Comparing Classical, Statistical, and Tree-Based Forecasting Models for WMATA Metro Ridership

ECE5424
Rayden Dodd
Department of Electrical and Computer Engineering
Virginia Tech

I. Abstract

This project compares several forecasting approaches for predicting hourly station-level ridership in the Washington Metropolitan Area Transit Authority (WMATA) Metrorail system. Using historical entries data (2012-2019) merged with hourly weather observations for Washington, D.C, I evaluated a simple moving-average baseline, a statistical model based on Prophet, and a tree-based gradient boosting model (LightGBM). The models are trained on pre-COVID ridership (2012–2018), validated on 2018, and tested on 2019, with 2020–2025 excluded due to the ridership break caused by the COVID-19 pandemic. Accuracy is measured using MAE, RMSE, and MAPE, with special attention to the instability of MAPE when actual counts are small or zero. The LightGBM model achieves the best performance, reducing MAE from roughly 260 entries/hour for the baseline to 35 entries/hour and RMSE from about 504 to 90 entries/hour. Prophet sits in between, significantly improving on the baseline but lagging behind the tree-based model. Feature importance and SHAP analyses show that lagged ridership and rolling averages dominate predictive power, while calendar effects, time-of-day, and weather features make smaller but meaningful contributions. These gains suggest that incorporating richer features and nonlinear interactions yields more accurate and operationally useful ridership forecasts than simple seasonal averages, while highlighting the challenges of evaluating percentage errors in low-demand settings. These prediction accuracy gains can have real world benefits. Which can support a more adaptive transit management, informing operational planning (staffing, capacity allocation), and passenger decision making (identifying least crowded travel times). This work demonstrates the practical role of predictive analytics in building resilient urban transit systems.

II. Background

Public transit is the backbone of urban mobility for many cities, and Washington, D.C. is one of these very transit-oriented cities. The WMATA metro is not small, as it is the second largest metro system in the United States, serving 166 million people a year, the D.C. Metro includes a wide range of rider's: residents, daily commuters, federal employees, students and tourists [1]. Forecasting passenger demand is a critical task for WMATA, as usage fluctuates significantly over time due to recurring seasonal patterns, day-of-week effects, weather shocks, and special events. Accurate forecasts allow agencies to anticipate changes in demand and allocate resources more efficiently, balancing service capacity with rider needs. Without reliable predictions, systems are at risk of overcrowding, underutilization, or staffing inefficiencies, all of which reduce the effectiveness and reliability of public transit.

Traditionally, most transit agencies have relied on older, heuristic based approaches for forecasting ridership methods such as professional judgment, rules of thumb, and simple moving averages. While, these techniques remain common, only about 20% of agencies use regression-based models [2]. While, these baselines capture recurring cycles to some extent, they often fail to adjust quickly to shifts caused by disruptions like severe weather, holidays, or large-scale events. As a result, such methods may underestimate demand in critical situations or overestimate demand during downturns, limiting their utility for adaptive management.

However, with recent advancements in machine learning, modeling techniques provide opportunities to improve ridership forecasting accuracy. Some recent time-series regression model advancements like Facebook's Prophet that came out in 2017 incorporate seasonal and holiday effects in its predictions[3]. 2016 tree-based approaches like LightGBM can learn nonlinear interactions between lagged ridership, calendar effects, and external features such as weather [4]. More recently, deep learning architectures such as Temporal Fusion Transformers (TFT) have been proposed for interpretable sequence-to-sequence forecasting, though they are computationally heavier [5]. These hold opportunities for more robust predictions but are more computationally complicated and don't always give benefits over simpler baselines.

WMATA's metro presents unique challenges due to its combination of commuter-driven weekday peaks, weekend tourism traffic, and seasonal variation. This creates a big separation in station use cases as well with some being commuter hubs, nightlife locations, and event locations. Even modest accuracy gains could inform operational planning (staffing, capacity allocation) and passenger guidance (identifying least crowded travel times).

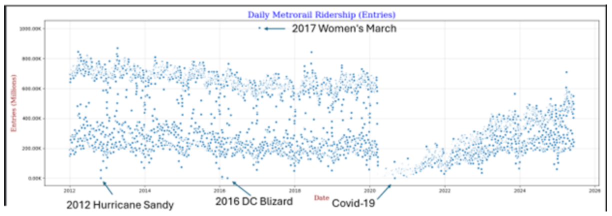

Fig. 1: *Daily Metrorail entries (2012–2025) with notable events annotated (Hurricane Sandy, 2016 Blizzard, 2017 Women's March, COVID-19).*

The WMATA ridership series exhibits all of these challenges and opportunities. Figure 1 shows strong weekday cycles, long-term trends, and large shocks from events such as Hurricane Sandy, major snowstorms, the 2017 Women's March, and, most dramatically, the collapse in demand during the COVID-19 pandemic. Any practical forecasting system must navigate these structural changes while providing useful predictions at the station-hour level.

In this paper I focus on three modeling families of increasing complexity:

- A **moving-average baseline** using several weeks of historical demand.
- A **Prophet model** with multiple seasonality and exogenous regressors (weather and calendar features).
- A **LightGBM gradient boosting model** trained on a rich set of engineered features, including lags, rolling statistics, and weather indicators.

III. Methods

A. Problem Formulation

The forecasting problem can be represented as minimizing the mean squared error of prediction (MSEP) as shown in [6]:

$$\text{MSEP} = \mathbb{E}\left[\left(Y_{t+\ell} - h(Y_1, Y_2, \ldots, Y_t)\right)^2\right],$$

where $Y_{t+\ell}$ is the future value, the goal is to predict future ridership (the true ridership ℓ steps ahead), and $h(\cdot)$is our forecasting function based on past observations of ridership.

The solution to this minimization problem is the conditional expectation:

$$\hat{Y}_t(\ell) = \mathbb{E}(Y_{t+\ell} \mid Y_1, Y_2, \ldots, Y_t, X_t)\,,$$

Where $\hat{Y}_t(\ell)$is the forecast of ridership ℓ steps ahead, the conditioning set $(Y_1, \ldots, Y_t)$represents all past ridership values, and $X_t$denotes exogenous features (weather, holidays, events). This is called the minimum mean squared error (MMSE) forecast, since it gives the best unbiased prediction of the future given past information and auxiliary features.

In practice, I train models to predict hourly station entries one step ahead (and, for some models, multi-step horizons) using historical entry counts and feature vectors X_t.

B. Data

1) Ridership Data

The primary dataset is the WMATA Metrorail ridership summary[7], aggregated to the station–by–hour level. The raw CSV file (Entries_by_Year_Full_Data_data.csv) contains the following fields:

- **Date and time:** Date, Hour
- **Station information:** Station Name
- **Ridership counts:** Entries, Exits, Tap Entries, Tap Exits
- **Calendar attributes:** Day of Week, DAY_TYPE (weekday, Saturday, Sunday), Holiday, Month, and TimePeriod, defined as:
  - **AM Peak:** Open–9:30am
  - **Midday:** 9:30am–3pm
  - **PM Peak:** 3pm–7pm
  - **Evening:** 7pm–12am
  - **Late Night:** 12am–Close

For modeling, I use Entries as the target variable. Redundant or dashboard-specific fields (eg. derived "Avg Daily" measures, Tableau coordinate metadata, and textual annotations) are removed.

A unified **Datetime** variable is created by combining Date and Hour. Because WMATA provides only the exact clock hour, boundaries between AM Peak and Midday both fall on the 9am hour. To resolve this ambiguity and maintain a consistent hourly structure, all overlapping 9am blocks are assigned to **AM Peak**. All station series are then aligned to an hourly resolution for downstream modeling.

2) Weather data and engineered indicators

Hourly weather for Washington, D.C. is obtained from a public API from Meteostat[8] and merged to the ridership data by timestamp. The base weather variables include:

- Temperature (°C)
- Precipitation (mm/hour)
- Wind speed (m/s or km/h)
- Relative humidity (%)

To better capture behavioral responses to weather, I derive several binary or transformed features:

- **Is_Rain:** 1 if precipitation > 0, else 0
- **Is_Freezing:** 1 if temperature ≤ 0°C, else 0
- **Is_Snow:** 1 if precipitation > 0 and temperature ≤ 1°C
- **Is_Hot:** 1 if temperature ≥ 30°C
- **FeelsLike:** a simple wind-chill–adjusted value defined as

$$\text{FeelsLike} = \text{temp} - 0.7 \times \text{wind speed}.$$

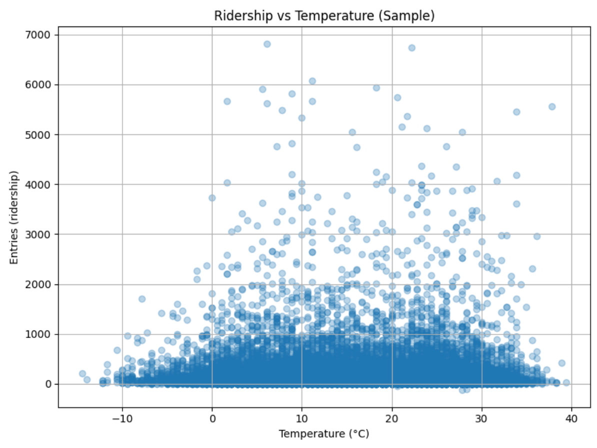

Fig. 2: *Ridership versus temperature in Washington, D.C.*

For example Ridership versus temperature indicates that demand decreases slightly during extreme conditions: very cold hours (below freezing) and very hot hours (above 30°C) show lower ridership relative to moderate temperatures, even after accounting for hour-of-day and day-of-week. These effects are small to ridership but make our models even better. These derived indicators allow the model to capture such threshold effects more effectively than raw temperature alone would.

3) Calendar and time-of-day features

Transit ridership is highly structured by the calendar. Weekday commuting peaks, lower weekend demand, and pronounced morning and evening rush hours produce sharp, non-sinusoidal patterns. To encode this information:

Day-of-week one-hot encoding:

From the DayOfWeek label (Mon–Sun) I create seven binary indicators DOW_Mon, . . . , DOW_Sun.

Weekend indicator:

Is_Weekend = 1 if DAY_TYPE is Saturday or Sunday, else 0.

Month one-hot encoding:

Variables Month_Jan–Month_Dec capture seasonal patterns such as summer tourism or winter weather.

Time period one-hot encoding:

The TimePeriod categories (eg. AM Peak, Midday, PM Peak, Late Night) are converted into binary indicators such as TP_AM_peak, TP_midday, etc.

These one-hot encodings are especially important for Prophet, which expects numeric regressors and models them linearly or additively. Prophet's built-in seasonalities (daily, weekly, yearly) use smooth Fourier terms; they capture gradual sinusoidal patterns but cannot easily represent large discontinuities such as "Saturday demand is half of weekday demand" or "AM peak is a sharp spike relative to surrounding hours." One-hot indicators allow the model to apply separate intercept shifts for specific days and time periods.

4) Hour-of-day as a cyclical feature

Hour-of-day is naturally cyclic (0 and 23 are adjacent), so treating it as a plain integer induces an artificial distance between these hours. For the LightGBM model I use both:

Raw hour: integer 0–23.

Cyclical encoding:

$$\mathrm{Hour}_{\mathrm{sin}} = \sin\left(\frac{2\pi\,\mathrm{Hour}}{24}\right), \qquad \mathrm{Hour}_{\mathrm{cos}} = \cos\left(\frac{2\pi\,\mathrm{Hour}}{24}\right).$$

Using both sine and cosine is essential: together they uniquely encode each hour as a point on the unit circle, preserving the proximity of 23 to 0. Importantly, using only a sine transformation would collapse distinct hours onto the same value (eg. 9 am and 9 pm share the same sine output), but the combination of sine and cosine makes each hour's coordinate unique. In practice, I find that both the raw hour and the sin/cos pair are useful: the tree model can place split thresholds on the raw integer, while the cyclic features help smooth temporal patterns over the day.

5) Lagged and rolling-demand features

Autoregressive structure is extremely strong in ridership: the best predictor of current station demand is often its recent history. For the LightGBM model I construct:

Lagged entry features: lags at 1, 3, 6, 12, 24, 168, and 336 hours. These correspond to within-day dynamics, previous-day patterns, and previous-week or biweekly periodicity.

Rolling mean features: moving averages over past 3, 6, 12, and 24 hours, computed solely from lagged values so that only past information is used.

Conceptually, these features provide the model with both short-term momentum (eg. how busy the last few hours have been) and longer-term seasonal baselines (eg. typical entries at the same station, on the same day-of-week, at the same hour).

For the simple moving-average baseline, I experiment with averaging ridership over many weeks of historical data aligned by station, day-of-week, and hour. Using up to 124 weeks of history yields only marginal gains relative to shorter windows and has the drawback of discarding early observations that lack sufficient historical weeks. For the final baseline I adopted a weekly lag structure consistent with the proposal and note that even a very long moving average is outperformed by learned models.

C. Train–Validation–Test Split

The daily ridership plot Figure1 clearly shows that COVID-19 introduced a massive structural break in 2020: ridership collapses in March 2020 and does not return to pre-pandemic levels even by 2025. To avoid training a single model that straddles two fundamentally different regimes, I restricted the modeling dataset to 2012-2019 and use a expanding window cross-validation split:

Training set: 2012-2017

Validation set: calendar year 2018

Test set: calendar year 2019

This arrangement uses a realistic forecasting scenario: we train on several years of pre-COVID data, tune hyperparameters and apply early stopping using 2018, and then evaluate on the final "normal" year (2019). Data from 2020 onward are used only for descriptive plots and are excluded from any of the model estimations.

D. Models

1) Metrics

Performance is evaluated across models using multiple error metrics:

MAE: Mean Absolute Error

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

MAPE: Mean Absolute Percentage Error

$$\mathrm{MAPE} = \frac{100}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

RMSE: Root Mean Squared Error

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\big(y_i - \hat{y}_i\big)^2}$$

MAE and RMSE are scale-dependent but behave robustly on low counts. In contrast, MAPE is known to perform poorly when actual values are close to zero: dividing by very small y_i can generate extremely large percentage errors and make the metric unstable. This issue has been widely documented; for example, Hyndman and Koehler [9] note that MAPE can be infinite or misleading in the presence of zeros and recommend alternatives such as the mean absolute scaled error (MASE) for series with low or intermittent counts.

In the WMATA setting, many station-hours have very low entries (eg. late-night service at lightly used stations). To mitigate the instability of MAPE, I compute metrics in two ways:

- Over all test observations.
- Over a filtered subset where entries > 20, removing the smallest-demand hours.

The latter provides a more meaningful interpretation of percentage error for hours that actually matter.

2) Baseline: Moving Average

The baseline model computes a moving average over past weeks. For station s and time t corresponding to a given day-of-week and hour, the forecast is

$$\hat{y}_{s,t} = \frac{1}{4}\sum_{i=1}^{4} y_{s,\,t-7i}$$

where W is the number of past weeks included in the average. I experimented with several window lengths (eg. 4 weeks, 16 weeks, and 64 weeks). Larger values of W smooth short-term noise more effectively but adapt more slowly to underlying trends and, importantly, require discarding many early observations that lack sufficient historical depth.

A 64-week window yields the lowest error among the moving-average variants, but the improvement over shorter windows is marginal. In contrast, using a smaller window such as $W = 4$ preserves substantially more data especially for stations with limited early history(eq. new Silver Line stations) coverage while still capturing the dominant weekly seasonal pattern. For this reason, I adopted the 4-week moving average as the final baseline specification.

3) Prophet

Prophet models a time series as a sum of components:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

where $g(t)$ is a piecewise linear or logistic trend, $s(t)$ captures seasonal patterns (daily, weekly, yearly), $h(t)$ represents holiday or event effects, and ε_t is noise.

For each station, I fit a Prophet model using:

- Built-in weekly and yearly seasonalities.
- A learned set of holidays aligned with WMATA's holiday indicator.
- A set of additional regressors capturing weather, calendar, and time-of-day effects.

Table 1 summarizes the exogenous features used as regressors in Prophet.

TABLE I: *Exogenous features used in Prophet*

Group	Example variables	Motivation
Weather (continuous)	Temperature, precipitation, wind speed, relative humidity	Capture weather-driven demand shifts
Weather indicators	Is_Rain, Is_Freezing, Is_Snow, Is_Hot, FeelsLike	Model threshold effects for extreme cold/heat, rain, and snow
Calendar	Holiday flag (0/1)	Distinguish special ridership on holidays
Time-of-day	Hour (0–23)	Capture coarse within-day variation
Day-of-week one-hot	DOW_Mon, …, DOW_Sun	Distinguish weekday vs. weekend patterns
Weekend indicator	Is_Weekend	Capture broad weekend level shifts
Month one-hot	Month_1–Month_12	Capture seasonal variation over the year
Time-period one-hot	TP_AM_peak, TP_midday, TP_PM_peak, TP_late_night, etc.	Reflect different shapes of rush-hour peaks and off-peak troughs

The calendar, month, and TimePeriod categories are one-hot encoded because Prophet treats regressors linearly; a single integer label would imply an artificial ordering (eg. "Monday > Tuesday > Wednesday") that has no meaningful interpretation. One-hot encoding allows Prophet to learn separate coefficients for each day or time period and combine them additively with smooth seasonal components.

This hybrid structure smooth seasonal curves plus sharp indicator shifts is particularly appropriate for ridership, where weekends and rush hours create large step changes relative to baseline demand rather than subtle sinusoidal variations.

4) LightGBM

LightGBM is a gradient-boosted decision tree algorithm. It constructs an collection of trees $f_m(x)$ added sequentially:

$$F_M(x) = F_0(x) + \sum_{m=1}^{M} \eta\, f_m(x),$$

where η is the learning rate. Tree based models are well suited for nonlinear interactions and thresholds, such as sudden ridership drops once temperature falls below freezing.

The LightGBM model is trained on the collection of all station-hour records, with station identity treated as a categorical feature. The input feature set includes the groups summarized in Table 2.

TABLE II: *Key feature groups for the LightGBM model*

Group	Examples	Role in prediction
Current calendar	Hour, Is_Weekend, Month, TimePeriod	Capture structural daily, weekly, and seasonal patterns
Cyclic hour encoding	Hour_sin, Hour_cos	Preserve the cyclic nature of the 24-hour clock
Weather (base)	Temperature, precipitation, wind speed, relative humidity	Reflect weather's direct effect on demand
Weather indicators	Is_Rain, Is_Freezing, Is_Snow, Is_Hot, FeelsLike	Capture nonlinear threshold behavior under extreme conditions
Lagged entries	Entries at 1, 3, 6, 12, 24, 168, 336 hours ago	Autoregressive structure over hours, days, and weeks
Rolling means	Mean entries over the last 3, 6, 12, and 24 hours	Smoothed short-term history
Station identity	Station categorical	Allow station-specific baselines and usage patterns

Compared to Prophet, I purposely avoid one-hot encoding for DayOfWeek and month in the LightGBM setup. Tree based models can naturally split on continuous or integered features, and introducing alot of binary columns can increase dimensionality without any real benefit. The combination of integer hour, cyclic encoded hours, and explicit indicators such as **Is_Weekend** and **TimePeriod** provides good flexibility for the model to look at complex calendar and behavioral patterns.

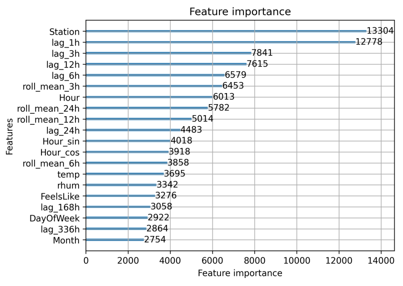

Fig. 3: *Global feature importance ranked by gain.*

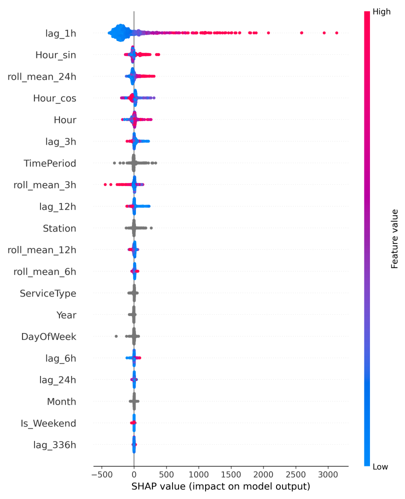

Fig. 4: *SHAP summary plot.*

The most important features, as revealed by LightGBM's built-in feature-importance scores and SHAP summary plots, are the lagged, rolling-entry features, cyclic hours, and time period. In particular, the 1-hour lag and 24-hour rolling which accounts for much of the model's predictive power. This confirms the idea that recent station history is a strong predictor of future demand. Calendar and weather also features provide secondary refinements: for example, the model learns to adjust expectations under extreme cold (**Is_Freezing = 1**) or during low demand weekend periods.

IV. Experiments Design

For this paper I use a straightforward forward-chaining split:

- **Training:** 2012-2017
- **Validation:** 2018
- **Test:** 2019

All features that depend on past ridership (lags, rolling means, etc.) are computed using only earlier timestamps.

Models are compared on the shared 2019 test set using MAE, RMSE, and MAPE. Because MAPE becomes unstable when entries are near zero, I report two versions of each metric:

- **All test station-hours**, including many low-demand hours.
- **Filtered test set with Entries** > 20, focusing on hours where ridership is operationally meaningful.

This setup keeps the evaluation simple while preserving the natural temporal ordering of the data.

V. Results and Discussion

Table 3 reports the performance of each model on the full 2019 test set.

TABLE III: *Forecast accuracy on full test set (all entries)*

Model	MAE	RMSE	MAPE (%)
Baseline	259.5	504.2	1574.6
Prophet	103.5	280.1	83.3
LightGBM	35.2	90.0	68.4

The moving average baseline performs poorly. On average it is off by about 260 entries per station-hour, which is large relative to typical hourly flows. Prophet substantially improves accuracy, cutting MAE by roughly 60% and RMSE by about 45% compared to the baseline. LightGBM delivers the best performance, reducing MAE by approximately 86% and RMSE by 82% relative to the baseline.

However, the MAPE values in Table 3 are almost meaningless. The baseline MAPE exceeds 1500%, and even LightGBM's MAPE remains very high (68%). This behavior results from many almost-empty hours from late-night hours or low-use stations. When y_i is near zero, even a small absolute error produces a huge percentage error, causing MAPE to be dominated by these observations. This issue is well documented in the forecasting literature: percentage-based measures can be infinite or unstable when actual values are zero or close to zero [9].

To address this, I recomputed metrics on a filtered test set where Entries > 20.

TABLE IV: *Forecast accuracy on test set with Entries* > 20

Model	MAE	RMSE	MAPE (%)
Baseline	270.2	531.8	154.7
Prophet	124.2	309.0	36.0
LightGBM	41.0	98.9	19.8

Once extremely low-entry hours are removed, MAPE becomes more interpretable. Among moderately busy to very busy station-hours:

- LightGBM achieves a MAPE of about 20%, which is a reasonable range for operational forecasting.
- Prophet's MAPE is roughly 36%, a clear improvement over the baseline (155%) but less accurate than LightGBM.
- The ordering of models is the same across MAE, RMSE, and MAPE.

Interestingly, MAE and RMSE remain similar between Tables 3 and 4 for each model, showing that absolute errors are driven more by high-demand periods than by near zero hours entries. The dramatic change appears in MAPE, reinforcing that percentage errors must be interpreted cautiously in the presence of many small actual values and should ideally be reported alongside scale-based metrics.

A. Compare Models

When looking at the three models we can observe difference with increasing model complexity increases prediction accuracy.

1) Baseline Vs Prophet

The 4 week moving average baseline assumes that ridership at a given station, DayOfWeek, and hour is relatively stable from week to week. This works tolerably well for simple periodic patterns but doesn't work when:

- trends shift shift (months, seasons, station decline) ,
- holidays or special events can cause sharp increases
- weather strongly decreases or increases demand on particular days.

Prophet improves on these weaknesses in two main ways:

- **Flexible trend and seasonality** The combination of a piecewise linear trend and multiple seasonal components allows Prophet to capture long term drifts and recurring weekly or yearly patterns more effectively than a fixed moving average.
- **Exogenous regressors** Weather variables, holiday flags, and calendar or time-of-day indicators giving the model additional information about days and hours when typical patterns can start to shift.

This increased complexity explains: Prophet reduces MAE from roughly 260 entries per hour to about 100.

2) Prophet vs LightGBM

To understand how LightGBM forms its predictions, I looked at both global feature importance and SHAP summary plots. Figures 3 and 4 illustrate the dominant predictors across all station-hours.

As shown earlier in Figures 3 and 4 both plots tell a consistent story: **recent station history overwhelmingly dominates model predictions**. The most influential feature is **lag_1h**, followed by other short-horizon lags (lag_3h, lag_6h, lag_12h) and rolling means (roll_mean_3h, roll_mean_6h, roll_mean_12h, roll_mean_24h). These features summarize how busy the station has been in the previous few hours and form the core of the model's forecasting ability.

Longer-term lags such as **lag_24h**, **lag_168h** (one week), and **lag_336h** (two weeks) also appear, but with noticeably smaller importance, suggesting that once the model knows what happened recently, additional long-range seasonal context provides only marginal improvement.

As a concrete example, Figure 5 shows one of the first trees in the LightGBM ensemble, illustrating how the model splits on lagged entries and weather thresholds.

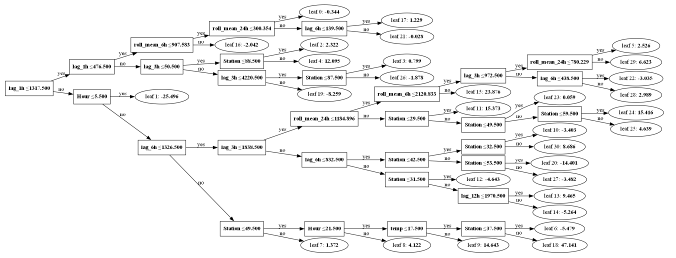

Fig. 5: *Example LightGBM tree (Tree 73) highlighting splits on lagged entries and weather features.*

Cyclical hour features **Hour_sin** and **Hour_cos**, along with the raw **Hour** variable, are consistently used to shape the daily pattern. In the SHAP plot, high values of **Hour_sin**, **Hour_cos**, and **TimePeriod** correspond to predictable peaks (eg. morning and evening rush periods), demonstrating that the model learns time-of-day structure and uses it to adjust the expected ridership level outside of immediate lag-based signals.

One notable difference between the two interpretation methods is the role of the **Station** feature. In the global feature-importance ranking it appears as the most important variable, yet in the SHAP summary plot it falls much lower. This discrepancy reflects the distinction between how the two diagnostics measure importance.

LightGBM's feature-importance metric counts how often a feature is used to split the data and how much those splits reduce the training loss. Because **stations** differ dramatically in their average ridership levels, many trees make early splits on the Station identifier to separate high-volume stations (eg. Metro Center, Gallery Place) from low-volume stations. These early splits yield large reductions in error, which pushes Station to the top of the global importance ranking.

In contrast, SHAP values measure the *marginal impact* of a feature on individual predictions. Once the model has accounted for recent ridership through lagged and rolling features, the Station effect is often a relatively small adjustment to the final prediction. This produces modest SHAP values even though Station was frequently used to structure the trees. SHAP therefore places Station much lower than the lagged and rolling-entry features, which directly drive variation in hour-to-hour forecasts.

VI. Conclusion

This project set out to compare three forecasting approaches for WMATA Metrorail hourly station entries: a simple moving-average baseline, a Prophet model with exogenous regressors, and a LightGBM gradient boosting model with extensive feature engineering. Using pre-COVID data from 2012–2019, merged with hourly weather observations, I trained models on 2012–2017, used 2018 for hyperparameter tuning, and evaluated final performance on 2019.

The results show a clear pattern: as model complexity and feature richness increase, forecast accuracy improves alot. Prophet, with its addition of trend seasonality structure and calendar/weather regressors, cuts MAE and RMSE roughly in half compared to the basic moving average baseline. LightGBM, using lagged entries, rolling means, cyclic hour encodings, and weather and holiday indicators, performs best overall, reducing MAE by about 86% and RMSE by more than 80% relative to the baseline on the full test set.

Feature-importance and SHAP analyses reinforce this story. Autoregressive features (lags and rolling windows) are the main drivers of accuracy, while calendar and weather features act as secondary adjustments, especially during weekends and or under extreme temperatures. A reasonable design pattern for operational ridership forecasting is therefore: start with strong autoregressive structure, and then layer on calendar and weather effects to handle systematic shifts.

The experiments also highlight an important evaluation issue: MAPE behaves poorly when many observations have very low counts. On the entire test set, percentage errors are dominated by close to zero entry hours, producing inflated and unstable MAPEs. Filtering the evaluation to hours with Entries > 20 yields much more usefull percentage errors while preserving the same ranking of models.

Overall, this project shows that relatively accessible machine learning approaches like LightGBM, combined with targeted feature engineering and careful evaluation, can substantially improve metro ridership forecasts over simple historical averages, while still keeping the models interpretable enough to inform real-world planning and operations. More accurate forecasts can support WMATA in allocating staff and train capacity more efficiently, reducing overcrowding during peak periods and minimizing waste during low-demand hours. Riders also benefit through improved reliability, better crowding information, and more predictable travel conditions.

VII. Future Work

There are several promising directions to extend this project. Due to time constraints, I was not able to implement the originally planned Temporal Fusion Transformer (TFT) model. Given LightGBM's strong performance and the central role of lagged features, a natural next step is to evaluate whether deep sequence models such as TFT can provide additional gains, particularly for multi step forecasting, probabilistic predictions, and station-cluster specific models that capture structural differences across the network.

Another important extension is to incorporate post-COVID ridership data. The pandemic introduced a long lasting ridership shift in travel behavior, with weekday commuting patterns, telework, and weekend leisure travel all changing significantly. Modeling this period likely requires ”regime” switching methods, hybrid approaches, or separate pre and post-COVID models to account for new demand baselines and altered seasonality.

Even better exogenous features would also improve predictions. The Washington, D.C. region hosts many large scale events such as sports games, concerts, parades, and political demonstrations. Integrating event calendars, stadium schedules, and protest times could help explain sudden spikes at specific stations (eg. Navy Yard–Ballpark, Gallery Place, Smithsonian). Additional spatial features like population density, land use, or transfer hub identifiers could further enhance station specific predictions.

Finally, a practical extension would be to build an interactive visualization or GUI. Such a tool could allow users to select a station, choose a day and time (eg. “next Tuesday at 5pm”), and immediately view predicted forecasted entries along with uncertainty intervals. This would make the modeling results more accessible for planners, operators, and even riders, transforming the analysis into a usable support system.

Together, these extensions would push the work from a focused forecasting study toward a more comprehensive, operational ridership prediction platform.

VIII. Code Availability

The full codebase for data processing, modeling, and analysis is available at: https://github.com/RaydenDodd/WMATA-Forecasting

IX. Key references

References

[1] American Public Transportation Association, “Ridership report, q4 2024,” https://www.apta.com/wp-content/uploads/2024-Q4-Ridership-APTA.pdf, 2025, [Online].

[2] Transportation Research Board, “Guidelines for providing access to public transportation stations, appendix b: Assessment of evaluation tools,” https://onlinepubs.trb.org/onlinepubs/tcrp/tcrp_rpt_153AppendixB.pdf, 2011, [Online], see p. B-8, Table B-2.

[3] C. Duong, “facebook/prophet in 2023 and beyond,” https://medium.com/%40cuongduong_35162/facebook-prophet-in-2023-and-beyond-c5086151c138, 2023, [Online].

[4] Microsoft Research, “Lightgbm: Publications,” https://www.microsoft.com/en-us/research/project/lightgbm/publications/, 2025, [Online].

[5] B. Lim, S. Ö. Arik, N. Loeff, and T. Pfister, “Temporal fusion transformers for interpretable multi-horizon time series forecasting,” *International Journal of Forecasting*, vol. 37, no. 4, pp. 1748–1764, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169207021000637

[6] J. Tebbs, “STAT 520 Forecasting and Time Series: Lecture Notes,” https://people.stat.sc.edu/tebbs/stat520/f13notes.pdf, 2013, [Online], see Section 9.1, p. 232.

[7] Washington Metropolitan Area Transit Authority, “Wmata ridership data portal,” https://www.wmata.com/initiatives/ridership-portal/, 2025, accessed: 2025-02-14.

[8] Meteostat, “Meteostat weather and climate api,” https://meteostat.net/en/api, 2025, accessed: 2025-02-14.

[9] R. J. Hyndman and A. B. Koehler, “Another look at measures of forecast accuracy,” *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0169207006000239