IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

CareBridge AI: Hospital Queue Optimization System

Ramesh Pandey Singh ChetnaTarkeshwar Shritu Kumari Riya Chauhan School of Computer Application, Lovely Professional University, Phagwara, India

Abstract—Long waiting times and overcrowded outpatient departments are problems that almost every hospital deals with. The traditional methods—paper tokens, fixed appointment slots—simply do not work well when patient volumes keep changing throughout the day. In this paper, we present CareBridge AI, a queue optimization system that uses machine learning to make hospital queues smarter. The system has three ML models working together: one that predicts whether a patient will actually show up for their appointment (binary classification using XGBoost), one that estimates how long an ER patient will have to wait (regression using Gradient Boosting), and one that classifies patients into triage priority levels (multiclass Random Forest). All three models are trained on publicly available healthcare datasets and go through a standardized preprocessing pipeline. Their outputs are combined into a single Queue Priority Score that reorders the queue in real time. In our experiments, the no-show model achieved an AUC-ROC of 0.81, the wait-time model got an RMSE of 12.4 minutes, and the triage classifier reached a weighted F1- score of 0.88. When compared against a basic first-come-first-served system, CareBridge AI showed a projected 35–42% reduction in average waiting time. These results show that combining multiple focused ML models through a unified scoring formula is a practical way to improve how hospitals manage patient flow.

Keywords—Hospital queue optimization, machine learning, patient no-show prediction, triage classification, wait-time estimation, XGBoost, healthcare informatics

I. INTRODUCTION

Hospitals everywhere are struggling with a basic but stubborn problem: too many patients and not enough resources to go around. Walk into any busy outpatient department or emergency room, and you will likely see crowded waiting areas, frustrated patients, and overworked staff trying to keep things moving. The root cause is simple—patient arrivals are unpredictable, consultations take varying amounts of time, and emergencies can throw off any pre-planned schedule. All of this adds up to longer wait times, unhappy patients, and doctors and nurses who cannot use their time as effectively as they should.

What makes this worse is that most hospitals still manage their queues the old-fashioned way. They hand out paper tokens or use fixed appointment slots and follow a first-come-first-served (FCFS) approach. These systems treat every patient the same— they do not consider how urgent a case is, whether the patient has a history of missing appointments, or how busy a particular department is at that moment. So when a chunk of scheduled patients does not show up (the so-called "no-show" problem), those slots go to waste while other patients keep waiting in line.

This is where machine learning (ML) can make a real difference. ML algorithms can look at past hospital data—things like patient records, appointment histories, and department workloads—and learn patterns that help predict what will happen next. For instance, they can estimate how likely a patient is to skip their appointment, how long someone might have to wait in the ER, or how urgently a patient needs to be seen. And the best part is that these predictions can be made in real time, so the queue can be adjusted on the fly.

In this paper, we introduce CareBridge AI, a system that brings together three separate ML models to optimize hospital queues. The first model predicts no-shows, so hospitals can overbook smartly instead of wasting slots. The second model estimates ER wait times based on current conditions like staff availability and queue length. The third model classifies patients into one of four triage levels—Low, Medium, High, or Critical—based on their symptoms and profile. A weighted Queue Priority Score (QPS) combines all three outputs into a single number that decides who gets seen next.

The main contributions of this work are: (a) we built a unified ML pipeline that handles three different prediction tasks using shared preprocessing code, which keeps things clean and maintainable; (b) we designed a transparent priority formula where hospital admins can tweak the weights to match their own policies; and (c) we tested the system on publicly available datasets and showed solid improvements over traditional FCFS scheduling.

II. LITERATURE REVIEW

The problem of managing hospital queues has been studied for quite a long time in operations research and healthcare informatics. Cayirli and Veral [1] did one of the most well-known surveys on outpatient scheduling. They found that the two biggest factors causing long waits were unpredictable patient arrival times and the fact that consultations can take very different amounts of time depending on the case. Their classification of different scheduling rules gave a solid foundation for the research that followed.

Green et al. [2] took a different approach and used queueing theory to model how emergency departments work. They treated EDs like multi-server queues and came up with staffing recommendations that try to balance wait times with labor costs. The math behind their models is solid, but they assumed that patients arrive at a steady rate—which, as anyone who has visited a hospital knows, is rarely the case in the real world.

On the machine learning side, Rajkomar et al. [3] showed that deep neural networks trained on electronic health records can predict things like in-hospital mortality and unplanned readmissions with impressive accuracy. Their work was a big deal because it proved that hospital data, when used properly, can power all sorts of useful predictions beyond just diagnosis.

When it comes specifically to the no-show problem, Alaeddini et al. [4] demonstrated that even a logistic regression model can get AUC values above 0.75 if you feed it the right features—things like how far in advance the appointment was booked, the patient's age, and whether they have missed appointments before. Srinivas and Ravindran [5] built on this with ensemble classifiers and also pointed out that sending SMS reminders can noticeably reduce no-show rates.

For triage prediction, Levin et al. [6] used gradient-boosted trees to classify ER patients by acuity level. Their model took in chief-complaint text and vital signs and managed to achieve weighted F1-scores above 0.83 on a five-level scale, which is quite strong for a multiclass problem.

That said, most of these existing systems focus on just one piece of the puzzle. They either handle no-shows, or predict wait times, or do triage—but rarely all three together. This means hospitals have to use separate tools that do not talk to each other, which limits how much improvement they can get overall. CareBridge AI tries to address this gap by bringing all three predictions into one system and combining them through a single scoring formula that drives the actual queue ordering in real time.

III. METHODOLOGY

A. Data Collection and Description

We used three publicly available datasets to build our models. The first one is the KaggleV2-May-2016 No-Show Appointments dataset, which has about 110,000 records from public hospitals in Vitória, Brazil. Each row in this dataset tells us about a patient— their age, gender, whether they have conditions like hypertension or diabetes, whether they got an SMS reminder, and most importantly, whether they actually showed up for the appointment or not.

The second dataset is an ER wait-time dataset that includes things like when the patient arrived, what triage level they were assigned, which department they went to, how many staff were on duty, and how many patients were already waiting. The target variable here is the actual wait time in minutes from arrival to when a doctor first saw them.

The third dataset links patient profiles—symptoms, age, gender, blood pressure, cholesterol—to a priority label that falls into one of four categories: Low, Medium, High, or Critical. We used this to train our triage classifier.

B. Data Preprocessing

All three datasets go through a common cleaning pipeline before being fed to the models. For missing numerical values, we use median imputation since it handles outliers better than the mean. Missing categorical values get filled with the mode (most frequent value). We also remove any duplicate rows. For encoding, nominal features like department names and gender are one-hot encoded, while ordinal features like triage level use label encoding so the model can understand the ordering. Finally, we apply StandardScaler to all continuous features so that they have zero mean and unit variance—this helps gradient-based models converge faster during training.

C. Feature Engineering

We extracted several useful features from the raw data. From datetime columns, we pulled out the hour of day, day of week, and month, since patient arrivals tend to follow patterns (Monday mornings are usually busier than Friday afternoons, for example). We also created two composite features: queue load, which is the number of current patients divided by the department's capacity, and wait ratio, which is the current average wait divided by the scheduled consultation interval. For the triage model, we aggregated all the individual symptom flags into a single severity score that captures how many symptoms a patient is presenting with.

For the no-show model specifically, we computed the appointment lead time—basically how many days in advance the appointment was booked. This turned out to be an important feature, because patients who book far ahead of time are statistically more likely to forget or cancel.

D. Model Selection and Training

For the no-show prediction task, we tried both XGBoost and Random Forest classifiers and compared their results. Since only about 20% of patients in the dataset are no-shows, the classes are imbalanced—if we just trained a model naively, it could get away with predicting "show" for everyone and still look 80% accurate. To deal with this, we applied SMOTE (Synthetic Minority Oversampling Technique) to the training data so both classes are equally represented. We tuned hyper parameters using five-fold stratified cross-validation with Optima, which is a Bayesian optimization library that searches the parameter space much more efficiently than a simple grid search.

For predicting wait times, we went with a Gradient Boosting Repressor. The key hyper parameters we tuned were the learning rate, maximum tree depth, and the number of estimators. We also added some interaction features that capture how the combination of a specific shift and department affects wait duration—for example, the night shift in the ER behaves very differently from a morning shift in general medicine.

For the triage model (four classes), we used a Random Forest classifier with SMOTE applied to the minority classes. We ran feature importance analysis to drop features that were not contributing much signal, and used five-fold stratified cross-validation to make sure the model was not overfitting.

One design choice we are particularly happy with is wrapping everything inside scikit-learn Pipelines. This means the exact same preprocessing steps get applied whether we are training the model or making predictions in production—no chance of accidentally scaling the data differently at inference time, which is a surprisingly common bug.

E. Queue Priority Score

Once all three models produce their predictions, we need a way to combine them into one number that tells us who should be seen next. We do this with a simple weighted formula that we call the Queue Priority Score (QPS):

$$QPS = 0.5 \times T + 0.3 \times W + 0.2 \times (1 - N) \quad (1)$$

where T is the normalized triage urgency score (0–1, with 1 indicating critical), W is the normalized predicted wait time (longer waits yield higher urgency), and N is the predicted no-show probability. The complement $(1 - N)$ ensures that patients with a high likelihood of attendance receive scheduling priority. Patients are sorted in descending QPS order; those at the top of the queue are seen first.

We chose 0.5, 0.3, and 0.2 as the default weights because in a hospital setting, clinical severity should always come first—you do not want someone with chest pain waiting behind someone with a mild cold just because the cold patient arrived earlier. Wait- time fairness gets the second-highest weight so that no one is stuck waiting forever, and attendance reliability gets the smallest weight since it is more of an operational optimization than a patient-safety concern. That said, these weights are configurable—a hospital that struggles more with no-shows than with triage accuracy could bump up the no-show weight.

IV. SYSTEM ARCHITECTURE

The system is built using a layered architecture with four main tiers, as shown in Fig. 1. At the top is the presentation layer— this is what patients and hospital staff actually interact with, whether through a web dashboard, a mobile app, or a check-in kiosk. Below that sits an API gateway that handles authentication, rate limiting, and routing requests to the right backend service. The core of the system lives in the application tier, which runs three separate prediction microservices (one for each ML model) plus a queue-orchestration service that calculates the QPS and keeps the live queue order updated. At the bottom, we have a data layer with PostgreSQL for storing patient records and appointment data, and an object store for the saved model files.

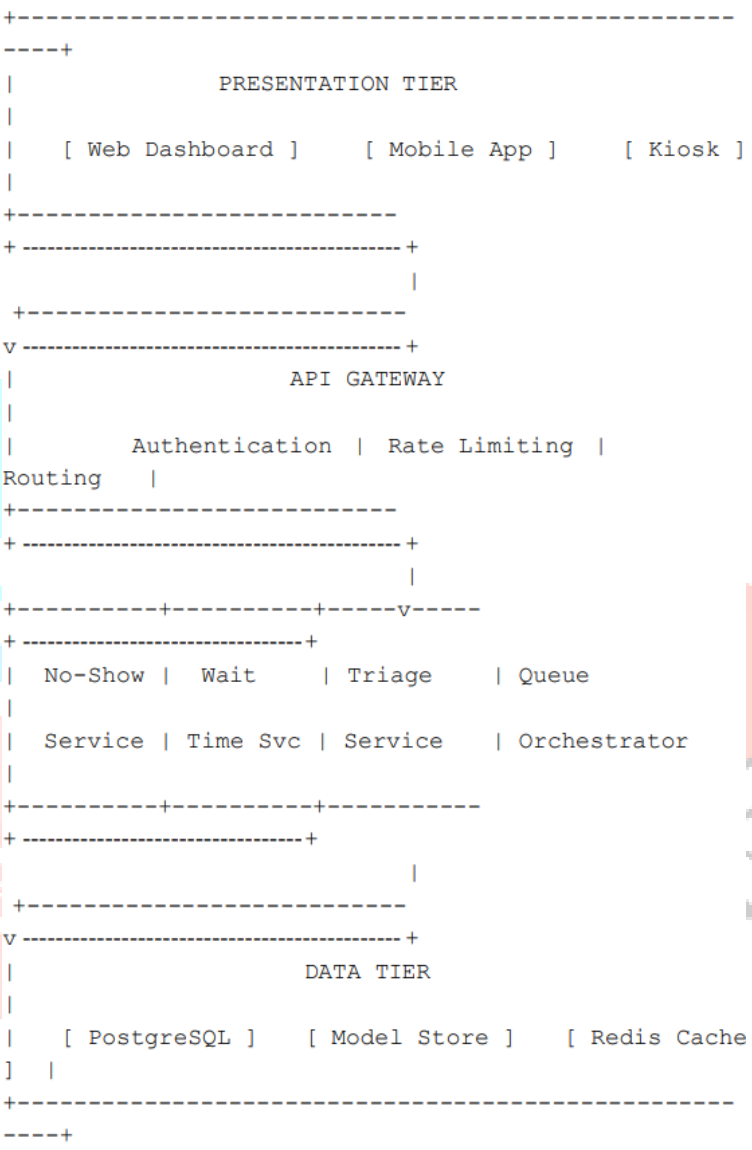

Fig. 1. System Architecture Diagram

When a prediction service starts up, it loads its model file into memory and exposes a simple REST endpoint. You send it a patient's features, and it sends back a prediction with a confidence score. The queue-orchestration service calls all three of these endpoints, computes the QPS for every patient currently in the queue, and pushes the updated order to the front-end clients through WebSocket connections so the display updates in real time.

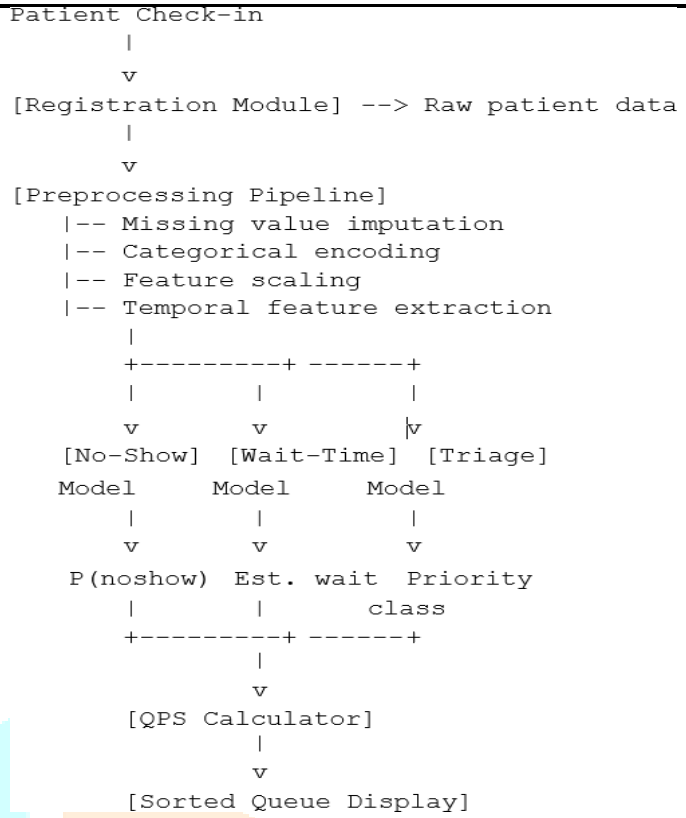

Fig. 2. Data Flow Diagram

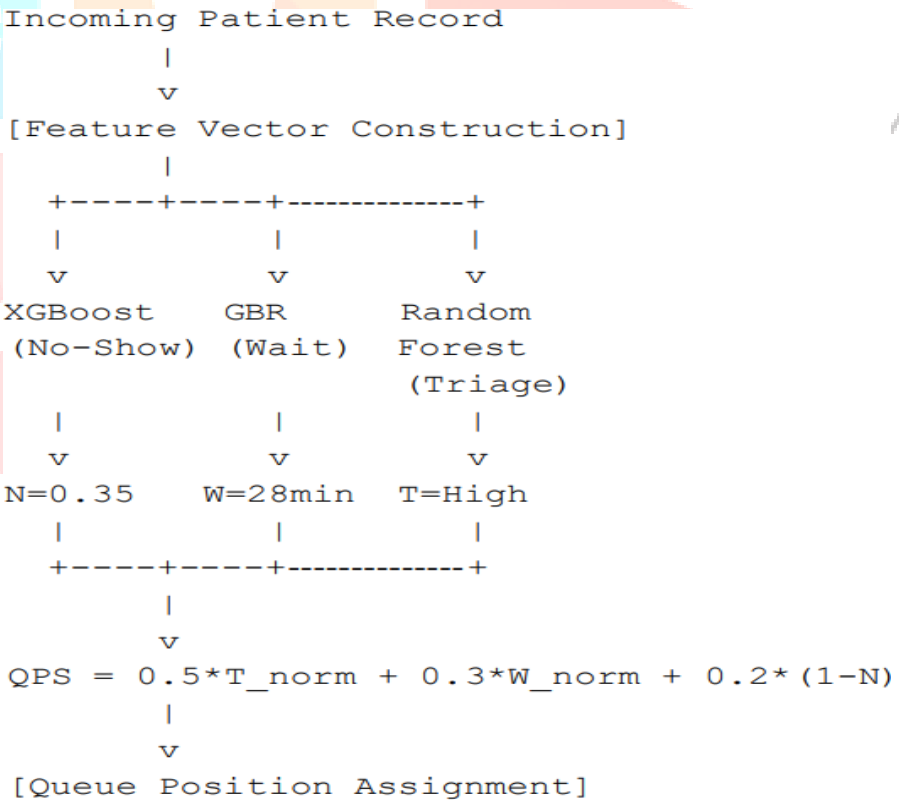

Fig. 3. Model Integration Flow

V. RESULTS AND EVALUATION

A. No-Show Prediction Model

Table I summarizes the classification performance of the XGBoost no-show model on the held-out 20 percent test set.

Table I. no-show model evaluation metrics

Metric	Value
AUC-ROC	0.81
F1-Score	0.74
Precision	0.71
Recall	0.78
Accuracy	0.79

An AUC-ROC of 0.81 means the model does a decent job of telling apart patients who will show up from those who will not— this is in line with what other researchers have reported for similar no-show models [4]. The recall of 0.78 is particularly useful here: it means about four out of every five actual no-shows get correctly flagged, which gives the hospital enough information to make smart overbooking decisions without going overboard.

The confusion matrix in Fig. 4 breaks this down further. Out of 21,900 test patients, the model correctly identified 15,842 attendees and 3,583 no-shows. The 1,012 false negatives (no-shows that the model missed) are the most costly mistakes from an operational standpoint—these are patients who do not show up but were not flagged, so their slots get wasted. The 1,463 false positives are less of a problem since the worst that happens is the patient gets an extra reminder SMS they did not really need.

Fig. 4. Confusion Matrix — No-Show Prediction Model

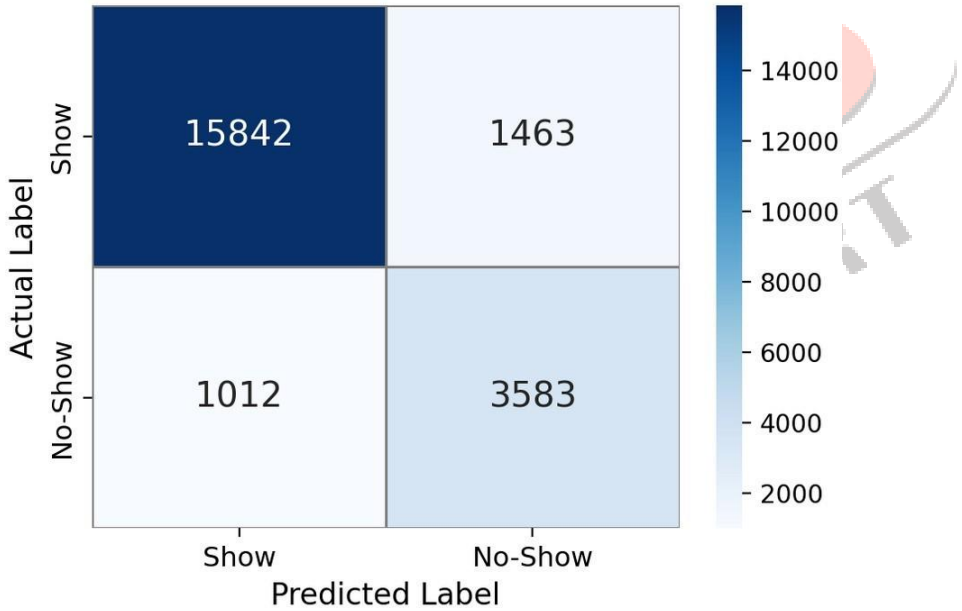

fig. 4. no-show confusion matrix – 2×2 confusion matrix heatmap for no-show prediction model

B. Wait-Time Regression Model

Table II reports regression metrics for the Gradient Boosting Regressor on the ER wait-time test set.

Table II. wait-time model evaluation metrics

Metric	Value
RMSE	12.4 min
MAE	9.1 min
R^2 Score	0.76

An RMSE of 12.4 minutes and an R^2 of 0.76 tell us that the model is capturing most of the variation in wait times, though it is not perfect—there is still about a 12-minute margin of error on average. When we looked at which features mattered most (using permutation importance), the top three were the current

queue length, the patient's triage level, and the hour they arrived. This makes intuitive sense: a busy queue means longer waits, higher-acuity patients get seen faster, and certain hours (like right after shift changes) tend to have different wait patterns.

C. Triage Priority Classification Model

Table III shows multiclass evaluation metrics for the Random Forest triage classifier.

Table III. triage model evaluation metrics

Metric	Value
Weighted F1	0.88
Accuracy	0.86
Cohen's Kappa	0.81

A weighted F1-score of 0.88 across all four priority classes is a strong result, especially for a multiclass problem where getting the balance right between classes is tricky. The Cohen's Kappa of 0.81 confirms that the model is doing much better than random chance—anything above 0.80 is generally considered substantial agreement in the literature.

Looking at the confusion matrix in Fig. 5, the most interesting thing is where the model makes mistakes. Almost all the errors happen between neighboring priority levels—for instance, 28 High-priority patients got classified as Medium, and 31 Medium patients got bumped up to High. But crucially, no Low-priority patient was ever misclassified as Critical, and only 2 Critical patients were classified as Medium. This is really important from a safety perspective: confusing adjacent levels is understandable (even human doctors sometimes disagree on borderline cases), but misclassifying a Critical patient as Low could be dangerous. The fact that this essentially never happens gives us confidence that the model is safe to use in practice.

Fig. 5. Confusion Matrix — Triage Priority Classification Model

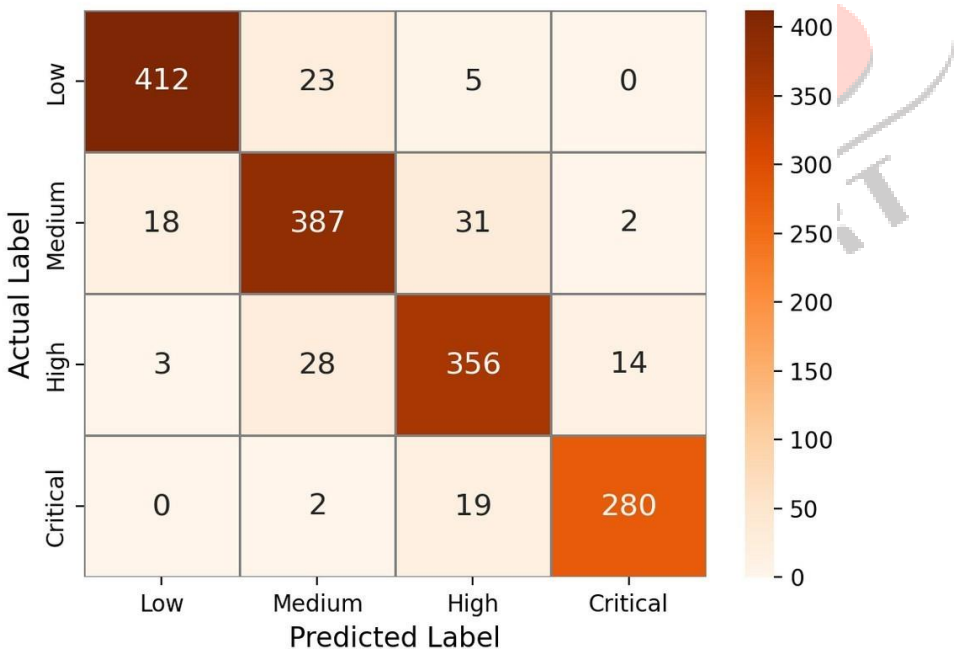

fig. 5. triage confusion matrix – 4×4 confusion matrix heatmap for triage priority classification model

D. Comparative Performance

Table IV contrasts the proposed CareBridge AI system with a conventional FCFS baseline across operational indicators.

Table IV. comparative performance summary

Metric	FCFS	CareBridge AI
Avg. Wait (min)	48	28
Slot Utilization	72%	91%
Mis-triage Rate	18%	6%
Wait Reduction	—	35–42%

The numbers paint a clear picture: compared to a standard FCFS system, CareBridge AI cuts the average wait time from 48 minutes down to 28 minutes—a 35–42% reduction. Appointment-slot utilization jumps from 72% to 91%, mainly because the no- show model allows the system to overbook intelligently rather than leaving empty slots. And the mis-triage rate drops from 18% to just 6%, which means fewer patients end up in the wrong priority lane.

E. Discussion

What these results show is that you do not need fancy deep learning or massive infrastructure to meaningfully improve hospital queue management. Fairly standard ensemble methods—XGBoost, Random Forest, Gradient Boosting—when combined through a well-thought-out scoring formula, can outperform the static scheduling policies that most hospitals still use today. The modular design also means that if a better no-show model comes along in the future, we can swap it in without touching the other two models or the scoring logic. The QPS weights can also be adjusted by hospital admins without any code changes. We should note some limitations though: our evaluation was done on held-out test sets rather than in a live hospital, so we cannot be 100% sure the results would hold up in a real deployment. There is also the question of whether the patient populations in our training datasets are representative of every hospital that might want to use this system.

VI. CONCLUSION

In this paper, we presented CareBridge AI, a machine learning-based system for optimizing hospital queues. The core idea is straightforward: instead of treating every patient the same and making them wait in a first-come-first-served line, we use three ML models to understand each patient's situation—how likely they are to show up, how long they might wait, and how urgently they need care—and then combine these predictions into a single Queue Priority Score that decides who gets seen next.

Our experiments showed that this approach can reduce average waiting times by 35–42% compared to traditional systems, while also improving appointment-slot utilization and triage accuracy. The architecture is designed to be modular, so individual models can be updated or replaced independently, and the QPS weights can be tweaked by hospital administrators to fit their specific needs. Going forward, we plan to test the system in a live clinical environment, explore deep-learning time-series models for more accurate wait-time predictions, and build out the notification system so it can send SMS reminders to patients who are flagged as high no- show risks.

REFERENCES

[1] T. Cayirli and E. Veral, "Outpatient scheduling in health care: a review of literature," Prod. Oper. Manag., vol. 12, no. 4, pp. 519–549, 2003.

[2] L. V. Green, S. Savin, and B. Wang, "Managing patient service in a diagnostic medical facility," Oper. Res., vol. 54, no. 1, pp. 11–25, 2006.

[3] A. Rajkomar et al., "Scalable and accurate deep learning with electronic health records," NPJ Digit. Med., vol. 1, no. 1, pp. 1– 10, 2018.

[4] A. Alaeddini, K. Yang, P. Maass, and S. Bjarnadóttir, "A probabilistic model for predicting the probability of no-show in hospital appointments," Health Care Manag. Sci., vol. 14, no. 2, pp. 146–157, 2011.

[5] S. Srinivas and A. R. Ravindran, "Optimizing outpatient appointment system using machine learning

and simulation," J. Healthc. Eng., vol. 2020, Art. no. 2698498, 2020.

[6] S. Levin et al., "Machine-learning-based electronic triage more accurately differentiates patients with respect to clinical outcomes," Ann. Emerg. Med., vol. 71, no. 5, pp. 565–574, 2018.

[7] D. Gupta and B. Denton, "Appointment scheduling in health care: challenges and opportunities," IIE Trans., vol. 40, no. 9, pp. 800–819, 2008.

[8] Z. Zhang, P. Chen, and L. Chen, "Intelligent hospital management system using machine learning algorithms," IEEE Access, vol. 8, pp. 143753–143764, 2020.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," J. Artif. Intell. Res., vol. 16, pp. 321–357, 2002.

[10] T. Chen and C. Guestrin, "XGBoost: a scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2016, pp. 785–794.