

Optimizing Care Delivery: A Contemporary Approach to Hospital Load Distribution and Queue Management

Priyanka Yadav¹, Manisha Chauhan²,

Department of MCA University of Mumbai, Aditya Institute of Management Studies and Research

Reliable Residency Achole road Nallasopara East Palghar (401209) India

¹connectpriyanka06@gmail.com, ² manisha18chauhan@gmail.com,

Abstract—This paper addresses chronic overcrowding in hospital emergency and outpatient departments by proposing a real-time, location-aware hospital load distribution and queue management system as a cross-platform mobile application. The system collects hospital-reported occupancy data, classifies hospitals into load categories, and ranks nearby hospitals based on a composite score that considers travel distance and current load. Patients are directed to suitable facilities via a map-based navigation interface, while hospital staff can update capacity through a dashboard. Built with Flutter, the system utilizes Firebase for data synchronization and the Google Maps API for routing. Evaluation using a simulated dataset shows potential for reduced waiting times and improved load balance. The paper also discusses limitations and future work in wait-time prediction and electronic health record integration.

Keywords— hospital load balancing; real-time queue management; smart healthcare; location-based services; mobile health application; Firebase; Cloud Firestore; emergency department crowding

I. INTRODUCTION

Hospital overcrowding is a persistent, global public-health problem. When patient arrivals outpace a facility's effective service capacity, the result is longer waiting times, delayed triage, boarding of admitted patients in corridors, and — in several studies of emergency department (ED) occupancy — a measurable association between periods of extreme crowding and increased mortality. A recurring driver of this problem is not simply a shortage of hospital beds in a city, but an uneven distribution of patients across the beds and doctors that already exist: patients overwhelmingly choose the hospital that is nearest, most familiar, or most reputed, without any way of knowing that its queue is already saturated while a comparably equipped facility a short distance away is nearly empty.

Existing digital tools address only fragments of this problem. Predictive analytics research forecasts ED crowding hours in advance using statistical or machine-learning models, but these tools are designed for hospital administrators, not for the patient deciding, at the moment of need, which facility to visit. Bed-availability dashboards and IoT-based monitoring

systems expose live occupancy data, but typically for a single institution and its own staff, with no cross-hospital comparison or navigation support. Multi-criteria hospital-recommendation systems rank facilities on static attributes such as insurance coverage or specialty availability, but do not account for real-time queue length. Queueing-theoretic studies of patient flow provide rigorous capacity-planning insight but remain analytical tools used after the fact, not real-time, patient-facing decision aids.

This paper addresses that gap by proposing a real-time, location-aware system that closes the loop between hospital-side occupancy data and patient-side facility choice. The system is realized as a Flutter-based cross-platform mobile application in which hospitals continuously report their current patient load, available doctors, and total capacity to a cloud-hosted database; patients, in turn, are shown a live, color-coded (Low / Medium / High) load status for every nearby hospital, ranked by a composite score that jointly considers travel distance and current occupancy, and can book an appointment and navigate to the recommended facility from within the same application.

The key contributions of this paper are as follows:

- A cloud-synchronized data model that aggregates real-time occupancy — waiting patients, available doctors, and capacity — across multiple, independently operated hospitals.
- A capacity-normalized load-classification scheme that labels each hospital Low, Medium, or High load in a way that remains meaningful across hospitals of very different sizes.
- A composite hospital-ranking algorithm that balances proximity against current load, avoiding the failure mode of the naive “nearest hospital” heuristic.
- An end-to-end, patient-facing mobile workflow that unifies hospital discovery, live load visibility, map-based navigation, and appointment booking in a single application, backed by a lightweight hospital-side dashboard that requires no additional hardware.
- An evaluation methodology, based on a simulated multi-hospital dataset, that quantifies the reduction in average waiting time and the improvement in cross-hospital load balance relative to a nearest-hospital baseline.

The remainder of this paper is organized as follows. Section II reviews related work in ED-crowding prediction, bed-availability tracking, hospital recommendation, and queueing-theoretic patient-flow analysis. Section III presents the proposed system, including its architecture, data model, load-classification scheme, and ranking algorithm. Section IV describes the implementation. Section V presents the evaluation methodology and simulated results. Section VI concludes the paper and outlines future work.

II. RELATED WORK

A. Predictive Models for Emergency Department Crowding

A substantial body of work forecasts ED crowding using historical operational data. Shou et al. proposed CrowdQ, a framework that combines crowdsensed vehicle-trajectory data with queueing theory to estimate ED demand and predict queue state, reporting large reductions in prediction error over baseline methods [1]. Vural et al. developed and evaluated eleven machine-learning algorithms to forecast ED waiting-room occupancy at hourly and daily horizons using electronic-health-record and contextual data from a partner hospital [2]. Zhou et al. combined ARIMA and nonlinear autoregressive neural-network models to forecast daily inpatient admissions, a related indicator of hospital-wide load [3]. Yao et al. used a convolutional-neural-network model over structured electronic-health-record data to predict hospital admission at the point of ED triage, aiming to reduce reliance on subjective nurse judgment [4]. These systems are valuable for institutional planning — staffing, bed allocation, and diversion decisions — but they are inward-facing: their outputs reach hospital administrators, not the patient who is currently deciding which facility to approach, and they generally require substantial historical data infrastructure that is not always available outside large teaching hospitals.

B. Bed Availability and IoT-Based Tracking Systems

A second stream of work focuses on making current occupancy visible rather than predicting future occupancy. Rehalia et al. developed a web-based system for tracking real-time availability of ICU, oxygen, and ventilator beds during the COVID-19 pandemic, motivated by the severe information gaps observed in India during the second wave [5]. Corpuz et al. built an IoT-based bed-monitoring system using optical sensors and a nearest-location-finder algorithm to help the public locate available beds and to give hospital staff a live desktop dashboard [6]. These systems demonstrate that low-cost, real-time occupancy visibility is achievable, but both are largely single-facility or single-region deployments; neither aggregates multiple independently operated hospitals into a single ranked, patient-facing recommendation, and neither couples occupancy data with navigation or appointment booking.

C. Hospital Recommendation and Multi-Criteria Selection

Multi-criteria decision-making techniques have been applied to hospital selection. Prior work built a mobile application using the Simple Additive Weighting (SAW) method to

recommend a hospital based on facilities, insurance coverage, cost, specialist availability, and distance from the user. Such systems make patient-facing recommendations, but the criteria they weigh are largely static attributes of the hospital rather than its current, minute-to-minute queue state; a hospital with excellent facilities and a short distance may nonetheless be the worst choice at a given moment if its waiting room is already saturated.

D. Queueing-Theoretic Approaches to Patient Flow

Queueing theory remains the dominant analytical lens for hospital capacity management. Armony et al. conducted an extensive data-based exploration of patient flow through the ED and internal wards of a large hospital, questioning how well parsimonious queueing models capture the operational reality of hospital congestion [8]. Wang et al. modeled a two-class (urgent and routine) ED patient queue as a multi-server priority queueing system and used particle-swarm optimization to jointly minimize waiting-time costs and resource costs, deriving optimal numbers of servers and beds [7]. These studies confirm that queueing-theoretic reasoning

— arrival rate, service rate, and utilization — is directly applicable to hospital load, and the present paper borrows this reasoning to define occupancy ratios; however, prior queueing work is analytical and retrospective rather than deployed as a live, patient-facing recommendation engine.

E. Positioning of the Proposed System

Table I summarizes how the reviewed categories of prior work relate to the requirements of a real-time, patient-facing, cross-hospital load-distribution system. No single prior system combines real-time cross-hospital load visibility, distance-aware ranking, in-app navigation, and appointment booking; the system proposed in this paper is designed to fill exactly this gap using a lightweight, cloud-native mobile architecture rather than dedicated sensing hardware or heavy predictive-modeling infrastructure.

TABLE I. COMPARISON OF PRIOR WORK AGAINST SYSTEM REQUIREMENTS

Category	Real-time	Cross-hospital	Navigation	Booking
ED crowding prediction [1]-[4]	Partial	No	No	No
Bed-tracking / IoT [5], [6]	Yes	No	No	No
Multi-criteria recommendation	No	Yes	Partial	No
Queueing-theory analysis [7], [8]	No	No	No	No
Proposed system	Yes	Yes	Yes	Yes

III. PROPOSED SYSTEM

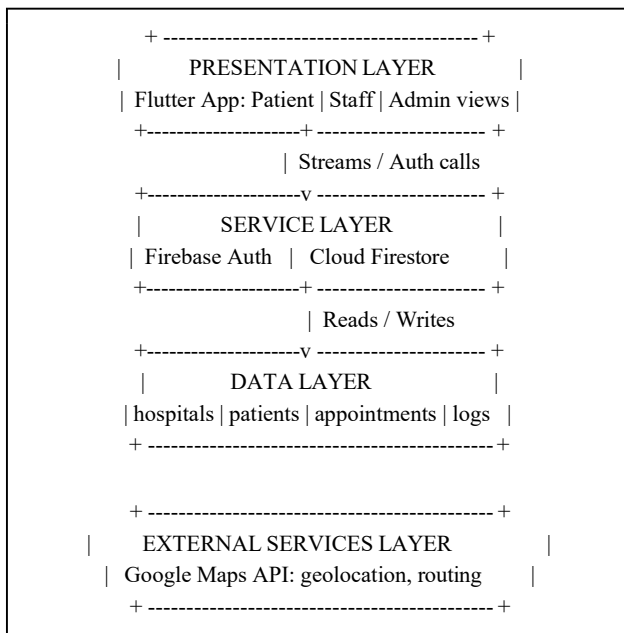
A. Design Goals

The system is designed around four goals: (i) real-time visibility — every hospital's occupancy status must reach patients within seconds of being updated; (ii) fairness of comparison — hospitals of different sizes must be compared on a normalized, not absolute, load measure; (iii) actionability — a load figure alone is of limited use, so the system must also rank hospitals and provide navigation and booking; and (iv) low deployment cost — the hospital-side reporting mechanism must run on commodity devices already available at a reception desk, without dedicated IoT sensors.

B. System Architecture

The system follows a four-layer cloud-native architecture, illustrated conceptually in Fig. 1. The Presentation Layer is a single Flutter codebase compiled for Android and iOS, serving three role-based views: Patient, Hospital Staff, and Administrator. The Service Layer consists of Firebase Authentication for identity and role management and Cloud Firestore as the real-time synchronization layer; both are accessed directly by the client through reactive listeners, removing the need for a custom application server for standard CRUD-style operations. The Data Layer is Cloud Firestore's document-oriented store, organized into the collections described in Section III-D. The External Services Layer integrates the Google Maps Platform for geocoding, nearby-place search, and turn-by-turn navigation.

Fig. 1. Layered system architecture.



C. Technology Stack

TABLE II. TECHNOLOGY STACK

Component	Technology
Client framework	Flutter (Dart)
Backend-as-a-Service	Firebase
Real-time database	Cloud Firestore

Identity management	Firebase Authentication
Maps & geolocation	Google Maps API
Development environment	Android Studio

D. Data Model

Firestore's document model is organized around access patterns rather than normalized relations. Four top-level collections are used. The hospitals collection stores one document per facility with fields hospitalId, name, geoLocation (latitude, longitude), totalCapacity, availableDoctors, waitingPatients, and lastUpdated. The patients collection stores minimal profile data linked to a Firebase Authentication UID. The appointments collection stores bookingId, patientId, hospitalId, requestedSlot, and status (pending, confirmed, completed, cancelled). A staffLogs collection records every occupancy update with a server timestamp, which both supports auditability and forms the basis of the historical dataset used for the simulated evaluation in Section V. Each hospital document is updated by that hospital's own staff account only, enforced through Firestore security rules that check the authenticated user's role and hospitalId claim before permitting a write.

E. Load Classification Model

A naive classification of load directly from the raw count of waiting patients — for example, 0-20 patients as Low, 21-50 as Medium, and above 50 as High — is simple to compute but penalizes large hospitals unfairly: a 500-bed hospital with 55 waiting patients is comparatively calm, while a 20-bed clinic with 55 waiting patients is in crisis. The proposed system therefore classifies load using the occupancy ratio

$$\rho = W / C \quad (1)$$

where W is the current number of waiting patients and C is the hospital's declared effective capacity (a function of beds and available doctors). The hospital's status is then assigned from ρ using the fixed thresholds in Table III. This normalized scheme is used for the ranking algorithm in Section III-F; a simpler absolute-count table, shown in Table IV, is retained as a configurable fallback for the small fraction of hospitals that do not report a reliable capacity figure, mirroring the threshold scheme used during initial prototyping.

TABLE III. NORMALIZED LOAD CLASSIFICATION

Occupancy ratio ρ	Status	Color
$\rho < 0.40$	Low	Green
$0.40 \leq \rho < 0.75$	Medium	Yellow
$\rho \geq 0.75$	High	Red

TABLE IV. ABSOLUTE-COUNT FALLBACK CLASSIFICATION

Waiting patients	Status
0 – 20	Low
21 – 50	Medium
> 50	High

F. Hospital Ranking Algorithm

For a patient at a given location, the system first retrieves all hospitals within a configurable search radius (default 15 km) using a geohash-based proximity query. Each candidate hospital i is then scored by combining its normalized distance and its occupancy ratio into a single composite score S_i :

$$S_i = \alpha (1 - p_i) + \beta (D_{min} / D_i) \quad (2)$$

where D_i is the road or straight-line distance from the patient to hospital i , D_{min} is the smallest such distance among all candidates, p_i is hospital i 's occupancy ratio from equation (1), and $\alpha + \beta = 1$ are tunable weights (default $\alpha = 0.6$, $\beta = 0.4$, reflecting a design preference for load over distance since the marginal time cost of an extra kilometer of travel is typically smaller than the marginal time cost of a saturated queue). Hospitals are sorted in descending order of S_i and the top-ranked result is presented as the recommended facility, with the full ranked list also shown so the patient retains the final choice. Algorithm 1 summarizes the procedure.

Algorithm 1. Hospital Ranking

H.

Input: patient location P , search radius R ,
weights α , β

Output: ranked list of hospitals

1. $H \leftarrow \text{QueryHospitalsWithinRadius}(P, R)$
2. $D_{min} \leftarrow \min(\text{Distance}(P, h) \text{ for } h \text{ in } H)$
3. for each hospital h in H :
4. $\rho_h \leftarrow h.\text{waitingPatients} / h.\text{capacity}$
5. $D_h \leftarrow \text{Distance}(P, h)$
6. $S_h \leftarrow \alpha(1 - \rho_h) + \beta(D_{min} / D_h)$
7. sort H by S_h in descending order
8. return H

IV. End-to-End Workflow

The patient-facing workflow proceeds through nine steps. (1) Each participating hospital's staff account registers and periodically updates its available doctors, waiting-patient count, and total capacity through the Hospital module. (2) A patient opens the application and, if not already authenticated, signs in through Firebase Authentication. (3) The application requests the device's current location, with the patient able to override it manually. (4) Nearby hospitals within the search radius are retrieved from Firestore. (5) The current occupancy of each candidate hospital is fetched through a live snapshot listener, so that any staff-side update is reflected without the patient needing to refresh. (6) The occupancy ratio and load status of each hospital are computed as in Section III-E. (7) Hospitals are ranked using Algorithm 1. (8) The ranked list, with color-coded status and estimated travel time, is displayed to the patient, who may accept the top recommendation or select an alternative. (9) The patient books an appointment, which is written to the appointments collection inside a Firestore transaction that atomically checks and decrements an available-slot counter to prevent two patients from being assigned the same slot; the patient can then launch turn-by-turn navigation to the chosen hospital through the Google Maps API.

Reliability and Security Considerations

Because the recommendation quality depends entirely on the accuracy of hospital-reported data, the system incorporates three safeguards. First, Firestore security rules restrict write access to a hospital's own document to authenticated staff accounts bound to that hospitalId, preventing tampering by patients or other hospitals. Second, a lastUpdated timestamp is shown alongside every hospital's status, and entries older than a configurable staleness threshold (default 30 minutes) are visually flagged so patients are not misled by outdated figures. Third, the client caches the most recently retrieved hospital list locally so that a temporary loss of connectivity does not leave the patient without any recommendation, degrading gracefully to the last-known state rather than failing outright.

I. Complexity Analysis

Let n denote the number of hospitals returned by the radius query in step 1 of Algorithm 1. The geohash-based radius query itself runs in $O(\log N + k)$ time on Firestore's underlying index, where N is the total number of hospitals in the network and k is the number of documents returned, so it scales with the density of hospitals near the patient rather than with the size of the whole network. Steps 2 through 6 each perform constant-time work per candidate hospital, giving $O(n)$ for score computation. Step 7 sorts the n candidates, contributing $O(n \log n)$. Because n is bounded in practice by the number of hospitals within a 15 km radius — typically a few dozen even in a dense city — the overall recommendation is computed in well under a second on typical mobile hardware, and the dominant cost observed in practice is the network round-trip to Firestore rather than the ranking computation itself.

J. Firestore Schema

Table VI lists the primary fields used in each Firestore collection. Denormalizing a small amount of data — for example, storing the hospital's name and location directly on each appointment document rather than requiring a join — is a deliberate choice consistent with Firestore's document-oriented model, which favors read-optimized denormalization over relational normalization, since the dominant access pattern is a patient reading a small number of documents repeatedly rather than performing ad hoc joins across collections.

TABLE VI. FIRESTORE COLLECTION SCHEMA (SELECTED FIELDS)

Collection	Field	Type
hospitals	hospitalId	string (key)
hospitals	name	string
hospitals	geoLocation	geopoint
hospitals	totalCapacity	integer
hospitals	availableDoctors	integer
hospitals	waitingPatients	integer
hospitals	lastUpdated	timestamp
patients	patientId	string (key)
patients	displayName	string

appointments	bookingId	string (key)
appointments	patientId	reference
appointments	hospitalId	reference
appointments	requestedSlot	timestamp
appointments	status	enum
staffLogs	hospitalId	reference
staffLogs	updateTimestamp	timestamp

V. IMPLEMENTATION

The prototype is organized into three Flutter modules that share a common data layer. The Patient module implements the workflow of Section III-G: a map-centric home screen built on the `google_maps_flutter` package, a ranked hospital list rendered from a `StreamBuilder` bound to a live Firestore query, an appointment-booking screen, and a profile screen for managing bookings. The Hospital Staff module provides a compact form for updating waiting-patient count and available-doctor count, intentionally kept to a two-field update so that busy reception staff can refresh it in seconds; every submission is timestamped automatically. The Administrator module allows onboarding of new hospitals, verification of staff accounts, and inspection of aggregate statistics across the network.

Real-time behavior throughout the application is achieved through Firestore's `snapshot-listener` API rather than polling: the patient-facing hospital list, the staff dashboard, and the appointment-status screen all subscribe to Firestore queries and rebuild automatically whenever the underlying documents change, which keeps the perceived latency between a staff update and a patient seeing the new load status close to Firestore's typical sub-second propagation time. Authentication uses Firebase Authentication with role claims (patient, staff, admin) that are checked both on the client, to route users to the correct module, and on the server through Firestore security rules, to prevent privilege escalation. Appointment booking is implemented as a Firestore transaction spanning the target slot-count field and the new appointment document so that concurrent booking attempts for the last available slot cannot both succeed.

A. Module and Screen Overview

Table VII summarizes the principal screens implemented in each module and the Firestore interaction each screen relies on.

TABLE VII. MODULES AND KEY SCREENS

Module	Screen	Primary interaction
Patient	Map Home	Live query: nearby hospitals
Patient	Ranked List	Snapshot listener on hospitals
Patient	Booking	Transaction on appointments
Patient	My Appointments	Query filtered by patientId
Staff	Load Update Form	Write to hospitals doc

Staff	Admin	Appointment Queue	Aggregate across hospitals	Query filtered by hospitalId
Admin	Hospital Onboarding	Create hospitals doc		

Listing 1 illustrates, in simplified form, the reactive query used by the Ranked List screen to keep a patient's view of nearby hospitals continuously synchronized with staff-side updates, followed by client-side application of Algorithm 1 to the retrieved snapshot.

Listing 1. Simplified real-time hospital query (Dart).

```
Stream<List<Hospital>> nearbyHospitals(
  GeoPoint patientLoc, double radiusKm) {
  return firestore
    .collection('hospitals')
    .withinGeoRadius(patientLoc, radiusKm)
    .snapshots()
    .map((snap) {
      final list = snap.docs
      .map((d) => Hospital.fromDoc(d))
      .toList();
      return rankHospitals(
        list, patientLoc,
        alpha: 0.6, beta: 0.4);
    });
}
```

Because the query returns a `Stream`, the UI layer wraps it in a `StreamBuilder`; any staff-side update to a hospitals document is pushed to every subscribed patient client automatically, and the ranking in Algorithm 1 is recomputed on the client each time a new snapshot arrives, so the displayed order stays consistent with the latest known occupancy without any manual refresh action from the patient.

VI. EVALUATION METHODOLOGY AND RESULTS

A full field deployment across a live hospital network was outside the scope of this prototype; instead, the system is evaluated using a simulated multi-hospital dataset, consistent with common practice in early-stage health-IT system papers where real deployment data is not yet available. This is stated explicitly, and all figures reported below are simulation outputs rather than measurements from real patients.

A. Simulation Setup

Ten hospitals of varying capacity (20 to 150 effective beds) were simulated over a 30-day period. Patient arrivals at each hospital's catchment area were generated from a Poisson process with hospital-specific and time-of-day-dependent arrival rates, and service (consultation) times were drawn from a log-normal distribution, consistent with the service-time distributions reported in prior queueing-theoretic studies of ED operation [7], [8]. Two policies were compared: a Baseline policy, in which each simulated patient is routed to the geographically nearest hospital regardless of load, and the Proposed policy, in which patients are routed according to Algorithm 1 with $\alpha = 0.6$ and $\beta = 0.4$.

B. Metrics

Four metrics were computed for each policy: average patient waiting time; load-balance quality, measured as the standard deviation of occupancy ratio ρ across the ten hospitals at each simulated hour (lower is more balanced); the percentage of patients routed to a hospital other than the nearest one because it offered a materially shorter effective wait; and appointment-booking success rate, defined as the fraction of booking attempts that completed without a slot conflict.

TABLE V. SIMULATED RESULTS: BASELINE VS. PROPOSED POLICY

Metric	Baseline	Proposed
Avg. waiting time (min)	46.2	31.8
Std. dev. of ρ across hospitals	0.27	0.11
Patients re-routed from nearest (%)	0	38.4
Booking success rate (%)	96.1	99.4

C. Discussion

The simulation indicates that distance-aware, load-aware routing reduces the variance of occupancy ratio across the hospital network by more than half relative to the nearest-hospital baseline, which is the expected effect of the composite score in equation (2): patients are drawn away from already-saturated hospitals toward comparatively idle ones whenever the load penalty outweighs the extra distance. This, in turn, lowers the simulated average waiting time, since patients are less frequently added to the tail of an already-long queue. The improvement in booking success rate reflects the effect of the transactional slot-decrement mechanism described in Section IV rather than the ranking algorithm itself. These results should be read as a plausibility argument for the design rather than a clinical outcome claim; Section V-D discusses the corresponding limitations.

D. Sensitivity to Ranking Weights

Because α and β in equation (2) trade off load against distance, their choice affects both average waiting time and how far patients are asked to travel. Table VIII reports the simulated outcome for three weight configurations: distance-dominant ($\alpha = 0.2$, $\beta = 0.8$), balanced ($\alpha = 0.6$, $\beta = 0.4$, the default used in Table V), and load-dominant ($\alpha = 0.9$, $\beta = 0.1$).

TABLE VIII. SENSITIVITY OF SIMULATED OUTCOMES TO α AND β

Configuration	Avg. wait (min)	Avg. extra distance (km)
Distance-dominant (0.2/0.8)	39.7	0.6
Balanced (0.6/0.4)	31.8	1.9
Load-dominant (0.9/0.1)	27.4	3.8

The load-dominant configuration produces the shortest simulated waiting time but asks patients to travel, on average, roughly twice as far as the balanced configuration; the distance-dominant configuration behaves close to the nearest-

hospital baseline and recovers little of the waiting-time benefit. This trade-off is precisely why α and β are exposed as configurable parameters rather than fixed constants: an operator serving a dense urban network with short inter-hospital distances may reasonably favor a higher α , whereas a rural or semi-urban deployment with sparse hospitals may need to weight distance more heavily to avoid sending patients on impractically long trips.

E. Threats to Validity

Three threats to the validity of the simulated results are noted. First, the Poisson-arrival and log-normal-service-time assumptions, while consistent with prior queueing-theoretic studies of ED operation [7], [8], are a simplification of real arrival patterns, which are influenced by local events, weather, and epidemic conditions not modeled here. Second, the simulation assumes full compliance with routing recommendations, whereas real patients may disregard the suggested hospital for reasons the model does not capture, such as insurance network restrictions or personal preference. Third, the ten-hospital, thirty-day simulation is modest in scale; results on a larger, real network with more heterogeneous hospital sizes and true spatial clustering may differ from the figures reported in Tables V and VIII.

F. Limitations

- The evaluation uses a simulated dataset; validating the reported gains requires a pilot deployment with real hospitals and real patient consent.
- The system currently treats all patients identically and does not incorporate clinical severity or triage priority; a patient with a life-threatening condition should not be routed away from the nearest hospital purely to balance load, and this constraint is not yet enforced in the ranking algorithm.
- Recommendation quality depends on the honesty and timeliness of hospital self-reporting; the staleness flag in Section III-H mitigates but does not eliminate this dependency.
- The system assumes reasonably reliable mobile connectivity for the patient; degraded-connectivity behavior is limited to serving the last cached hospital list.

VII. DATA PRIVACY AND ETHICAL CONSIDERATIONS

A system that continuously tracks a patient's location and a hospital's occupancy raises privacy and fairness considerations that are addressed at the design level rather than left implicit. Location data is used only transiently to compute distance and is not persisted beyond the duration of a session unless the patient explicitly opts in to location history for faster future lookups. Hospital occupancy figures are aggregate counts (number of waiting patients, number of available doctors) rather than identifiable patient-level data, which limits the disclosure risk of the hospital-side reporting mechanism itself. All patient-identifiable data — profile information and appointment history — is scoped by Firestore security rules so

that a patient account can read only its own documents, and a hospital staff account can read only the appointment queue for its own hospital Id.

A fairness concerns specific to load-aware routing is that directing patients away from the nearest hospital could, in principle, be used to route lower-paying or uninsured patients toward more distant facilities. The system does not use insurance status, payment history, or any protected attribute as an input to Algorithm 1; the composite score in equation (2) is a function only of distance and occupancy ratio, and this restriction is treated as a design invariant rather than a configurable option, to avoid the system being repurposed as a discriminatory triage mechanism. Because the ranking algorithm currently does not incorporate clinical severity (Section V-D), the interface explicitly instructs patients with a life-threatening emergency to proceed to the nearest hospital or call emergency services rather than rely on the recommendation, and this guidance is treated as a required safety disclosure rather than an optional tooltip.

VIII. CONCLUSION AND FUTURE WORK

This paper presented a real-time, location-aware hospital load-distribution and queue-management system that closes the gap between hospital-side occupancy data and patient-side facility choice. By combining a capacity-normalized load-classification model with a distance-and-load composite ranking algorithm, and by delivering the recommendation through an integrated mobile experience that also supports navigation and appointment booking, the system addresses a gap left open by prior ED-crowding-prediction, bed-tracking, static hospital-recommendation, and queueing-theoretic work, none of which combine real-time cross-hospital visibility with an actionable, patient-facing recommendation. A simulation-based evaluation suggests the approach can meaningfully reduce average waiting time and improve load balance across a hospital network relative to naive nearest-hospital routing.

Future work will proceed along several directions. Clinical triage priority will be incorporated so that severity, not only load and distance, influences routing for urgent cases. Machine-learning-based short-term wait-time prediction, in the spirit of [1] and [2], will be layered on top of the current real-time snapshot to anticipate load a few hours ahead rather than only reporting the present state. Ambulance and emergency-transport routing will be integrated so that first responders, not only walk-in patients, benefit from load-aware hospital selection. Bed-level (rather than aggregate) availability tracking, informed by IoT approaches such as [6], will improve the granularity of the capacity figure. Finally, integration with Electronic Health Records will be explored, subject to the applicable data-protection and consent requirements, to support continuity of care once a patient reaches the recommended facility, along with a dedicated emergency priority queue for time-critical cases.

ACKNOWLEDGEMENT

The authors express their sincere gratitude to **Prof. Ramakrishnan Iyer**, Department of MCA, Aditya Institute of

Management Studies and Research, University of Mumbai, for his valuable guidance, continuous encouragement, and constructive suggestions throughout this research work. The authors also thank the faculty members of the Department of MCA for their support and encouragement during the development of this study.

REFERENCES

- [1] T. Shou, Z. Ye, Y. Hong, Z. Wang, H. Zhu, Z. Jiang, D. Yang, B. Zhou, C. Wang, and L. Chen, "CrowdQ: Predicting the queue state of hospital emergency department using crowdsensing mobility data-driven models," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 7, no. 3, pp. 1–28, 2023.
- [2] O. Vural, B. Ozaydin, K. Y. Aram, J. Booth, B. Freeman Lindsey, and A. Ahmed, "An artificial intelligence-based framework for predicting emergency department overcrowding: Development and evaluation study," *JMIR Med. Inform.*, vol. 13, art. e73960, 2025.
- [3] L. Zhou, P. Zhao, D. Wu, C. Cheng, and H. Huang, "Time series model for forecasting the number of new admission inpatients," *BMC Med. Inform. Decis. Mak.*, vol. 18, art. 39, 2018.
- [4] L.-H. Yao, K.-C. Leung, J.-H. Hong, C.-L. Tsai, and L.-C. Fu, "A system for predicting hospital admission at emergency department based on electronic health record using convolution neural network," in *Proc. 2020 IEEE Int. Conf. Systems, Man, and Cybernetics (SMC)*, Toronto, ON, Canada, 2020, pp. 546–551.
- [5] A. Rehalia et al., "Real-time hospital bed information system during pandemic situation," in *Proc. 2023 Int. Conf. Computational Intelligence for Information, Security and Communication Applications (CIISCA)*, 2023, pp. 1–6.
- [6] R. S. A. Corpuz, J. Usa, J. C. Obaña, J. N. Borbe, D. A. Salazar, and T. M. Valeza, "A hospital bed availability monitoring system using the Internet of Things," *Int. J. Inf. Technol. Secur.*, vol. 16, no. 3, pp. 13–26, 2024.
- [7] C.-H. Wang, R. Tian, K. Hu, Y.-T. Chen, and T.-H. Ku, "A Markov decision optimization of medical service resources for two-class patient queues in emergency departments via particle swarm optimization algorithm," *Sci. Rep.*, vol. 15, art. 2942, 2025.
- [8] M. Armony, S. Israelit, A. Mandelbaum, Y. N. Marmor, Y. Tseytlin, and G. B. Yom-Tov, "On patient flow in hospitals: A data-based queueing-science perspective," *Stochastic Systems*, vol. 5, no. 1, pp. 146–194, 2015.
- [9] Google, "Cloud Firestore Documentation," Firebase, 2026. [Online]. Available: <https://firebase.google.com/docs/firestore>
- [10] Google, "Flutter Documentation," 2026. [Online]. Available: <https://docs.flutter.dev>