WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

Ambulance Emergency Resource Allocation System: A Web-Based Decision-Support Prototype for Emergency Hospital Selection

Nilavan M, Pugalesh P, Kabilesh Sakthi S, *Mrs. Saranya M

*Student's, *Assistant Professor*
Department of Information Technology,Sri Shakthi Institute of Engineering and Technology, Coimbatore-641062, Tamil Nadu, India.

ABSTRACT

During a medical emergency, the receiving hospital is often chosen by telephone or by habit, without a shared view of free beds and specialised units. A hospital that is nearest may be full or may lack the unit that the patient needs, and the resulting re-transfer costs time. This paper presents the Ambulance Emergency Resource Allocation System, implemented as Crisis-Grid, a web-based decision-support prototype that selects a receiving hospital by combining the ambulance position, an estimate of severity and required specialty, and live bed and specialty-unit availability. The back end uses Python with FastAPI; a rule-based triage module assigns a severity from 1 to 5, an intent extractor reads location and urgency from typed or spoken text, and a forecasting module projects free beds at one, three and six hours. A weighted score ranks the hospitals that have a free bed, and the selected bed is reserved inside an asyncio lock so that concurrent requests cannot take the same last bed. Hospital state and incidents are stored in SQLite and every change is pushed to all browsers over WebSocket. A React interface provides capacity, SOS, free-text dispatch, map and incident-log screens, with road routes obtained from OSRM. The prototype works on four seeded hospitals and a simulated ambulance movement. Executing the intelligence modules showed that the keyword rules miss some wordings, such as "acute crushing chest pain", so clinical validation, authentication and real data integration are required before any operational use.

Keywords: Emergency Medical Services, Ambulance Dispatch, Hospital Selection, Triage, Resource Allocation, FastAPI, WebSocket, Geospatial Routing.

1. INTRODUCTION

Choosing the receiving hospital is the last decision in a chain that begins with an emergency call and ends with patient handover. The choice depends on information held by different parties: the paramedic knows the clinical picture, the crew knows the position, and each hospital knows its free beds. When these pieces are not combined, a patient may be taken to a hospital that is full or that lacks the required unit, such as a catheterisation laboratory or a burn unit, and may have to be transferred again.

Operations research has long studied where ambulances should be based and which vehicle should be sent [1], [2], [3]. Triage systems such as the Emergency Severity Index and the Manchester Triage System assign patients to five urgency levels [4], [5]. Routing engines compute road routes over open map data [6], [7]. The destination decision that joins these elements at the moment of an emergency receives less attention in application-level software.

1.1 Existing challenges in hospital selection

Common practice includes telephone or radio coordination, navigation to the nearest hospital, and bed dashboards that are visible only inside one hospital. These methods leave no shared live view, ignore specialty availability, allow two crews to be promised the same last bed, and produce little structured record. Distance-only selection is particularly weak for time-critical conditions in which the available unit matters more than the shortest road.

1.2 Role of triage, geospatial analysis and real-time web technology

Triage supplies the urgency and the specialty need. Geodesic distance, computed on an ellipsoid [8], [9], gives a consistent estimate of proximity, and road routes show the real path. A persistent WebSocket channel [10] lets every screen see the same capacity at the same time. The proposed system combines these elements in one application, but it presents its triage and forecasting as simple rules and states that they do not replace clinical judgement.

1.3 Problem statement and objectives

There is a need for a system that accepts the ambulance location, the patient's condition and the live hospital capacity, selects a suitable hospital by a transparent rule, reserves the bed safely under concurrent requests, and records the decision. The specific objectives are:

i. To accept the ambulance position by manual entry, regional presets or browser geolocation.

ii. To infer severity (1–5) and required specialty from symptoms and vital signs.

iii. To extract location, urgency and specialty from typed or spoken dispatch text.

iv. To score hospitals with a free bed and reserve a bed without over-subscription.

v. To forecast bed availability for the next one, three and six hours.

vi. To store hospital state and incidents in SQLite and broadcast changes over WebSocket.

vii. To show the allocated hospital, the road route and an incident log.

viii. To evaluate the prototype by screenshots, module execution and test cases, and to report limitations.

2. MATERIALS AND METHODS

2.1 Development approach

The project followed an iterative approach [11]: an allocation endpoint and form were built first, and specialty matching, vital signs, persistence, the incident log, free-text dispatch, forecasting, WebSocket synchronisation and the map were added in later iterations. Each iteration was checked by automated tests for back-end logic and by manual use for the interface.

2.2 System architecture

The system is a layered client–server application (Fig. 1). The presentation layer is a React single-page application with five screens. The application layer is a FastAPI service with REST endpoints, one WebSocket endpoint, request validation and the allocation engine. Three intelligence modules (triage, intent extraction and forecasting) are plain Python functions called by the service. The data layer holds an in-memory working copy of the hospitals and an SQLite database. The browser calls the external OSRM server and OpenStreetMap tile servers directly.

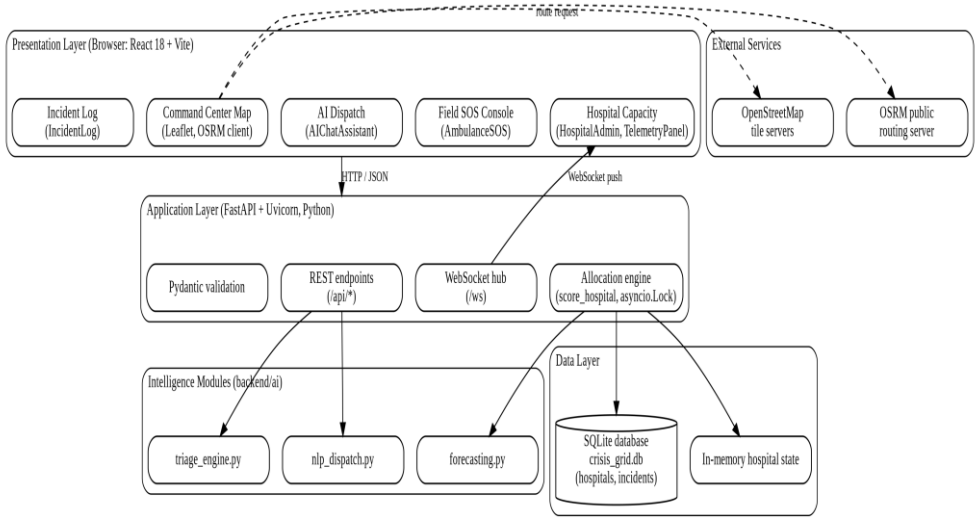


Fig. 1. System architecture of the proposed Ambulance Emergency Resource Allocation System.

2.3 Technology stack and development environment

Layer	Technology	Responsibility
Front end	React 19, Vite, Tailwind CSS [12]	Five-screen interface, styling and build
Maps and charts	Leaflet, React Leaflet, Recharts [13]	Interactive map, route and bed charts
Back end	Python, FastAPI, Uvicorn, Pydantic [14]	REST and WebSocket endpoints, validation
Distance	geopy geodesic [9]	Ambulance-to-hospital distance
Storage	SQLite in WAL mode [15], [16]	Hospitals and incidents
Routing	OSRM public server, OpenStreetMap [7], [17]	Road route, distance, duration

Table 1. Technology stack and responsibilities.

2.4 Data collection and processing

The prototype uses four seeded hospitals located at the centres of Chennai, Coimbatore, Madurai and Trichy, with 20, 15, 10 and 25 beds and 5, 2, 0 and 8 free beds respectively, together with counts for five specialty units (ICU, Cath Lab, Burn Unit, Neuro ICU and Trauma). These are sample values created for the prototype and are not taken from real hospitals. An allocation request contains the ambulance latitude and longitude, symptom text, severity from 1 to 5 and optional pulse, systolic pressure, SpO2 and temperature. Pydantic rejects values outside the valid ranges before processing, and all SQL statements use parameter placeholders.

2.5 Database design

Two tables are used. The hospitals table stores id, name, coordinates, total and available beds and a JSON text of specialty counts. The incidents table stores the identifier, time, ambulance identifier and position, symptoms, severity, vital signs, allocated hospital, required specialty, whether it was matched, estimated time, distance and status (default DISPATCHED). The relation between incidents and hospitals is logical; no foreign key is declared. At start-up the stored hospitals are used if their number equals that of the defaults; otherwise the defaults are loaded, and if the database fails the service continues with in-memory data.

2.6 Triage and intent extraction

The triage function starts from the reported severity and can only raise it. Symptom keywords raise it to 3, 4 or 5, and vital-sign thresholds raise it as shown in Table 2. The required specialty is the first match among Neuro ICU, Cath Lab, Burn Unit, Trauma and ICU keyword lists; without a match, the default is Trauma for severity 3 or higher and ICU otherwise. The thresholds are illustrations chosen by the developers and are not clinically validated.

Vital sign	Condition	Minimum severity
Pulse	> 130 per minute	4
Systolic pressure	< 90 mmHg	5
Oxygen saturation	< 90 %	5
Temperature	> 39.5 °C	4

Table 2. Vital-sign rules in the triage function.

The intent extractor reads explicit latitude and longitude with regular expressions and validates their range; otherwise it searches for the city keywords Guindy, Chennai, Coimbatore, Madurai and Trichy and defaults to Chennai. Urgency (2–5) and specialty are obtained from keyword lists. The reported confidence is a fixed constant, not a computed probability.

2.7 Allocation algorithm and concurrency control

Only hospitals with at least one free bed are scored. Let d be the geodesic distance in km, $t = (d/30)\times 60$ the estimated minutes at 30 km/h, a the free beds, n the total beds and s the severity. The score, in which a lower value is better, is

score = 0.45*t + 0.2*cap_pen + 0.75*spec_pen + (10/max(a,1))*(6-s) + 0.15*d/(1+a/n)

where the capacity penalty is 100 when no bed is free (never active, because such hospitals are excluded) and the specialty penalty is 30 when the required unit has no free place. Then 25 is subtracted if the required specialty is available and 10 is added if a ≤ 1. The minimum score is selected, the bed and specialty counts are decremented and the hospital is saved. The whole selection runs inside an asyncio lock, so two simultaneous requests cannot take the same last bed; the incident is logged and the update broadcast after the lock is released. The lock protects a single server process only.

2.8 Real-time synchronisation and mapping

After each allocation, bed change or reset, the server sends the full hospital list with forecasts to all WebSocket clients; the browser reconnects with a delay that grows by a factor of 1.5 up to 15 seconds. The forecast uses a net-flow rule with a discharge rate of 1.2 and an arrival rate of 1.0 beds per hour [18], and flags High Risk of Capacity Strain when utilisation exceeds 75 % or the one-hour projection is 2 or lower. The command-center map requests an OSRM driving route and falls back to a straight geodesic line if the request fails. The ambulance movement along the route is simulated in the browser; no vehicle position is received.

2.9 Analysis workflow

The workflow (Fig. 2) starts when the user submits a structured SOS or a text dispatch. The input is validated, triaged and scored, the bed is reserved under the lock, and the result is logged and broadcast. The map then shows the route, and the incident log lets the user move the status through DISPATCHED, EN_ROUTE, ARRIVED and RESOLVED. If no hospital has a free bed, the service returns an HTTP 400 error that is shown to the user.

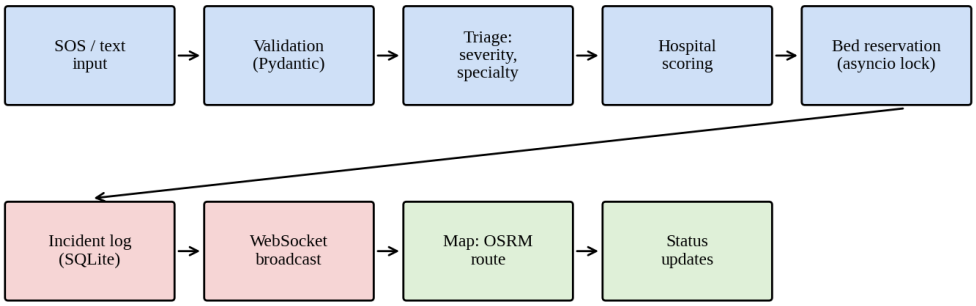


Fig. 2. Workflow of the proposed system.

2.10 Testing and evaluation

The repository contains Python unit and integration tests (coordinate parsing, validation, standard and concurrent allocation, specialty decrement, vital-sign effects and persistence) and a Node test for the front-end error formatter. Inputs were chosen by equivalence classes and boundary values [19]. For this paper the three intelligence modules were executed directly on sample inputs, the Node test was run, and the interface was examined through screenshots. The Python integration tests could not be executed while this paper was prepared, and their results are marked as pending. No performance, accuracy or user-feedback figures were measured.

3. RESULTS AND DISCUSSION

3.1 Implemented system functionality

The interface has five screens: Hospital Capacity, Field SOS Console, AI Dispatch, Command Center Map and Incident Log. Fig. 3 shows the header indicators and hospital cards: 13 free beds out of 70 (81 % utilisation), with Madurai flagged at critical capacity. The risk badges and projections agree with the forecasting rule; for example, a hospital with four free beds out of 20 has 80 % utilisation and is flagged, and its projections of 4, 5 and 7 beds match the formula (4.2, 5.2 and 7.2 before rounding).

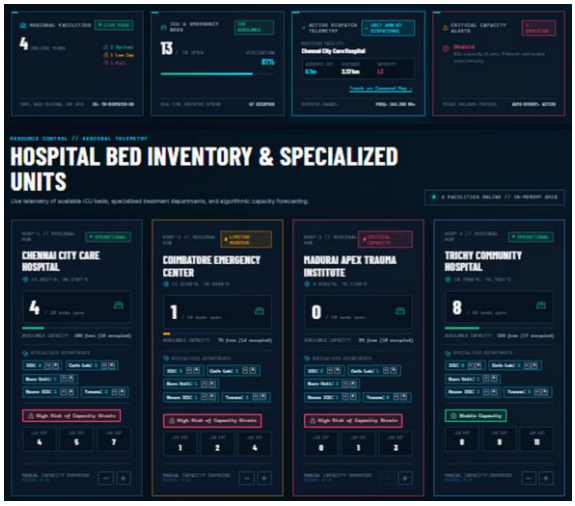


Fig. 3. Header indicators and hospital capacity cards of the running application.

Fig. 4 shows the command-center map for a moderate Trauma dispatch near Guindy, Chennai. The route to Chennai City Care Hospital has a geodesic distance of 3.37 km (estimated 6.7 minutes at 30 km/h) and an OSRM road distance of 4.2 km, and the unit status is labelled as simulated telemetry.

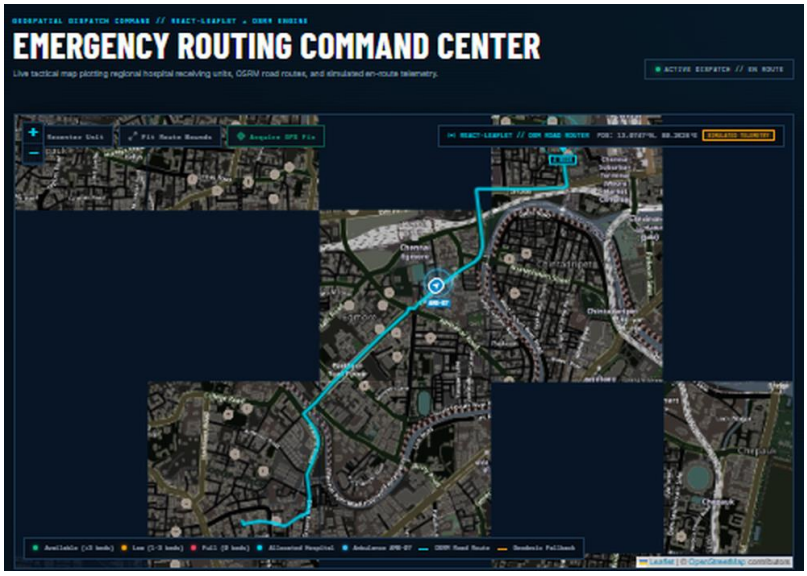


Fig. 4. Command center map with the OSRM road route for an allocated dispatch.

3.2 Functional evaluation

Table 3 summarises the test scenarios. Of 29 defined cases, 12 passed on observed evidence (seven by screenshot and five by direct execution), three failed, seven automated Python tests await execution, and seven have not been tested.

ID	Module	Test scenario	Result
T1	Interface	Five screens show the expected elements (screenshots)	Pass
T2	Allocation display	Moderate Trauma SOS shows hospital, rationale and route	Pass (screenshot)
T3	Triage	Eleven threshold and keyword inputs	Pass (11 of 11)
T4	Intent extraction	Burn and trauma prompts give correct specialty	Pass
T5	Forecast	Seeded counts give 5/6/8, 2/3/5, 0/1/3, 8/9/11	Pass
T6	Error formatter (Node)	Six sample error objects	Pass (6 of 6)

ID	Module	Test scenario	Result
T7	Triage wording	“acute crushing chest pain” expected Cath Lab	Fail (Trauma)
T8	Triage wording	Stroke described by facial drooping expected Neuro ICU	Fail (Trauma)
T9	Intent extraction	Facial drooping text expected Neuro ICU	Fail (ICU)
T10	Concurrent allocation	Two requests for the last bed	Pending (test exists)
T11	WebSocket, OSRM fallback, restart	Multi-client update, offline route, persistence	Not tested

Table 3. Representative functional test results.

3.3 Comparison with conventional approaches

Conventional coordination relies on telephone calls, navigation to the nearest hospital or hospital-internal dashboards. The prototype adds a shared live capacity view, specialty-aware scoring, an atomic bed reservation, an explanation of each decision and an audit log. These are design properties; the system has not been compared with other tools or used with real hospitals, so it should not be considered more accurate or faster than existing practice.

3.4 Discussion

The results show a coherent prototype whose displayed values are consistent with the formulas in the code. The main finding of testing is the brittleness of keyword rules: a fixed phrase such as “acute chest pain” is matched but “acute crushing chest pain” is not, and a stroke described by its signs is not recognised. In the free-text path, the extracted specialty is not passed to the allocation function, which runs its own triage, so the two can disagree. Other limitations are the fixed vital values added to text dispatches, recognition of only five place names, a single ambulance identifier, straight-line scoring at a constant 30 km/h, constant forecast rates, a one-second incident-identifier resolution, silent database write errors, open CORS and the absence of authentication [20]. Future evaluation should include the pending integration tests, clinically reviewed scenarios and tests with several simultaneous users.

4. CONCLUSION

The Ambulance Emergency Resource Allocation System demonstrates that a conventional web stack can combine severity estimation, specialty matching, live capacity, a lock-protected bed reservation and mapping into one tool for choosing an emergency destination. It is a prototype on seeded data and a decision-support illustration, not a clinical system.

Future work includes richer phrase rules followed by learned triage models, using one specialty decision across screens, authentication and restricted access, real vehicle telemetry for several ambulances, integration with hospital information systems, database-level atomic reservation, forecasting from admission history and road-distance-based scoring. Evaluation with real data and clinical review is required before stronger claims can be made.

5. ACKNOWLEDGMENTS

The authors thank Sri Shakthi Institute of Engineering and Technology for the academic environment and facilities, and express their gratitude to the Head of the Department of Information Technology, the project guide Mrs. Saranya M, and the faculty members for their guidance and support throughout the project.

6. REFERENCES

1. C. Toregas, R. Swain, C. ReVelle, and L. Bergman, “The location of emergency service facilities,” Operations Research, vol. 19, no. 6, pp. 1363–1373, 1971.

2. L. Brotcorne, G. Laporte, and F. Semet, “Ambulance location and relocation models,” European Journal of Operational Research, vol. 147, no. 3, pp. 451–463, 2003.

3. V. Schmid, “Solving the dynamic ambulance relocation and dispatching problem using approximate dynamic programming,” European Journal of Operational Research, vol. 219, no. 3, pp. 611–621, 2012.

4. N. Gilboy, P. Tanabe, D. Travers, and A. M. Rosenau, Emergency Severity Index (ESI): A Triage Tool for Emergency Department Care, Version 4, Implementation Handbook, 2012 ed. Rockville, MD, USA: Agency for Healthcare Research and Quality, AHRQ Pub. No. 12-0014, 2011.

5. K. Mackway-Jones, J. Marsden, and J. Windle, Eds., Emergency Triage: Manchester Triage Group, 3rd ed. Chichester, U.K.: Wiley-Blackwell/BMJ Books, 2014.

6. D. Luxen and C. Vetter, “Real-time routing with OpenStreetMap data,” in Proc. 19th ACM SIGSPATIAL Int. Conf. Advances in Geographic Information Systems, 2011, pp. 513–516.

7. Project OSRM, “Open Source Routing Machine.” [Online]. Available: https://project-osrm.org/

8. C. F. F. Karney, “Algorithms for geodesics,” Journal of Geodesy, vol. 87, no. 1, pp. 43–55, 2013.

9. GeoPy Contributors, “GeoPy documentation: geopy.distance.” [Online]. Available: https://geopy.readthedocs.io/

10. I. Fette and A. Melnikov, “The WebSocket Protocol,” Internet Engineering Task Force, RFC 6455, Dec. 2011. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6455

11. I. Sommerville, Software Engineering, 10th ed. Harlow, U.K.: Pearson, 2016.

12. Meta Platforms, Inc., “React documentation.” [Online]. Available: https://react.dev/

13. V. Agafonkin, “Leaflet: an open-source JavaScript library for interactive maps.” [Online]. Available: https://leafletjs.com/

14. S. Ramírez, “FastAPI documentation.” [Online]. Available: https://fastapi.tiangolo.com/

15. SQLite Consortium, “SQLite documentation.” [Online]. Available: https://www.sqlite.org/docs.html

16. SQLite Consortium, “Write-Ahead Logging.” [Online]. Available: https://www.sqlite.org/wal.html

17. OpenStreetMap Foundation, “OpenStreetMap copyright and license.” [Online]. Available: https://www.openstreetmap.org/copyright

18. R. J. Hyndman and G. Athanasopoulos, Forecasting: Principles and Practice, 3rd ed. Melbourne, Australia: OTexts, 2021. [Online]. Available: https://otexts.com/fpp3/

19. G. J. Myers, C. Sandler, and T. Badgett, The Art of Software Testing, 3rd ed. Hoboken, NJ, USA: Wiley, 2011.

20. OWASP Foundation, “OWASP Top Ten.” [Online]. Available: https://owasp.org/www-project-top-ten/