EVAT Project: EV Charging Station Congestion Prediction Complete Implementation Plan

Task: Train LSTM & Queue Prediction Models

July 22, 2025

Contents

1 Project Overview

1.1 Objective

Develop a machine learning pipeline to predict congestion at electric vehicle charging stations using the Kaggle EV charging dataset (`station_data_dataverse.csv`, available at this Kaggle link). The system will forecast occupancy rates, queue lengths, and expected wait times using LSTM neural networks combined with queueing theory.

1.2 Dataset Description

The dataset contains **charging session records** with the following key attributes:

- **Temporal (Related to time)**: `created`, `ended`, `startTime`, `endTime`, `chargeTimeHrs`
- **Identifiers**: `sessionId`, `stationId`, `locationId`, `userId`
- **Usage**: `kwhTotal`, `dollars`
- **Context (Additional descriptive information)**: `weekday`, `platform`, `facilityType`
- **Calendar**: Binary flags for days (`Mon` to `Sun`)

1.3 Success Metrics

The following metrics are selected to evaluate model performance based on operational requirements and user experience considerations:

$$\text{Occupancy Prediction:} \quad \text{MAE} < 0.5 \text{ sessions per interval} \tag{1}$$

$$\text{Arrival Prediction:} \quad \text{MAPE} < 25\% \tag{2}$$

$$\text{Queue Wait Time:} \quad \text{RMSE} < 10 \text{ minutes} \tag{3}$$

1.3.1 Metric Justification

Mean Absolute Error (MAE) for Occupancy:

- Measures average absolute difference between predicted and actual number of active charging sessions
- Chosen for its intuitive interpretation: MAE = 0.5 means predictions are off by half a car per interval on average
- Threshold of 0.5 ensures reliable occupancy estimates for operational planning and user availability information

Mean Absolute Percentage Error (MAPE) for Arrivals:

- Measures average percentage error between predicted and actual new charging sessions per interval
- Scale-independent metric that is easy to communicate to stakeholders
- 25% threshold is considered good performance for demand forecasting in operational settings with volatile patterns

Root Mean Squared Error (RMSE) for Queue Wait Time:

- Measures standard deviation of prediction errors, penalizing larger mistakes more heavily
- Critical for wait time predictions where large errors (e.g., predicting 5 min when actual is 30 min) significantly impact user experience

- 10-minute threshold ensures wait time estimates are practically useful for user decision-making

Performance Benchmarks:

Metric	Measures	Why Selected	Threshold Meaning
MAE	Avg. error in occupancy	Simple, interpretable	Off by <0.5 cars/interval
MAPE	Avg. % error in arrivals	Scale-free, intuitive	Off by <25% on average
RMSE	Error in wait time predictions	Penalizes big mistakes	Off by <10 min on average

2 Technical Architecture

2.1 Data Flow Pipeline

1. **Raw Data**: Session logs from `station_data_dataverse.csv`
2. **Preprocessing**: Time-series construction, feature engineering
3. **LSTM Model**: Predicts arrivals (λ_t) and occupancy (O_t)
4. **Queue Model**: Estimates wait times (W_q) and queue lengths (L_q)
5. **Output**: Congestion predictions per station per time interval

2.2 Mathematical Framework

This section details the mathematical approach for constructing time series data, forecasting with LSTM models, and estimating congestion using queueing theory.

2.2.1 Time Series Construction

For each charging station s and each discrete time interval t (e.g., 15-minute bins), we define the following quantities:

$$\text{Arrivals}_t^s = \sum_i \mathbf{1}[\text{created}_i \in [t, t+\Delta t),\ \text{stationId}_i = s] \tag{4}$$

$$\text{Departures}_t^s = \sum_i \mathbf{1}[\text{ended}_i \in [t, t+\Delta t),\ \text{stationId}_i = s] \tag{5}$$

$$\text{Occupancy}_t^s = \sum_i \mathbf{1}[\text{created}_i \leq t < \text{ended}_i,\ \text{stationId}_i = s] \tag{6}$$

Explanation:

- Arrivals_t^s is the number of new charging sessions that start at station s during interval t.
- Departures_t^s is the number of sessions that end at station s during interval t.
- Occupancy_t^s is the number of vehicles actively charging at station s at time t.

This time series forms the basis for both forecasting and congestion estimation.

2.2.2 LSTM Prediction Model

The Long Short-Term Memory (LSTM) neural network is used to forecast future values of occupancy and arrival rates based on historical patterns and contextual features.

Input Sequence:

$$X_t = [O_{t-n}, O_{t-n+1}, \ldots, O_{t-1}, \text{features}_t]$$

where O_{t-k} is the occupancy at previous time steps, and features_t may include hour of day, day of week, facility type, and other contextual variables.

Model Outputs:

$$\hat{O}_{t+1} = \text{LSTM}(X_t; \theta_1) \tag{7}$$

$$\hat{\lambda}_{t+1} = \text{LSTM}(X_t; \theta_2) \tag{8}$$

where:

- $\hat{O}_{t+1}$ is the predicted occupancy for the next interval.
- $\hat{\lambda}_{t+1}$ is the predicted arrival rate (number of new sessions) for the next interval.
- θ_1, θ_2 are the learned parameters of the respective LSTM models.

Training: The LSTM is trained on historical time series data, minimizing loss functions such as Mean Absolute Error (MAE) for occupancy and Mean Absolute Percentage Error (MAPE) for arrivals.

2.2.3 Queueing Theory Application

To estimate congestion (queue length and wait time), we apply the M/M/c queueing model, which assumes:

- Arrivals follow a Poisson process (random, memoryless).
- Service times (charging durations) are exponentially distributed.
- There are c identical charging ports (servers) at each station.

Key Parameters:

- λ = predicted arrival rate (from LSTM), i.e., expected number of new sessions per interval.
- μ = service rate, calculated as $\mu = \frac{1}{\text{mean(chargeTimeHrs)}}$, i.e., average number of sessions a port can serve per hour.
- c = number of charging ports at the station.

Queueing Formulas:

$$\rho = \frac{\lambda}{c \cdot \mu} \qquad \text{(utilization factor)} \tag{9}$$

$$P_0 = \left[\sum_{n=0}^{c-1} \frac{(\lambda/\mu)^n}{n!} + \frac{(\lambda/\mu)^c}{c!} \cdot \frac{1}{1-\rho} \right]^{-1} \qquad \text{(probability system is empty)} \tag{10}$$

$$L_q = \frac{P_0 \cdot (\lambda/\mu)^c \cdot \rho}{c! \cdot (1-\rho)^2} \qquad \text{(expected queue length)} \tag{11}$$

$$W_q = \frac{L_q}{\lambda} \qquad \text{(expected wait time in queue)} \tag{12}$$

Interpretation:

- ρ indicates how busy the station is; if $\rho \geq 1$, demand exceeds capacity and queues grow indefinitely.
- L_q gives the average number of vehicles waiting to charge.
- W_q gives the average waiting time before a vehicle can start charging.

Application:

1. Use the LSTM model to predict λ (arrivals) for the next interval.
2. Estimate μ from historical average charging durations.
3. Use the known c for each station.
4. Compute L_q and W_q using the above formulas to provide real-time congestion and wait time predictions.

Note: If the assumptions of the M/M/c model do not hold (e.g., arrivals are not Poisson, or service times are not exponential), simulation-based queueing models or empirical analysis may be used for more accurate estimation.

3 Implementation Plan

3.1 Phase 1: Data Exploration & Preprocessing

3.1.1 Task 1.1: Dataset Analysis

- Load `station_data_dataverse.csv`
- Examine data types, missing values, and distributions
- Identify unique stations and time range
- Calculate basic statistics (sessions per station, average duration)

3.1.2 Task 1.2: Data Cleaning

- Parse `created` and `ended` as datetime
- Handle missing values and outliers
- Filter stations with sufficient data (¿100 sessions)
- Validate session durations and timestamps

3.1.3 Task 1.3: Time Series Construction

- Create 15-minute time bins for each station
- Calculate occupancy, arrivals, and departures per bin
- Handle overlapping sessions correctly
- Export processed time series as `station_timeseries.csv`

3.2 Phase 2: Feature Engineering

3.2.1 Task 2.1: Temporal Features

Create features for each time interval:

- **Calendar**: hour, day_of_week, weekend_flag
- **Lag features**: occupancy at $t-1, t-2, t-4, t-8$ intervals
- **Rolling statistics**: 2-hour and 4-hour moving averages
- **Station context**: facility_type encoding

3.2.2 Task 2.2: Target Variable Definition

Define prediction targets:

$$y_1 = \text{occupancy}_{t+1} \quad \text{(regression)} \tag{13}$$

$$y_2 = \text{arrivals}_{t+1} \quad \text{(count prediction)} \tag{14}$$

$$y_3 = \mathbf{1}[\text{occupancy}_{t+1} > \text{threshold}] \quad \text{(classification)} \tag{15}$$

3.2.3 Task 2.3: Data Splitting

- **Training**: First 70% of time series per station
- **Validation**: Next 15% for hyperparameter tuning
- **Test**: Final 15% for final evaluation
- Ensure no data leakage across splits

3.3 Phase 3: LSTM Model Development

3.3.1 Task 3.1: Baseline Models

Implement simple baselines for comparison:

- **Naive**: $\hat{y}_{t+1} = y_t$
- **Seasonal naive**: $\hat{y}_{t+1} = y_{t-96}$ (same time yesterday)
- **Moving average**: $\hat{y}_{t+1} = \frac{1}{k}\sum_{i=1}^{k} y_{t-i+1}$

3.3.2 Task 3.2: LSTM Architecture Design

Model Architecture:

```
model = tf.keras.Sequential([
    tf.keras.layers.LSTM(64, return_sequences=True,
                         input_shape=(timesteps, n_features)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.LSTM(32),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1)  # or 2 for multi-output
])
```

Hyperparameters:

- Sequence length: 32 intervals (8 hours)
- Batch size: 64
- Learning rate: 0.001
- Loss function: MAE (regression) or Poisson (counts)

3.3.3 Task 3.3: Model Training

- Train separate models for top 5 busiest stations
- Use early stopping (patience=10)
- Monitor validation loss and avoid overfitting
- Save best model weights

3.4 Phase 4: Queue Modeling

3.4.1 Task 4.1: Service Rate Estimation

For each station, calculate:

$$\mu_s = \frac{1}{\text{mean}(\text{chargeTimeHrs}_s)} \tag{16}$$

$$\sigma_s = \text{std}(\text{chargeTimeHrs}_s) \tag{17}$$

3.4.2 Task 4.2: Queue Theory Implementation

Implement Erlang-C formula:

```
def erlang_c_wait_time(arrival_rate, service_rate, servers):
    rho = arrival_rate / (servers * service_rate)
    if rho >= 1:
        return float('inf')

    # Calculate P0 (probability of empty system)
    sum_terms = sum([(arrival_rate/service_rate)**n /
                    factorial(n) for n in range(servers)])
    last_term = ((arrival_rate/service_rate)**servers /
                (factorial(servers) * (1 - rho)))
    P0 = 1 / (sum_terms + last_term)

    # Expected queue length and wait time
    Lq = (P0 * ((arrival_rate/service_rate)**servers) * rho) / \
         (factorial(servers) * (1 - rho)**2)
    Wq = Lq / arrival_rate if arrival_rate > 0 else 0

    return Wq, Lq
```

3.4.3 Task 4.3: Integration Pipeline

Create end-to-end prediction function:

```
def predict_congestion(station_id, current_features):
    # 1. LSTM prediction
    pred_arrivals = lstm_model.predict(current_features)

    # 2. Queue calculation
    wait_time, queue_length = erlang_c_wait_time(
        pred_arrivals, service_rates[station_id],
        num_chargers[station_id])

    # 3. Return results
    return {
        'predicted_arrivals': pred_arrivals,
        'expected_wait_time': wait_time,
        'queue_length': queue_length,
        'congestion_level': 'High' if wait_time > 15 else 'Low'
    }
```

4 Evaluation Framework

4.1 Model Performance Metrics

4.1.1 LSTM Evaluation

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{18}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{19}$$

$$\text{MAPE} = \frac{100\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{20}$$

4.1.2 Queue Model Validation

Compare predicted wait times against:

- Theoretical M/M/c results
- Simulation-based estimates (if time permits)
- Historical patterns in the data

4.2 Visualization Requirements

1. **Time series plots**: Actual vs predicted occupancy
2. **Error analysis**: Residual plots and error distributions
3. **Station comparison**: Performance across different stations
4. **Congestion heatmap**: Wait times by hour and day of week

5 Deliverables

5.1 Code Deliverables

1. `01_data_preprocessing.py`: Data cleaning and time series construction
2. `02_feature_engineering.py`: Feature creation and data splitting
3. `03_lstm_training.py`: Neural network training and validation
4. `04_queue_modeling.py`: Queueing theory implementation
5. `05_evaluation.py`: Model evaluation and visualization
6. `predict_congestion.py`: End-to-end prediction pipeline

5.2 Documentation

1. **Technical Report**: Methodology, results, and analysis (5-10 pages)
2. **Model Cards**: Performance metrics and limitations for each model
3. **README**: Installation instructions and usage examples
4. **Presentation Slides**: 5-minute demo for stakeholders

5.3 Output Files

1. `station_timeseries.csv`: Processed time series data
2. `lstm_models/`: Trained model weights and configurations
3. `predictions_sample.csv`: Sample predictions for top stations
4. `evaluation_metrics.json`: Performance summary
5. `visualizations/`: Plots and charts for analysis

6 Risk Mitigation

6.1 Technical Risks

Risk	Impact	Mitigation	Contingency
Insufficient data per station	Poor model performance	Focus on top 5-10 busiest stations	Use global model with station embeddings
LSTM overfitting	Poor generalization	Early stopping, dropout, validation monitoring	Switch to simpler models (ARIMA, Prophet)
Queue model assumptions invalid	Inaccurate wait time predictions	Validate against data patterns	Use simulation-based approach
Computational limitations	Training time constraints	Use smaller models, fewer stations	Cloud computing (Google Colab)

6.2 Timeline Risks

- **Data preprocessing delays**: Allocate extra time for data quality issues
- **Model convergence problems**: Have backup simpler models ready
- **Integration challenges**: Test components separately before combining

7 Success Criteria

7.1 Minimum Viable Product (MVP)

1. Working LSTM model predicting occupancy for at least 3 stations
2. Queue model estimating wait times based on predicted arrivals
3. Visualization showing actual vs predicted congestion patterns
4. Documentation explaining methodology and results

7.2 Stretch Goals

1. Multi-station global model with transfer learning
2. Real-time prediction API endpoint
3. Interactive dashboard for congestion monitoring
4. Comparison with advanced models (Transformer, Prophet)