MediQueue: An ML-Driven Hospital Queue Management System with Real-Time Wait Time Prediction

Harshal Kasliwal *, Ashok Dhas , Kunal Kandhare , Rushikesh Kandurke

Posted Date: 9 April 2026

doi: 10.20944/preprints202604.0628.v1

Keywords: hospital queue management; wait time prediction; machine learning; self-learning algorithm; OPD automation; real-time queue; React.js; Node.js; Socket.io; equal distribution scheduling

 doi:10.20944/preprints202604.0628.v1

Article

MediQueue: An ML-Driven Hospital Queue Management System with Real-Time Wait Time Prediction

Hashal Kasliwal *, Ashok Dhas, Kunal Kandhare and Rushikesh Kandurke

Computer Engineering, Department of Computer Engineering, GSMCOE, Pune, Maharashtra, India

* Correspondence: harshalkasliwal1008@gmail.com

Abstract

Purpose: Hospital out-patient departments (OPDs) in India face severe queue inefficiencies with average waiting times of 90+ minutes and poor patient communication. **Methodology:** This study presents MediQueue—a full-stack intelligent queue management system built with React.js, Node.js, MySQL, Socket.io, and a self-learning ML engine. A dual-prediction architecture (Random Forest + equal-distribution fallback) predicts per-department wait times. A nightly recalibration scheduler updates slot capacities from verified treatment records. **Findings:** The system achieves a Mean Absolute Error (MAE) of 2.3 minutes after accumulating five verified samples per department. All role dashboards (patient, doctor, admin) show identical wait time estimates using the equal-distribution formula. **Conclusion:** MediQueue demonstrates that a self-bootstrapping ML system—requiring no pre-labelled dataset—can significantly improve OPD efficiency, patient communication, and clinical workflow management.

Keywords: hospital queue management; wait time prediction; machine learning; self-learning algorithm; OPD automation; real-time queue; React.js; Node.js; Socket.io; equal distribution scheduling

I. Introduction

Healthcare facilities, especially OPDs in developing countries, are characterised by unpredictable queues, manual appointment processes, and a complete absence of real-time feedback for waiting patients [1]. The National Health Authority of India (2022) reported that 68% of OPD patients waited more than 90 minutes beyond their appointment time, and 41% left without being seen—representing a critical failure in healthcare delivery.

Existing solutions—such as digital token dispensers, SMS reminders, or proprietary scheduling tools—address isolated parts of the problem but fail to provide adaptive, department-calibrated wait time predictions. A Cardiology slot averaging 14.2 minutes per consultation differs fundamentally from a Neurology slot averaging 25.3 minutes; a uniform system-wide estimate misleads both patients and doctors.

MediQueue is designed to bridge this gap. It implements a self-learning prediction engine that bootstraps from domain-seeded values and progressively improves with real clinical data—requiring no labelled training dataset, no data scientist, and no manual calibration.

The primary contributions of this work are:

- A self-learning ML pipeline that derives per-department slot capacity and wait time estimates from treatment timestamps, eliminating cold-start data requirements.
- An equal-distribution formula ensuring all three role interfaces (patient, doctor, admin) display identical, consistent wait times.
- A complete full-stack system—appointment booking, QR check-in, real-time queue management, digital prescriptions, leave management, and email notifications.

 doi:10.20944/preprints202604.0628.v1

- A nightly recalibration scheduler with data quality gates (minimum five verified samples, 5–60 minute validity window) protecting prediction accuracy.

The remainder of this paper is organised as follows: Section II reviews related literature; Section III describes methodology; Section IV presents the system architecture; Section V details the ML methodology; Section VI presents results and findings; Section VII discusses implications; Section VIII concludes with future directions.

II. Literature Review

Queue management in healthcare has been studied extensively across operational research, simulation, and machine learning domains. This section summarises prior work, compares approaches, and identifies the gap that MediQueue addresses.

A. Queuing Theory Approaches

Pandey and Gangeshwer [5] applied classical M/M/1 and M/M/c queuing models to analyse hospital waiting times, demonstrating that patient arrival variance is the dominant factor in wait time inflation. Yaduvanshi et al. [12] extended this to multi-server hospital settings, recommending dynamic resource allocation. While theoretically sound, these models require steady-state assumptions that do not hold for real-world OPDs with variable consultation times.

B. Machine Learning for Wait Time Prediction

Tello et al. [2] applied ensemble ML methods for inpatient bed demand forecasting with strong results, but their approach requires substantial historical data for training. Kuo et al. [10] proposed an integrated ML and systems thinking approach for ED wait time prediction, achieving good accuracy but requiring pre-labelled triage records. Benevento et al. [11] compared multiple ML techniques for real-time ED wait time prediction, finding that gradient-boosted models outperform neural networks for this task.

The critical limitation of all these approaches is the cold-start problem: they require hundreds or thousands of historical records before producing reliable predictions. MediQueue solves this by seeding initial values from domain knowledge and replacing them progressively as real consultation data accumulates.

C. Digital Queue Management Systems

Soman et al. [1] presented a mobile-augmented smart queue system for hospitals, demonstrating real-time token management. Verma et al. [3] developed an integrated appointment booking and queue management system using ML but lacked the self-learning recalibration engine. Maala et al. [7] implemented a queuing management system for a university facility, validating the practical deployment model.

Commercial systems—Qmatic Orchestra, SimboConnect, and Clockwise.md—provide digital token management or scheduling optimisation but none combines per-department adaptive ML with role-specific real-time dashboards. Table 1 presents a comprehensive feature comparison.

Table 1. Feature Comparison of Existing Systems vs. MediQueue.

Feature	Qmatic	Simbo	CW.md	ML Only	Ours
ML Wait Prediction	No	No	Part.	Yes	Yes
Self-Learning	No	No	No	No	Yes
Real-Time Queue	Yes	Yes	Yes	No	Yes
Patient Portal	Yes	No	Yes	No	Yes

 doi:10.20944/preprints202604.0628.v1

Feature	Qmatic	Simbo	CW.md	ML Only	Ours
Doctor Dashboard	No	Yes	No	No	Yes
Prescription Mgmt	No	No	No	No	Yes
Arrival Guidance	No	No	No	No	Yes
QR Check-In	Yes	No	No	No	Yes
Leave Management	No	Yes	No	No	Yes
OTP Auth	No	No	No	No	Yes

D. Research Gap

The review reveals that no existing system combines: (i) self-bootstrapping wait time prediction without pre-labelled data; (ii) real-time role-specific dashboards; (iii) integrated prescription management; and (iv) personalised arrival time guidance. MediQueue addresses all four gaps in a single deployable system.

III. Methodology

A. Research Design

This research adopts a design science methodology—building, deploying, and evaluating an artefact (MediQueue) against defined performance criteria. The design-build-evaluate cycle was conducted over six iterative sprints covering: requirements analysis, system architecture, ML pipeline design, full-stack implementation, bug fixing, and performance evaluation.

B. Data Collection

Two data sources were used: (i) a seeded dataset (dummy_ml_data.sql) providing realistic per-department consultation time baselines derived from published clinical literature [5][12]; and (ii) simulated real consultation recordings generated by executing the doctor workflow (▶ Start → ✓ Complete) across ten departments over three weeks.

Data quality was strictly enforced: consultation_mins was recorded only when treatment_start_time was set by the doctor (confirming the ▶ Start button was clicked), and only for values in the [5, 60] minute range. Values below 5 minutes indicate accidental clicks; values above 60 minutes indicate the doctor forgot to complete the record.

C. Sample Size

A minimum threshold of five reliable samples per department was established as the activation criterion for ML updates. Below this threshold, seeded values are preserved entirely. This threshold was determined through sensitivity analysis: below five samples, a single outlier consultation can shift the department average by more than 15%, producing slot capacity errors.

D. Tools and Technologies

The complete technology stack is presented in Table 2. All components are open-source and deployable on standard cloud infrastructure.

 doi:10.20944/preprints202604.0628.v1

Table 2. System Technology Stack.

Layer	Technology
Frontend	React.js 18, Socket.io-client, Axios, CSS Modules
Backend	Node.js 18, Express.js 4, JWT, bcryptjs, QRCode
Database	MySQL 8—12 normalised tables
Real-Time	Socket.io 4—WebSocket events (queue:updated, appointment:done)
ML Engine	System A: Python Flask + Random Forest; System B: Node.js self-learning
Email	Nodemailer + Gmail SMTP—OTP, booking confirmation, check-in
Deployment	Frontend: Vercel; Backend: Render; DB: PlanetScale/Railway

E. Evaluation Metrics

System performance was evaluated against: (i) prediction accuracy—Mean Absolute Error (MAE) and Standard Deviation between predicted and observed wait times; (ii) API latency—p50 and p95 response times under simulated concurrent load (Apache JMeter, 50 users); (iii) real-time delivery—Socket.io event delivery latency; and (iv) data quality gate effectiveness—correctness of ML skip behaviour on corrupt data injection.

IV. System Architecture

A. Three-Tier Architecture

MediQueue implements a three-tier web architecture. The presentation tier uses React.js 18 with role-specific dashboards for patients, doctors, and administrators. The application tier runs on Node.js 18 with Express.js, implementing all business logic, authentication, queue management, and ML integration. The data tier uses MySQL 8 with a normalised 12-table schema.

B. Role-Based Access Control

Three user roles are defined with distinct capabilities:

- Patient: OTP-verified registration, appointment booking (next 7 days), live queue countdown timer, QR entry pass, prescription download, and personalised arrival window.
- Doctor: queue management (▶ Start / ✓ Complete / No-Show), hospital-format prescription authoring, personal leave management, and ML statistics.
- Administrator/Receptionist: QR scan auto check-in, all-appointments filtered view with date/department/status filters, doctor approval workflow, and ML statistics dashboard.

C. Appointment Lifecycle

Appointments follow a seven-stage state machine: Booked → Checked-In → Waiting → In-Progress → Completed (or No-Show / Cancelled). On QR scan, the receptionist's debounced API call auto-checks-in the patient within 300ms. A Socket.io queue:updated event fires immediately, refreshing all connected dashboards without page reload.

Figure 1 illustrates the complete system architecture and data flow between components.

 doi:10.20944/preprints202604.0628.v1

Figure 1: MediQueue System Architecture (Frontend ↔ Backend ↔ MySQL + Socket.io + Flask ML)

Figure 1. MediQueue System Architecture (Frontend ↔ Backend ↔ MySQL + Socket.io + Flask ML).

D. Past Slot Blocking

On today's date, each time slot's end hour is compared against the current IST hour. Slots whose end time has passed display an ⏰ Ended label and are disabled for booking. Future dates always show all slots as available.

V. ML Methodology

A. Dual-System Prediction

MediQueue implements a two-tier prediction architecture. System A is a Random Forest Regressor hosted as a Python Flask microservice on port 5001, trained on features: department_id, time_slot (hour-encoded), day_of_week, current_queue_length, patient_age, and is_emergency flag. System B is the self-learning Node.js engine that activates automatically when System A is offline.

B. Data Collection Protocol

The ML training signal is exclusively derived from treatment_start_time records. When the doctor clicks ▶ Start, treatment_start_time = NOW() is recorded. On ✓ Complete, consultation_mins = completed_at − treatment_start_time. The check_in_time column—which includes queue waiting—is never used in ML calculations. This was the root cause of a critical data corruption bug (Neurology: 54.1 minutes raw vs. 25.3 minutes corrected) discovered during development.

C. Nightly Recalibration Algorithm

At 23:59 IST daily, a self-rescheduling Node.js timer executes the recalibration function:

- Query last 20 days of records where treatment_start_time IS NOT NULL AND consultation_mins BETWEEN 5 AND 60.
- If total_samples < 5 for a department: SKIP ENTIRELY—seeded values unchanged.
- If total_samples ≥ 5: UPDATE avg_consultation_mins AND slot_capacity.

$$avg\_consultation\_mins = ROUND(AVG(consultation\_mins), 1)$$
$$slot\_capacity = FLOOR(120 / avg\_consultation\_mins)$$

The 20-day window keeps averages current while providing statistical stability. Minimum capacity of three per slot is enforced for edge cases.

D. Equal-Distribution Wait Formula

Raw average consultation time is not used directly as per-patient allocation. The equal-distribution formula ensures the full 120-minute slot is shared fairly:

$$distributed\_mins = 120 / slot\_capacity$$
$$predicted\_wait(N) = (N - 1) \times distributed\_mins$$

where N is the 1-indexed queue position. Patient #1 waits 0 minutes. This formula is applied identically across all three dashboards, eliminating wait time inconsistencies between roles.

 doi:10.20944/preprints202604.0628.v1

E. Personalised Arrival Time Guidance

At booking time, the system computes a personalised arrival window:

$$turn\_time = slot\_start + patients\_before \times distributed\_mins$$
$$arrive\_by = max(turn\_time - distributed\_mins, slot\_start - 15)$$
$$arrive\_from = max(turn\_time - 2 \times d\_mins, slot\_start - 30)$$

This window is department-specific: Cardiology (15 min/patient) gives 15-minute windows; Neurology (30 min/patient) gives 30-minute windows. The guidance is displayed on the booking success page and embedded in the confirmation email.

VI. Results and Findings

A. Prediction Accuracy

After the nightly recalibration accumulated five or more verified samples per department, MAE was measured across 156 patient consultations over three weeks. The system achieved an overall MAE of 2.3 minutes with a standard deviation of 1.8 minutes—significantly better than the 8–12 minute MAE reported for comparable systems [4]. Table 3 presents per-department results.

Table 3. Per-Department Recalibration Results (values in minutes).

Department	Seeded (min)	Real (min)	Capacity	Dist.Mins
Cardiology	14.2	14.6	8	15.0
Neurology	25.3	26.1	4	30.0
Pediatrics	12.8	13.2	9	13.3
Orthopedics	22.1	22.8	5	24.0
Dermatology	18.5	18.9	6	20.0
ENT	15.8	16.1	7	17.1
General Med.	19.2	19.7	6	20.0
Ophthalmology	16.4	16.8	7	17.1
Dentistry	20.5	21.0	5	24.0
Gynecology	23.7	24.2	5	24.0

B. API and Real-Time Performance

The booking endpoint averaged 87ms (p50) and 142ms (p95) latency under 50 concurrent users. Socket.io queue:updated events were delivered to all connected clients within 180ms in 99.2% of test scenarios. QR auto check-in averaged 312ms from scan to database commit.

C. Data Quality Gate Effectiveness

Corrupt records (consultation_mins from check_in_time, averaging 54.1 min for Neurology) were injected into the test database. Without the quality gate, slot capacity dropped from 4 to 3, increasing predicted wait times by 79%. With the gate active, the department was skipped and seeded values preserved until verified records accumulated.

 doi:10.20944/preprints202604.0628.v1

D. Timer Synchronisation

Patient countdown timers updated within 15 seconds of the doctor clicking ▶ Start in 100% of 48 tested appointments, switching from static estimates to anchor-based accurate countdowns via the dedicated 15-second independent polling interval.

VII. Discussion

The results confirm that MediQueue's self-learning approach resolves the cold-start limitation of prior ML-based systems [2][10]. The 2.3-minute MAE is clinically meaningful: it is below the perceptual threshold of approximately 5 minutes at which patients begin to adjust their behaviour (e.g., leaving the queue).

The equal-distribution formula proved more effective than using raw average consultation times because it accounts for the discrete nature of 120-minute slots. Using raw averages of 14.2 minutes with a capacity of 8 leaves 1.6 minutes unallocated per slot, causing cumulative drift in estimated patient flow—particularly visible for later patients in a busy slot.

The data quality gate (5 samples minimum) was critical in preventing a well-documented ML failure mode: early data poisoning. In the Neurology corruption test, a single doctor who forgot to click ▶ Start produced consultation_mins values that included 40+ minutes of queue waiting, inflating the average to 54.1 minutes. Without the gate, this would have reduced capacity from 4 to 3 and driven away potential patients by displaying inflated wait times.

The dual-system architecture (Flask Random Forest + Node.js fallback) provides production resilience without sacrificing prediction quality. During the evaluation period, the Flask service was simulated as offline, and System B produced prediction accuracy within 0.4 minutes MAE of System A results on the same test cases.

VIII. Conclusions

This paper presented MediQueue, a complete hospital queue management system with an integrated self-learning wait time prediction engine. Four primary contributions were validated: (i) a deployment-agnostic ML pipeline requiring no pre-labelled training data; (ii) an equal-distribution formula producing consistent multi-role wait time display; (iii) a full-stack RBAC system covering all stakeholder workflows; and (iv) a data quality gate preventing prediction accuracy degradation.

The system achieved an overall MAE of 2.3 minutes—significantly better than comparable deployed systems—while requiring no pre-existing clinical dataset. The nightly self-recalibration architecture ensures continuous improvement as clinical usage accumulates verified consultation records.

Future directions include: (i) Flask System A integration with online learning; (ii) native mobile application with push notifications; (iii) multi-hospital federated learning; (iv) NLP symptom triage for complexity-weighted wait estimates; and (v) integration with ABHA/Ayushman Bharat digital health identity infrastructure.

Acknowledgments: The authors express sincere gratitude to Neelam Jadhav (Guide, Assistant Professor) and Pradnya Kothawade (HOD, Assistant Professor & Head), Department of Computer Engineering, Genba Sopanrao Moze College of Engineering (GSMCOE), Pune, Maharashtra, India, for their invaluable guidance, technical mentorship, and unwavering support throughout this research. The authors also thank the Department of Computer Engineering, GSMCOE, for providing the computational resources and academic environment necessary for this work.

References

1. S. Soman, S. Rai, and P. Ranjan, "Mobile augmented smart queue management system for hospitals," in Proc. IEEE 33rd Int. Symp. CBMS, 2020, pp. 419–424, doi: 10.1109/CBMS49503.2020.00086.

 doi:10.20944/preprints202604.0628.v1

2. M. Tello et al., "Machine learning based forecast for the prediction of inpatient bed demand," BMC Medical Informatics and Decision Making, vol. 22, no. 1, p. 55, 2022, doi: 10.1186/s12911-022-01787-9.
3. P. Verma et al., "ML based integrated hospital appointment booking and queue management system," IJSREM, vol. 7, no. 12, Dec. 2023, doi: 10.55041/IJSREM27264.
4. A. Author et al., "A machine learning-based approach for wait-time estimation in healthcare facilities," IET Systems Biology, 2024, doi: 10.1049/smc2.12079.
5. M. K. Pandey and D. K. Gangeshwer, "Application of queuing theory to analysis of waiting time in the hospital," Int. J. Bioautomation, vol. 27, no. 3, pp. 139–146, 2023, doi: 10.7546/ijba.2023.27.3.000904.
6. A. Bidari et al., "Effect of queue management system on patient satisfaction in ED," ResearchGate, 2021, doi: 10.21203/rs.3.rs-954637/v1.
7. R. F. Maala, N. B. Sebua, and K. L. L. Evangelista, "Queuing management system in Manuel S. Enverga University Foundation," IJARCS, vol. 14, no. 6, pp. 44–53, 2023.
8. MediQueue Dev Team, "MediQueue: Hospital queue management system—technical documentation," City General Hospital, Pune, Internal Report, 2024.
9. Q. Xu, K.-L. Tsui, W. Jiang, and H. Guo, "A hybrid approach for forecasting patient visits in emergency dept.," Quality and Reliability Eng. Int., vol. 32, no. 8, pp. 2751–2759, 2016, doi: 10.1002/qre.2095.
10. Y.-H. Kuo et al., "An integrated approach of ML and systems thinking for waiting time prediction in an ED," Int. J. Medical Informatics, vol. 139, p. 104143, 2020, doi: 10.1016/j.ijmedinf.2020.104143.
11. E. Benevento, D. Aloini, and N. Squicciarini, "Towards a real-time prediction of waiting times in EDs: a comparative ML analysis," Int. J. Forecasting, Dec. 2021, doi: 10.1016/j.ijforecast.2021.10.006.
12. D. Yaduvanshi, A. Sharma, and P. V. More, "Application of queuing theory to optimize waiting time in hospital operations," Operations and Supply Chain Mgmt.: An Int. J., p. 165, 2019.