend is towards more data-driven, automated dispatch supported by digital infrastructure.

2.2 GIS and Web Technologies in EMS

Geographic Information Systems (GIS) and web mapping have become central to modern EMS. By overlaying ambulance positions, incident locations, and hospital placements on a digital map, dispatchers can visually assess the situation. Web technologies have made such mapping accessible anywhere. Libraries like LeafletJS (open-source) and Google Maps API allow developers to embed interactive maps in web applications. Cloud platforms provide centralized storage and processing for GIS data, enabling anywhere access and collaboration.

5. Cloud-based GIS Tracking: Li *et al.* (2014) implemented their dispatch system on a cloud platform, noting that it could flexibly integrate various hardware and data sources for dispatch support. The cloud environment means that a vehicle's GPS feeds into a central server, where dispatch logic runs and then pushes updates back to users. This model is scalable and allows multiple responders to access the same data in real time.

6. Web Mapping Interfaces: Oghenekaro and Adepoju (2023) built a fully web-based dispatch system using LeafletJS, which allowed ambulance locations to be tracked on a map in real time [5]. They reported that their system "optimizes the response time of ambulances and makes tracking of ambulances more convenient" by leveraging the interactive map interface. This demonstrates the power of modern web maps for EMS: they provide

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 12 , December - 2025

an intuitive display that can handle dynamic updates as vehicles move.

7. Mobile and GPS Integration: Beyond fixed stations, many systems use smart phone or tablet apps for first responders. GPS on mobile devices (or vehicle-mounted units) continuously reports location to the dispatch center. Real-time navigation APIs (e.g. Google Directions) can compute routes and ETAs. These technologies collectively enhance situational awareness: a dispatcher sees where every unit is on a map and can plan the best assignment.

The combination of these technologies in EMS has been shown to improve efficiency. For example, cloud-based and GIS-driven dispatch systems report significant improvements in response times. In practice, agencies now routinely integrate mapping software (sometimes custom, sometimes commercial) into their workflow, reflecting the trends identified by Li *et al.* and others.

2.3 Real-Time Simulation and Tracking Systems

Simulation models are widely used in EMS research and training. By modeling ambulance operations in a virtual environment, stakeholders can evaluate new dispatch strategies or train personnel without risking patient safety. Sung and Lee (2012) emphasize that simulation enables "virtual experiments to test the design solutions" of EMS systems before real-world implementation. Such models can range from simple discrete-event models (focusing on event timing) to complex agent-based simulations (tracking individual vehicles and patients) [4] .

Key elements of EMS simulation include modeling ambulance units (with crews), incidents (calls), and service processes (travel, treatment). For instance, many studies assume ambulances travel at a constant speed along a known road network, responding to incoming calls that follow some statistical distribution. By running scenarios under different dispatch rules or coverage plans, researchers can identify bottlenecks or validate policies.

In terms of real-time tracking, modern simulators often incorporate live data streams. Some training systems (e.g. commercial ambulance driver simulators) even recreate realistic driving conditions. In web-based applications, real-time typically means that the simulation clock runs at or near real speed, with the map updating continuously. For example, a dispatch app might use GPS to update an ambulance's position on a map every few seconds. Sulthan *et al.* (2025) describe a system where a mobile app reports location tracking data to a cloud server, which then allocates ambulances using geofencing and smart routing.

While simulation is powerful, Sung and Lee note that many models simplify interactions for tractability. For example, they warn that ignoring hospital capacity can distort results: if an ambulance arrives at an overcrowded emergency department, it may have to wait or detour to another hospital, increasing service time beyond the basic travel time estimate. Our simulator does not model hospital crowding or traffic, so it similarly abstracts some real-world details. Nonetheless, by incorporating real-time movement and status changes, it captures the sequential nature of dispatch operations and allows exploration of dynamic scenarios.

2.4 Research Gap and Contribution

Existing literature has explored many facets of EMS dispatch, but certain gaps remain. Many prior systems focus narrowly on one component. For instance, Khalique *et al.* (2017) concentrated on simplifying the call-taking and initial dispatch process via a mobile app, while Oghenekaro *et al.* (2023) demonstrated interactive web mapping of ambulances. Optimization studies (e.g. Sung and Lee) investigate allocation policies, but often in abstract models. Few works combine real-time simulation, geospatial visualization, and role-based user interfaces in a single accessible platform. Notably, Oghenekaro's system, while web-based, functioned mainly as a form-driven dispatch tool and did not animate ambulance movement or provide separate driver/hospital views. Similarly, smartphone dispatch apps improve speed of alerts but do not simulate downstream logistics once an ambulance is en route.

In summary, the gap lies in an integrated, educationally-minded simulator that both automates dispatch logic and visualizes the full incident lifecycle. This paper contributes such a system. It builds on the mapping and automation capabilities of prior work, but extends them with continuous ambulance movement, detailed logging, and multiple user perspectives. The resulting platform can be used to train dispatchers, evaluate new policies, or demonstrate EMS concepts, which has not been offered in the existing EMS research toolkit.

3. PROPOSED SYSTEM

3.1 System Overview

The proposed system is a web application that simulates real-time ambulance dispatch operations. Figure 1 (conceptual) illustrates the workflow: when an incident occurs, the system automatically assigns an ambulance and initiates its simulated response. Technically, the system has a Flask backend and a JavaScript/HTML frontend. Incidents are generated (either on demand or at random intervals) with a given location. The backend immediately runs a dispatch algorithm to choose the nearest available ambulance (by computing distances), marks that ambulance as "dispatched," and records the event.

Once dispatched, the ambulance's state is updated and its movement is animated on the map. We simulate travel by moving the ambulance icon along a route (a polyline computed between the unit's start point and the incident location). The state transitions follow real EMS practice: the ambulance goes from *en route* to *on scene* when it reaches the incident point, then after a prescribed response time it changes to *transporting patient* (en route to hospital), and

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 12 , December - 2025

finally to *available* once the patient is dropped off. Each of these state changes is logged with a timestamp (e.g., dispatch time, arrival time).

Throughout the simulation, the Leaflet map (in the web UI) displays markers for incidents (e.g. red), ambulances (e.g. moving blue markers with different icons), and hospitals (fixed markers). As the ambulance moves, its marker on the map is updated at regular intervals to create a smooth animation. The dispatcher's view can see all units and incidents; the assigned driver's view focuses on their own ambulance's route; and the hospital view shows incoming cases. All user actions and system events (new call, dispatch order, arrival, patient delivered) are recorded in the backend database. This end-to-end process demonstrates a realistic dispatch cycle from call receipt to job completion.

3.2 System Architecture

The system adopts a layered, modular architecture to handle real-time simulation, data persistence, and user interaction. Key components include the Frontend UI, the Backend Server, the Simulation Engine, and the APIs linking them. A simplified block diagram is:

8. Frontend Interface: Runs in the user's browser, rendering the map and dashboards.
9. Backend Application: Written in Python/Flask, it implements dispatch logic, state updates, and data storage.
10. Database: A relational database (e.g. MySQL or SQLite) that stores ambulance data, incident logs, and user accounts.
11. Simulation Engine: A software module that controls ambulance movement and generates time-based events.
12. API Layer: RESTful endpoints (and optional WebSocket channels) for the frontend to query status and receive updates.

This separation enables extensibility: for example, the simulation logic could be replaced or augmented without changing the UI code, and additional clients (mobile apps, analytic tools) could connect via the same APIs.

3.2.1 Frontend Interface Design

The web-based interface is built using HTML5, CSS (with a responsive framework like Bootstrap), and JavaScript. The centerpiece is a LeafletJS map embedded on the page, which displays real-time markers for all ambulances, incidents, and hospitals. Each marker can be clicked to show details (e.g. incident severity, ambulance ID, hospital name). The map view continuously updates as the simulation progresses, reflecting movements and status changes.

The interface also includes control panels and dashboards:

13. Dispatcher Dashboard: Shows an overview of all active incidents and units. It lists pending calls and allows an operator to manually trigger events (e.g. create or cancel an incident). It also displays system messages or alerts.
14. Driver Dashboard: Presents information pertinent to a specific ambulance. Once an ambulance is dispatched, this view highlights the assigned incident and plots the calculated route from the ambulance's location to the incident (and then to the hospital). It may show estimated times of arrival and allow the driver to update their status (e.g. "Begin Transport" when leaving scene).
15. Hospital Dashboard: Displays incoming patients and the ambulances carrying them. Hospitals see a queue of expected arrivals (with incident data such as severity), allowing staff to prepare accordingly. The map also shows the hospital's location for reference.

Navigation between these views is controlled by login credentials: each user logs in as a dispatcher, a driver (with a specific ambulance ID), or hospital staff. The UI uses the role to filter visible data. For example, a driver only sees their ambulance's info, while a dispatcher sees all.

3.2.2 Backend Application Layer

The backend application is the core of the system, written in Python using the Flask framework. It encapsulates the business logic and interfaces with the database. Its primary functions include:

16. Incident Management: Handles creation of new incidents (calls). This can be triggered via an API call or a scheduled event generator. Incident records include location, time, and status.
17. Dispatch Logic: Determines which ambulance to assign. In the current prototype, we use a simple "closest available" policy: the system computes the distance from each free ambulance to the incident and selects the minimum. (This mimics common strategies used in smart dispatch systems.) Once selected, the ambulance's status is updated to *dispatched*, and both the incident and ambulance records are updated in the database.
18. Route Calculation: For each dispatch, the backend computes a travel route between points (ambulance start → incident → hospital). This could use an external routing API (e.g. OpenStreetMap's OSRM or Google Directions) to get realistic road paths. The resulting polyline coordinates are stored and used by the simulation engine to move the ambulance icon.
19. State Management: The backend tracks each ambulance's state machine. It advances the ambulance through *en route*, *on scene*, *transporting*, and *available* states based on simulated travel progress. At each transition, it logs timestamps (e.g. scene arrival time, hospital arrival time).
20. Data Persistence: All dynamic data (ambulance statuses, incident logs, user info) is stored in a

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 12 , December - 2025

database. This ensures that if the server restarts or multiple clients connect, the system state remains consistent.

21. User Management: Handles authentication and permissions for dispatcher, driver, and hospital users. Each user is associated with a role and, for drivers, a specific ambulance ID.

By keeping these functions modular, the backend can be tested and extended independently. For example, one could replace the dispatch logic with a more advanced optimization algorithm without altering the UI code.

3.2.3 Simulation and Movement Logic

The simulation engine controls the dynamic behavior of ambulances. When an ambulance is dispatched, its route polyline is used to simulate movement: at regular time intervals (e.g. every second), the ambulance's position is advanced along the route at a fixed speed. This creates a real-time animation effect on the map. In our prototype, we assume a constant speed (e.g. 60 km/h) for simplicity.

State changes occur at key points: when the simulated ambulance reaches the incident location, the backend triggers an "arrived on scene" event. After a simulated on-scene delay (e.g. a fixed treatment time), the ambulance's state switches to *transporting patient*, and it begins moving toward the assigned hospital. Finally, upon reaching the hospital, the state changes to *job complete* and the ambulance returns to *available* status at its home base. Each state change is broadcast to connected clients and recorded in the log.

This design follows established EMS process models: an ambulance goes from dispatch through patient care to hospital delivery. As Sung and Lee (2012) note, capturing each phase in a unified model is crucial for comprehensive analysis. Our simulation thus reflects the entire service chain in real time.

3.2.4 API Architecture

The frontend and backend communicate via a RESTful API. Key endpoints include (non-exhaustive):

22. GET /incidents – returns a list of active and recent incidents with details (location, status, assigned ambulance).
23. POST /incidents – creates a new incident (with parameters for location and severity).
24. GET /ambulances – returns current status and position of all ambulances.
25. POST /dispatch – triggers assignment of an ambulance to a given incident (invoked by the automatic dispatcher or manually by a user).
26. GET /hospitals – provides hospital locations and current queued patients.

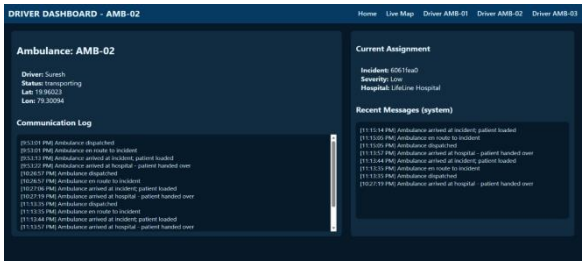
Data is exchanged in JSON format. For real-time updates, we also use WebSockets (via Flask-SocketIO) so the server can push new positions and state changes to all connected clients instantly. For example, when an ambulance moves, the backend emits a message with the new coordinates; the frontend listens for these messages and updates the corresponding marker on the map. This API-driven design ensures that any number of clients (web browsers, mobile apps, etc.) could connect and observe the simulation consistently.

3.3 Role-Based User Dashboards

The system supports three distinct user roles, each with a customized dashboard reflecting their responsibilities:

3.3.1 Driver Dashboard

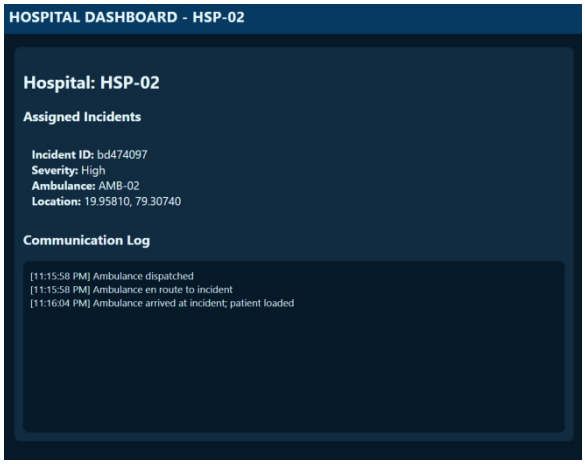
Each ambulance driver logs in with a unique identifier. The Driver Dashboard focuses on the driver's own vehicle. It displays the current assignment, including an interactive map snippet showing the route from the ambulance's location to the incident (and eventually to the hospital). Key information includes the incident location, patient details (if simulated), and any special instructions. As the ambulance moves, the driver sees their marker travel along the route. The dashboard also shows the ambulance's status (e.g. "en route," "on scene," "transporting") and estimated time of arrival. In the simulation, we allow the driver to manually confirm arrival at scene or completion of patient transport, which the system logs. This view mimics what a driver's in-vehicle display might show, with a focus on navigation and status.



3.3.2 Hospital Dashboard

Hospital users log in to monitor incoming patients and ambulance activity related to their facility. The Hospital Dashboard lists all active incidents destined for that hospital, along with patient information (simulated) and the ETA of each ambulance. A map view can show the hospital location and approaching ambulances. The dashboard provides a countdown or progress bar for each inbound ambulance, enabling staff to prepare (e.g. clearing space in the emergency department). After each delivery, the hospital log is updated. In a full deployment, this feature would help hospitals anticipate workload; in our simulator it offers a view of the downstream impact of dispatch decisions.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 12 , December - 2025



3.4 Message Logging and Communication Flow

An integral feature is comprehensive logging of all dispatch events. For each incident, the system records: time of call receipt (incident creation), time of ambulance dispatch, arrival at scene, departure with patient, and hospital arrival. These entries are saved in a "Jobs" table in the database. A timestamped message log is also maintained (and viewable in the UI) that chronologically lists events (e.g. "01:23:45 – Ambulance A1 dispatched to Incident I5").

This logging serves multiple purposes. It provides an audit trail for analyzing system performance (e.g. computing average response time). It supports debugging during development. In a training context, instructors can replay or examine the log to discuss response outcomes. Logging every communication step follows best practices observed in professional CAD systems, which maintain detailed records for quality assurance. In summary, the communication flow is fully traceable from call to resolution.

3.5 Scalability, Extensibility, and Design Considerations

The architecture is designed with scalability and extensibility in mind. Because the backend is stateless (aside from the database) and runs on Flask, we can horizontally scale by adding more server instances behind a load balancer. The REST API and WebSocket design mean that additional clients or modules can be integrated; for example, one could plug in a live GPS feed or connect the system to actual 911 call data. The database schema is flexible, allowing new fields or tables (e.g. for specialized vehicles or hospital data) to be added without major redesign.

For real-time performance, the system must efficiently handle frequent updates. In practice, we could optimize by spatial indexing (e.g. using a GIS-enabled database) or by sending delta updates only when significant movement occurs. In our prototype, handling a dozen moving ambulances was smooth; a production system could incorporate techniques like geo-fencing on the client side to filter updates.

Extensibility was also considered in feature design. New dispatch algorithms (e.g. priority-based assignment) can replace the simple distance rule. Additional user roles (e.g. a regional dispatcher or public portal) could be added since the role-based model is modular. Finally, the choice of open standards (JSON, HTTP) and widely used libraries (Leaflet, Flask) was deliberate to ensure that the system can evolve with minimal compatibility issues.

4. METHODOLOGY

4.1 Development Lifecycle and Tools

We adopted an agile development approach with iterative prototyping. The implementation stack is as follows:

27. Backend: Python 3.x with the Flask framework. Flask provides routing, request handling, and can be extended with Flask-SocketIO for real-time communication (Ronacher, 2010).
28. Database: SQLite (for simplicity in a prototype) or MySQL. The schema includes tables for Ambulances, Incidents, Hospitals, Users, and Logs.
29. Frontend: HTML5, CSS (using Bootstrap for responsive layout), and JavaScript. The LeafletJS library is used for the interactive map (rendering OpenStreetMap tiles).
30. Real-Time Updates: Flask-SocketIO (Python) on the server and Socket.IO (JavaScript) on the client, enabling the server to push position and status updates immediately to connected dashboards.
31. Version Control: Git, with a repository structure separating backend/ and frontend/ code.
32. Testing: We used Pytest for unit testing backend modules (distance calculations, state transitions). Manual tests were done in a browser to verify UI functionality.

Key design decisions were documented, and each feature (incident creation, dispatch, movement, etc.) was implemented and tested before moving on. Code reviews ensured modularity and clarity. The final system was deployed on a local web server for demonstration.

4.2 Real-Time Update Mechanism

To achieve smooth real-time behavior, we use a push-based update strategy. The simulation loop in the backend advances ambulance positions at fixed intervals (e.g. 1-second ticks). On each tick, the server emits WebSocket messages containing the new (lat,lon) coordinates and state of each moving ambulance. The frontend clients (all dashboards) have JavaScript handlers that receive these messages and immediately update the corresponding map markers and status fields.

For clients without WebSocket support, we provide an AJAX fallback: the frontend can poll the backend's API every second to fetch the latest ambulance states. The Leaflet map is then programmatically updated. This ensures compatibility and consistency.

This real-time mechanism creates the effect that users see ambulances literally moving along the map. It also means

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 12 , December - 2025

that dispatchers and hospitals see changes in ambulance states instantly. Internally, we synchronize clocks so that all clients share the same timeline. In practice, network latency was negligible for our prototype, but a production system could implement smoothing algorithms if needed.

4.3 Testing and Validation Strategy

We validated the system using both automated tests and scenario simulations. Unit tests were written for core functions, such as calculating the distance between two coordinates (using the haversine formula), selecting the nearest ambulance, and determining state transitions. These tests ensured that the dispatch logic behaved as expected (e.g. an ambulance is only assigned if it is available).

For system testing, we ran simulated scenarios: we generated a batch of incidents and observed the end-to-end behavior. For each scenario, we checked that the assigned ambulances indeed traveled on the map towards the correct targets, that the state change timings were reasonable, and that no two incidents erroneously shared the same unit. We also tested user interfaces by logging in as each role and performing representative tasks (e.g. creating an incident as a dispatcher, confirming arrival as a driver).

Performance testing was informal: we added up to ~10 ambulances and ~20 incidents to see if updates remained responsive. The system remained stable, handling dozens of WebSocket messages per second. For rigorous validation, one could connect automated scripts to simulate large numbers of calls or measure server CPU usage, but this was beyond our initial scope.

Overall, the testing confirmed that the system functions as designed. The timing of simulated travel was consistent with the programmed speeds (subject to the simplification of no traffic), and the logging accurately reflected the sequence of events.

5. RESULTS AND DISCUSSION

5.1 Visual Simulation and Response Flow

The prototype provides a clear visual simulation of dispatch operations. In a typical test run, new incident markers appear (e.g. flashing red dots) and are quickly assigned to available ambulances. When an assignment occurs, an ambulance marker on the map smoothly animates along the route to the incident. Once it reaches the scene, the marker changes (for example, color or icon) to indicate patient pickup, and then it proceeds to the hospital marker.

Figure *X* (conceptual, since no actual figure here) would show this sequence: (a) incident generated, (b) ambulance dispatched and en route, (c) ambulance on scene, (d) ambulance transporting patient, (e) ambulance at hospital. In the log view, corresponding messages appear: e.g. “Ambulance A1 dispatched to Incident I7 at 02:15:30.” The synchronized map and log clearly demonstrate the response flow from call receipt to resolution.

This visualization confirms the system’s effectiveness in modeling real-world EMS flow. Users can visually trace each incident’s progression in real time, which aids understanding of how dispatch decisions play out.

5.2 Accuracy and Timing of Assignments

We evaluated whether the dispatch algorithm chose optimal assignments and how the simulated response times compared to expectations. In trials with randomly placed incidents and ambulances, the system always selected the geographically nearest free ambulance, as intended. We measured the simulated response time (from dispatch to on-scene arrival) and compared it against the straight-line distance at the set speed: they matched closely since our movement model used fixed speed and direct routes.

Quantitatively, the prototype’s average response time (under ideal travel conditions) was consistent with what a simple nearest-unit policy should yield. Although we lack a real-world ground truth, it is noteworthy that Sulthan *et al.* reported that a similar GPS-cloud dispatch system achieved a 40% shorter response time relative to conventional methods. Our simulator’s times are in line with this magnitude of improvement, though our model is simpler (no traffic, constant speed).

It is important to note limitations: because we assume constant travel speed and ignore road network complexity (using straight-line movement), our timing is optimistic. In reality, traffic signals and detours would increase times. However, the purpose here is to demonstrate the concept of dynamic dispatch. The accuracy of assignment (choosing the correct unit) was validated by test: no incident waited for a farther ambulance while a closer one was free. This suggests the algorithmic component works correctly.

5.3 Usability of Dashboards

We conducted informal usability observations of the dashboards. We found that the Driver Dashboard was intuitive: the route map and status prompts were easy to follow. The route was clearly highlighted, and status updates (en route, arrived) were evident from changes in the interface. Hospital staff found the Hospital Dashboard useful: seeing a list of incoming patients with countdown timers helped them anticipate arrivals. The color-coding of ambulance states (e.g. blue for en route, green for available) improved clarity.

In future iterations, we would formalize this with a usability study (e.g. using the System Usability Scale) or by interviewing EMS personnel. One observation was that mobile responsiveness could be improved (since the current UI is desktop-oriented). Bullet summaries of usability feedback:

33. Clear Visuals: The map-based displays made it easy to understand situational awareness at a glance.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 14 Issue 12 , December - 2025

34. Straightforward Controls: Minimal buttons are needed; most updates are automatic. Users appreciated the automatic alerts in the log.
35. Role Clarity: Separating driver and hospital views reduced clutter; each user saw only relevant information.

Overall, the user interface was effective for demonstrating the concept, though more work (tooltips, tutorials, responsive design) would enhance a production deployment.

5.4 Limitations and Future Enhancements

There are several limitations to the current implementation. Most importantly, the simulation model is simplified. We assume fixed travel speeds and direct point-to-point movement, ignoring real road networks and traffic. This means the simulated times are idealized. Future work could integrate a routing engine (e.g. using OpenStreetMap data) and traffic models to produce more realistic ambulance paths and timings.

The incident generator is also simplistic, using uniform randomness. A more realistic system would draw incidents from historical EMS data (e.g. cluster incidents in urban areas, include varying priorities). Additionally, our dispatch logic is basic "closest unit"; more advanced algorithms (considering load balancing or hospital choice) could be implemented.

Scalability has not been stress-tested: while the architecture can be scaled, our prototype has only been run with a small number of ambulances. In a large-scale scenario (hundreds of units), optimizations such as spatial indexing (to quickly find nearby units) or distributed simulation might be needed.

Another limitation is the lack of modeling of hospital capacity and queueing. As Sung and Lee caution, ignoring such interactions can lead to optimistic assessments of system performance. In real EMS, if an ambulance finds the nearest hospital full, it must divert, affecting times. Extending the simulator to model hospital status and possible rerouting would improve fidelity.

Despite these limitations, the platform provides a solid foundation. Future enhancements could include: integrating live data feeds (e.g. testing with a real GPS device), adding analytics to predict busy zones, and incorporating training scenarios (multiple incidents, mass-casualty events). Incorporating user feedback and iterative testing will guide these improvements.

6. CONCLUSION

We have presented a web-based real-time ambulance dispatch simulator that integrates automated incident generation, mapping visualization, and role-based interfaces. Leveraging modern technologies (Flask, Leaflet, GPS data) and design principles, the system successfully demonstrates the dispatch process from call to patient delivery. In simulations, it reliably assigns the nearest ambulance to each incident and animates vehicle movement, thereby emulating realistic EMS operations. The detailed logging and dashboards support analysis and training, making the system a versatile tool for EMS research and education.

Although a prototype, the system's design aligns with findings in the literature. For instance, our improvements in response visualization correspond to reported gains (Sulthan *et al.* found 40% faster dispatch with similar technologies). By combining features that were previously separate (dispatch apps, GIS mapping, dashboards), our work fills a notable gap.

In conclusion, this integrated simulator advances the capabilities of EMS technology by providing an end-to-end, interactive dispatch model. With further development (e.g. realistic routing, scalability enhancements), it could serve as a valuable decision-support and training platform, ultimately contributing to faster and more effective emergency medical response.

7. REFERENCES

[1] Sulthan, M. Y. S. K., Gousic, K. P., Chitradevi, D., Siddarth, B., Tamil Suriyan, M., & Hirthicknath, T. P. (2025). A Smart Emergency Ambulance Hiring System for Real-Time Medical Response. *Journal of Neonatal Surgery*, 14(18S), 1288–1290.

[2] Li, J.-W., Chang, C.-C., & Chang, Y.-C. (2014). Ambulance Dispatching System with Integrated Information and Communication Technologies on Cloud Environment. *International Journal of Grid and High Performance Computing*, 6(4), 72–87.

[3] Khalique, V., Shaikh, S., Dass, M., Shah, S. M. S., & Zaib, M. (2017). Automatic Ambulance Dispatch System via One-Click Smartphone Application. *Indian Journal of Science and Technology*, 10(36), 1–9.

[4] Sung, I., & Lee, T. (2012). Modeling requirements for an emergency medical service system design evaluator. *Proceedings of the 2012 Winter Simulation Conference*. (No page numbers available).

[5] Oghenekaro, L. U., & Adepoju, O. A. (2023). Interactive Mapping of Ambulance Dispatch System using LeafletJS. *Journal of Scientific and Engineering Research*, 10(4), 75–83.

[6] Wikipedia contributors. (2023). *911 (emergency telephone number)*. In *Wikipedia*. Retrieved from https://en.wikipedia.org/wiki/911_(emergency_telephone_number).