Kings University College (MEM)

USING **MACHINE LEARNING** TO PREDICT G/G/S **QUEUE PERFORMANCE** FROM SIMULATION-BASED **DATASETS**

Aashrith Raj Tatipamula
Mother Teresa Catholic Secondary School
Dr. Rodrigues
Mr. Laszlo
June 25th, 2025

Motivation

Importance of Queuing Systems

- Lots of places rely on Queues (ie. Hospitals)
- long wait times = frustration & inefficiency.
- Efficient queuing leads to:
  - Better resource allocation
  - Reduced costs
  - Improved patient/customer satisfaction

Limitations of Queuing Equations

- Classical formulas (like M/M/s) rely on strict assumptions:
  - Constant arrival/service rates
  - Poisson/exponential distributions
- Real-life queues have:
  - messy arrivals
  - variability in service

Idea

- What if we learned the patterns from data instead of forcing assumptions?
- ML models can predict metrics like Wq without needing all the strict math assumptions.
- More flexible, more accurate, and scalable across systems.

Objective

Build and test a machine learning model that predicts Wq for M/M/s queuing systems.

My Plan:

- Build a simulated dataset with 10,000 data points
- Train multiple models (Neural Networks, Random Forests, etc.)
- Compare performance using metrics like MAE, RMSE and MSE
- Identify the best performing algorithm for Wq prediction

An M/M/s queue is a queuing model where arrivals and service times are exponentially distributed, and there are s parallel servers serving customers from a single queue.

Methodology

Inputs:
- λ (arrival rate), μ (service rate), s (servers), ρ (utilization), Lq (queue length)

Output:
- Wq

- Used Visual Basic for Applications (VBA) to simulate 10,000 rows of M/M/s queue data.
- Each data point was generated by randomly sampling:
  - **ρ:** between 0.5 – 0.99 (realistic system load without system collapse)
  - **λ (Arrival rate):** Between 1 – 20
  - **s:** Random integer between 1 – 10
- **μ (Service rate):** Formula
- **Lq:** Formula
- **Wq:** Allen-Cunneen formula

$$\mu = \lambda / (\rho \times s)$$

$$Lq = \lambda \times Wq$$

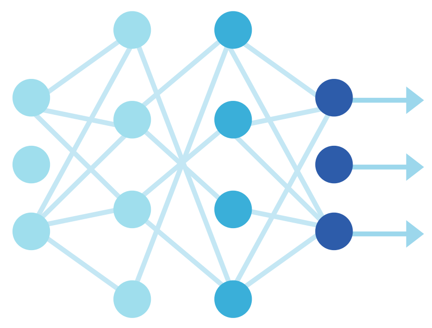
$$W_q = \frac{1}{s\mu} \cdot \frac{\rho\left(\sqrt{2(s+1)} - 1\right)}{1-\rho}$$

Descriptive Analysis for Dataset

Variable	Mean	Std Dev	Min	Max
ρ (Utilization)	0.746	0.142	0.5001	0.9899
s (Servers)	5.49	2.9	1	10
λ (Arrival Rate)	10.49	5.52	1	20
μ (Service Rate)	4.37	5.24	0.1065	38.5716
Wq (Wait Time)	0.87	2.62	0.0045	58.915
Lq (Queue Length)	5.57	11.72	0.0784	95.96

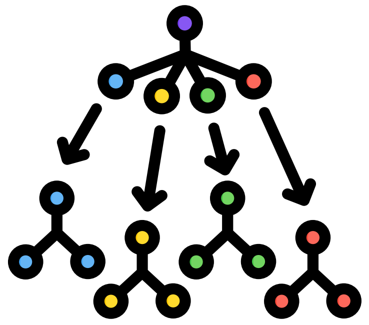
Methodology

Neural Network (TensorFlow Keras)



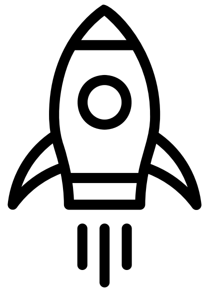
- Architecture: Dense(32) → Dense(16) → Dense(1)
- Activation: ReLU, final layer Softplus
- Optimizer: Adam
- Learning Rate = 0.1
- Epochs: 100, Batch size: 32

Random Forest (Scikit Learn)



- n_estimators = 100
- No max depth tuning (default)

XGBoost



- n_estimators = 100
- max_depth = 6
- learning_rate = 0.1

Evaluation Metrics Used

- MAE – Mean Absolute Error
- MSE – Mean Squared Error
- RMSE – Root Mean Squared Error
- Comparison done on both training and test sets

Results

Actual vs. Predicted Random Wq

Neural Network

Actual Wq	Predicted Wq	Difference
0.3095	0.172997	0.136503
0.1523	0.265574	0.113274
0.0386	0.016812	0.021788
0.1558	0.138838	0.016962
0.1415	0.024299	0.117201
0.3245	0.343307	0.018807
0.0173	0.047095	0.029795
0.1893	0.068019	0.121281
0.3286	0.413843	0.085243
0.0150	0.002284	0.012716

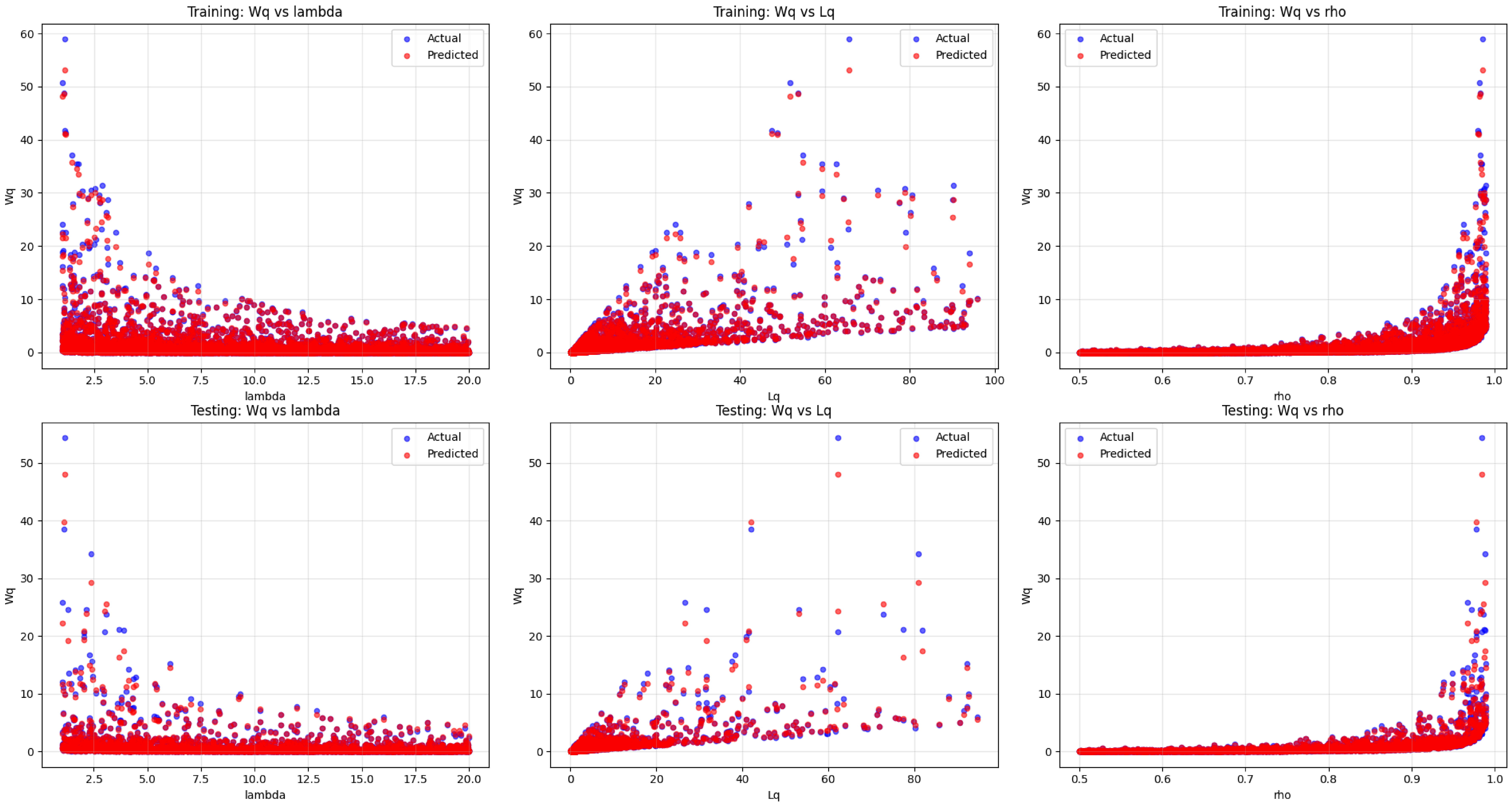
XGBoost

Actual Wq	Predicted Wq	Difference
0.3095	0.312973	0.003473
0.1523	0.151618	**0.000682**
0.0386	0.040916	0.002316
0.1558	0.133239	0.022561
0.1415	0.157413	0.015913
0.3245	0.308196	0.016304
0.0173	0.015558	0.001742
0.1893	0.186924	0.002376
0.3286	0.309357	0.019243
0.0150	0.016093	0.001093

Random Forest

Actual Wq	Predicted Wq	Difference
0.3095	0.307267	**0.002233**
0.1523	0.156843	0.004543
0.0386	0.038860	**0.000260**
0.1558	0.153880	**0.001920**
0.1415	0.138292	**0.003208**
0.3245	0.322262	**0.002238**
0.0173	0.017903	**0.000603**
0.1893	0.189210	**0.000090**
0.3286	0.331816	**0.003216**
0.0150	0.015451	**0.000451**

Plots



Training: Wq vs lambda
Training: Wq vs Lq
Training: Wq vs rho
Testing: Wq vs lambda
Testing: Wq vs Lq
Testing: Wq vs rho
Actual
Predicted
Wq
lambda
Lq
rho

Overall Performance

Training Metrics

Model	MAE	MSE	RMSE
Random Forest	**0.0162**	0.0166	0.1287
Neural Network	0.0982	0.0687	0.2621
XGBoost	0.0173	**0.0014**	**0.0368**

Testing Metrics

Model	MAE	MSE	RMSE
Random Forest	**0.0342**	**0.0642**	**0.2533**
Neural Network	0.0959	0.0729	0.2700
XGBoost	0.0455	0.0818	0.2860

Random Forest

- Lowest error overall
- Captures non-linear relationships
- Robust to outliers

Neural Network

- Decent
- Learns smooth approximations (Softplus)
- Performance consistent with training

XGBoost

- Solid
- Boosting improves performance
- Learns from RF's mistakes

What didn't Work?

Condition	Common Mistakes
Too many low ρ (close to 0)	underestimation of Wq → model understates Wq
Overfitting	Learning rate of 0.1 is pretty high. NN overshot optimal weights. Not setting a max_depth meant some trees grew too deep.
No Cross-Validation (CV)	I used a basic train-test split. K-fold CV would've helped reduce variance
Not Enough Feature Engineering	I used basic parameters (λ, μ, s, ρ, Lq)

Random Forest is the best in both accuracy and reliability.
XGBoost is a close second and would be more tunable for optimization.
Neural Network is still useful for smooth approximations but could improve with more tuning or deeper architecture.

Limitations

- **Synthetic data only:** No real-world hospital or service data used.
- **Only M/M/s queues:** Model doesn't account for G/G/s or multi-phase systems.
- **Limited hyperparameter tuning:** Especially for Neural Network and XGBoost
- **No time-dependent analysis:** Static averages used, not dynamic simulations

Future Work

- **Apply to real hospital queue data:** test model in real-world systems.
- **Variability:** train the ML models with coefficients of variation
- **Compare with Simulation: I**ntegrate with Simul8 to verify accuracy against real simulations.
- **Hyperparameter Optimization :** GridSearch for NN and XGB.

Key Takeways

- ML can accurately predict Wq in queuing systems especially Random Forests.
- Random Forest = reliable: Consistently low error, handles non-linearities well, and doesn't need crazy tuning.
- Neural Networks = okay here: Struggled unless perfectly tuned. Better for more complex patterns or huge data.
- XGBoost = powerful but picky: Performs great, but sensitive to hyperparameters. Not always plug-and-play. Was overfitting.

THANK YOU

Scan QR Code to access Report, Code and Website