Ahm Graphin' Hee-uh: Utilizing Spatio-Temporal GCNs in Subway Volume Predictions

Adin Pepper Fox, Christopher Parvankin, Matthew DaSilva

May 2, 2025

Abstract :

New York City Subway passenger volumes vary wildly across different stations and different times of the day. These passenger volumes are the result of a number of spatial and temporal factors, such as the location of the station, day of the week, hour, etc. In this paper, we build upon to work of Yu et al., 2021 to propose a unique method for predicting subway ridership volumes through the use of a Deep Spatio-Temporal Convolution Network Model. Incorporating a diverse set of datasets, including New York City Weather, historic ridership volumes, and subway spatial information, our model was fairly successful in prediction, yielding an average of .0855 Mean Absolute Error on testing data and .0808 Mean Absolute Error on training data. For the relevant temporal data, we trained on the year 2023 and tested on the year 2024. This technology can have particular utility for the MTA who can utilize such to optimize subway resource allocation (energy, train scheduling, crowd control etc.). Our work showcases that such a model can be relatively successful for regressive prediction tasks; if we were to continue, we hope to improve upon the model by incorporating additional data like large local events, using batched graph convolution to optimize GPU usage, and creating more complex graph representations that better embed edge information.

1 Introduction

New York City's Subway System faces volatile passenger volumes dependent on a number of variables — station servicing, time of day, weather, season, etc. Finding a means to reliably predict passenger volume can bring particular utility to the MTA. By knowing when and at what stations high subway volume will likely occur, the MTA can preemptively allocate resources such as security officers or station personnel to assist and guide riders. This can not only relieve MTA resource burdens by optimizing resource allocation, but it can also bring better safety to riders in these particularly voluminous ridership periods.

From a rider's perspective, having knowledge of future subway volumes can inform them of the potential crowding and delays in their trip, allowing them to replan accordingly and potentially lessen the burden on the subway infrastructure during high-volume hours.

With these motivations in mind, our team set out to create a deep learning model that can accurately predict hourly passenger volumes for every NYC Subway station. To achieve this, our team collected a number of datasets —incorporating information like NYC weather, historic NYC Subway ridership volumes, and NYC Subway geographic information — and applied them to a Deep Spatio-Temporal Graph Network Model. This model builds upon the work by Yu et al., 2021, who created a similar model for car accident prediction using PyTorch. Making some of our own adjustments to this model, we implemented the project using Tensorflow, incorporated more complex data embedding that used RNNs, and adjusted the architecture to output regressive numerical prediction instead of binary classification.

2 Methodology

Since the subway network can be modeled as a graph where each node is a subway station and there are edges between subsequent stations, we saw fit to use a spatio-temporal graph convolutional network (ST-GCN).

2.1 Data Collection

As a group, we decided to use five different types of data for the prediction model: hourly turnstile usage at each subway stop, hourly weather data, individual subway station data, public transit's daily ridership, and temporal data (day of the week, month, and hour of the day).

Our primary dataset is Metropolitan Transportation Authority, 2020-2024, which contains the hourly turnstile counts of each subway station in New York City. We used each hour's data as its timestamp's ground truth and passed in the prior 24 hours as context for each prediction. This data was passed into our spatio-temporal layer as defined in the architecture subsection.

Our secondary dataset was Data.gov, 2023, which has useful information about each subway station: its lines, coordinates, schedules, and station ids. Particularly, we used the stops and schedules data to populate our input graph's nodes and edges, which will be elaborated further in the preprocessing subsection. This data was passed into our spatial layer as defined in the architecture subsection.

Outside of direct subway data, we were interested in exploring how other external factors could influence or predict subway usage at each station. In particular, we used the National Weather Service, 2023-2024 dataset to collect hourly weather data (humidity, temperature, precipitation, dew point, etc.) and the aforementioned temporal features. We passed sequential weather data from the prior 24 hours through a GRU to capture an embedding for the next hour. Additionally, we used the **MTA** dataset to collect data about the daily ridership on various forms of transport in New York (LIRR, buses, tunnels, subway, etc.). At each time step, we passed in the ridership from the prior day through a fully connected layer to obtain an embedding that hopefully improves the model's predictions by considering the population totals over the given timeframe.

2.2 Preprocessing

Constructing the graph that will be convolved by the model required various sorts of data. Utilizing the Data.gov, 2023 dataset,

we gathered a csv of the train routes that occurred in 2023 and a csv of subway station ids. Each subway station id serves as a node and the edges occur between subsequent stations on routes. This meant iterating through the routes, and populating edges where consecutive stops occur. For possible future uses, we also encoded the number of lines (and which ones) that connected two stations). We further populated each node with 'structural' information, such as coordinates, name (for easy id), borough, and lines. In order to populate the spatial input for the model, we one-hot encoded the subway lines and borough for each node.

For the turnstile data, we downloaded the portion of the Metropolitan Transportation Authority, 2020-2024 dataset containing the years 2023 and 2024. This data consisted of near-hourly counts of subway riders' use of different kinds of payment (OMNI, a ticket, etc.). Aggregating the counts told us the amount of people at each timestamp arriving hourly (counted at almost every hour) at every subway station. Upon review, it seemed that missing counts occurred exclusively between the hours of 12 and 4 a.m. Additionally, they were far more common at less populated subway stations, the missing timestamps were completely irregular, and no station had 0 counts for any given T=24 time range, making it clear that it was not a product of defunct machinery. So, we safely assumed that we could fill in the missing turnstile counts with 0s.

To prepare these turnstile counts as the spatio-temporal input, we created a 3D tensor with the transfer and entry counts for each station at each time step. Once inputted into the train and test functions, we then perform a "window" operation to collect the previous 24 hours of ridership context at each station for the given timestep. This was passed in with the adjacency matrix, and since the stations are in the same order along their dimension, we can perform convolutions on the traffic context in surrounding stations to make predictions at each station for the next hour. The ground truth at each timestamp and station is the turnstile count, so we use the row with the counts across all stations as the ground truth for our model's prediction given a timestamp and its corresponding data.

While cleaning and preparing the turnstile data for input, we limited the stations' counts to between $(0, 1)$ using per-station min-max normalization:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}.$$

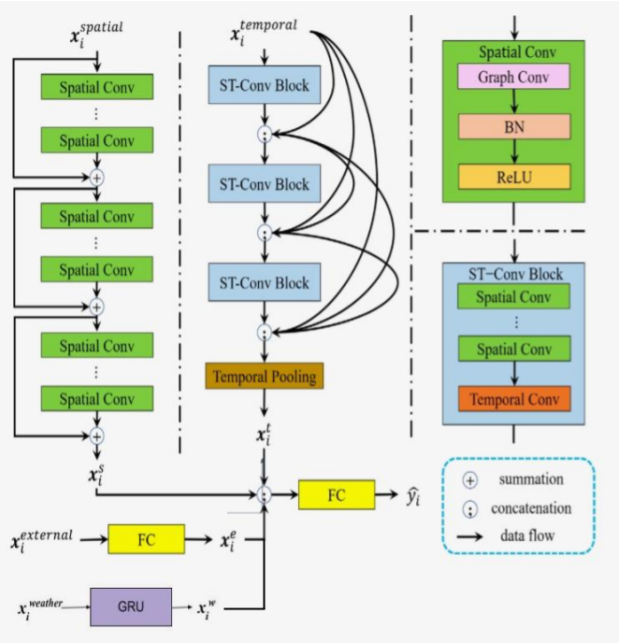
This was in order to prevent exploding gradients and more popular stations from being over-prioritized; they are likely to have larger error simply due to the massive difference in terms of people traffic. We stored the minimum and maximum counts for each station in 2023 as features of its node in order to later convert the normalized predictions and ground truth into meaningful subway traffic counts if desired.

Using the National Weather Service, 2023-2024 dataset, which was recorded at JFK airport, we were able to download hourly weather information for the 2023-2024; it is measured consistently at the top of every hour! This data contained features such as temperature, dew point, humidity, cloud coverage, precipitation, and several other metrics. Of course, each of these categories' columns were min-max normalized. Alongside the weather information, we also one-hot encoded the hour and month to further use non-spatial but temporal data. Once passed into training or testing, we used the same aforementioned "window" operation to collect the 24 hours of context for each timestamp and pass it into the model as weather context.

Finally, we utilized the Metropolitan Transportation Authority, 2020-2025 dataset to incorporate other external features that may impact subway ridership in a given timestep. Each column of the data was normalized and the day of the week was one-hot encoded to serve as non-hourly and non-spatial data that will be passed through a fully connected layer to obtain an embedding.

2.3 Architectural Overview

For our model's design, we largely utilize the architecture outlined in Yu et al., 2021. This architecture consists of four separate layers, whose results are concatenated and passed through a fully connected layer to obtain a $(426, 1)$ vector that contains normalized predictions for each subway station's turnstile count in the next time step. Evidently, our graph contains 426 subway stations. Adapted from a diagram in Yu et al., 2021, the general architecture of the ST-GCN is pictured below:



For the spatial layers, we inputted our graph's feature matrix (number of nodes, dimension of nodes) through a series of two dense layers to obtain a spatial embedding. Then, we passed the spatial embedding and the graph's adjacency matrix through a series of Spatial Convolution blocks. As outlined in the diagram, the input is passed through three spatial convolution blocks, followed by a residual connection with the input itself; three more spatial convolution blocks, followed by a residual connection from the output of the prior series; and a final four spatial convolution blocks, followed by a residual connection from the output of the prior series. Each individual spatial convolution block contains a graph convolution, batch norm, and ReLu activation.

For the spatio-temporal layers, we pass the adjacency matrix of the subway graph and spatio-temporal (ST) data matrix through three spatio-temporal convolution blocks and temporal pooling. The shape of the ST matrix is (number of nodes, ST dimension,

number of time steps). Essentially, for each node, we are inputting the information from the past 24 hours. Each spatio-temporal convolution (STC) block has four spatial convolution blocks followed by a temporal convolution block. The output of each STC block is concatenated to the prior outputs and original ST input. The final concatenation is passed through a round of average temporal pooling.

The temporal but non-spatio data was inputted as the weather data, which is of shape (context hours, weather dimension), is passed through a type of RNN (GRU). Besides hourly weather, we also pass in the hour of day and month as features. The weather data for the previous 24 hours is passed through GRU to obtain an embedding for the next hour. We simply input the ridership data from the previous day into a fully connected layer to obtain an embedding for the next hour.

The four outputs (spatial embedding, ST embedding, weather embedding, external embedding) are concatenated and passed through a fully connected layer to obtain a (426, 1) vector that predicts the turnstile count at every subway station for the next hour.

2.4 Train-Test Split

Because our model is time-dependent, we need to be mindful about the overlap between training and testing data. We begin by splitting the data into 2023 and 2024 for training and testing, respectively. Since, at each time step, the model took in the previous 24 hours of context, we want to ensure that we prevent overlap between the three datasets in terms of prior context. To consistently collect 24 hours of prior context, we eliminated the sampling of timestamps for January 1. We intentionally chose a 50-50 split to ensure that the model generalizes well to an entire year given an entire year's worth of data.

2.5 Metrics

We utilize mean absolute error (MAE) as our loss function for the model to measure how far our predictions were from the actual values:

$$L = \frac{1}{n}\sum_{i}^{n} |y_i - \hat{y}_i|.$$

We do not use mean squared error (MSE) to compute loss because we do not necessarily want to penalize large differences more than small differences. Additionally, experimentally we have found that MAE performs better on the task in terms of average margin of accuracy.

We opted for Adam as the optimizer with a learning rate of 0.001 to produce meaningful yet stable gradient updates. We utilize a margin accuracy metric to gauge what portion of the model's predictions are within 5% of the true subway turnstile count.

3 Results

We trained over multiple different epoch lengths to observe for overfitting, and provide the training and testing results for 8 and 10 epochs with a batch size of 64 and 128 respectively.

8 epoch graphics

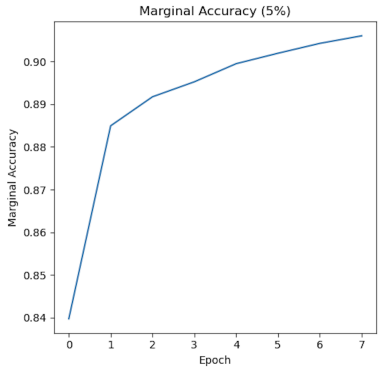


Figure 1. Marginal Accuracy (5% threshold) Over 8 Epoch Training Run

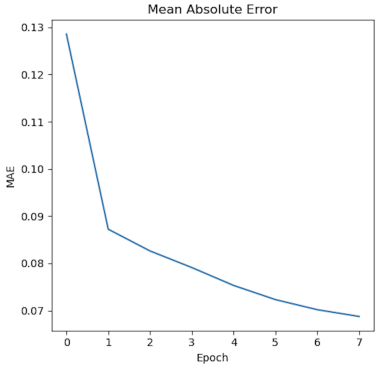


Figure 2. MAE Over 8 Epoch Training Run

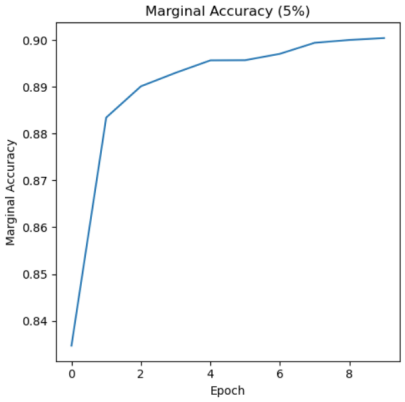


Figure 3. Marginal Accuracy (5% threshold) Over 10 Epoch Training Run

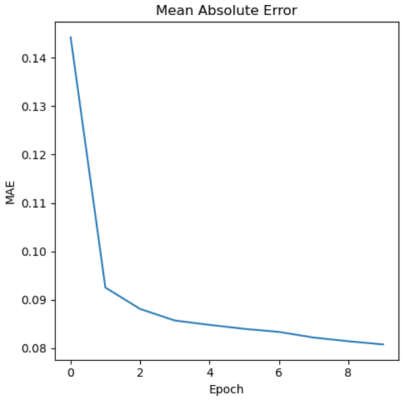


Figure 4. MAE Over 10 Epoch Training Run

Table 1
Model performance metrics. **Loss**: mean absolute error; **Marginal Accuracy**: accuracy within a 5% margin. Training metrics on last epoch.

Run/Metric	Loss	Marginal Accuracy
Training (8 epochs)	0.0687	0.9059
Testing (8 epochs)	0.1136	0.8701
Training (10 epochs)	0.0808	0.9004
Testing (10 epochs)	0.0869	0.8922

For our 8 Epoch training, we were able to utilize Pearson Correlation to assess the linearity of our outputs compared to the ground truth. Training yielded a final epoch average performance of 0.6924, whereas testing only yielded an average 0.0531, so it's apparent this instance of the model had overfit.

4 Challenges

The three main challenges we faced were finding suitable data, aggregrating the data from diverse datasets, and ensuring our data formatting and tensorflow translation of the DSTGCN fit nicely.

4.1 Data Search

Discussed below in section 5.2, we initially experienced difficulty finding suitable data for a previous desired task of predicting traffic accident. We spent many hours combing through datasets, querying apis, downloading CSVs and checking columns, only to find much insufficient information.

4.2 Data Aggregration and Graph Construction

Since we used four separate datasets, we required lots of work to combine our relevant features for the desired model input.

As mentioned in Section 2.2, the graph construction relied mainly on station ids, and route information. We decided to create the nodes using station ids from the Metropolitan Transportation Authority, 2020-2024 turnstile dataset, given the bulk of our data (temporal hourly ridership) would be coming from here, and the daily ridership data had matching ids. The Data.gov, 2023 route information unfortunately did not have matching ids. We implemented distance matching (using geodesic and coordinates from both datasets) and matched differing station ids from the two datasets accordingly; this allowed us to use the Data.gov, 2023 route information to generate edges, and subsequently an adequate adjacency matrix.

For windowing operations mentioned in section 2, we then needed to ensure that correct T=24 timeframes were gathering data from all source (graph hourly ridership, daily ridership of previous day, past 24 hours of weather.

4.3 TF Translation

Our tensorflow translation was the expected challenge: losing the low-level customizability of pytorch. This required new library usage (spektral) and some simplification in our implementation, e.g. single graph convolution as opposed to batched graphs, at the loss of vectorization (that would hopefully be [re-]implemented in future work).

5 Reflection

Given the compute required for training our model (1 hour per epoch), and our deadline squeeze, we only trained a few times and did not have the chance for successful hyperparamter tuning. However we saw positive results training on 5 epochs as follows:

Final Epoch Avarge Train loss (MSE) = 0.08,
Average Test loss (MSE) = 0.08,
Marginal accuracy (5%) = 90%.

Our target goal was initially 90% marginal accuracy and we just made it! This means for every 9/10 examples the model's prediction was with 5% of the true value. Given buffed data inputs and hyperparameter tuning, we believe our model could reach our stretch goal of 95% marginal accuracy. We feel great about these results; the positive results support our initial driving assumption that the graph structure of the NYC subway system would aid in predicting features about each station (pedestrian count).

Overall our model worked out the way we expected to, with the added bonus of few bugs in the train/test runs (we were uncertain about our translation of graph layers into tensorflow but looks good!).

5.1 Improvements

Our implementation of spatial convolution blocks in tensorflow took a single graph as input for its forward pass. We would like to adjust this implementation to perform batched graph convolution, thereby vectorizing operations and allowing greater use of GPU optimizations. This would allow for greater training and testing speeds.

This speed-up would also have promoted hypertuning. We would have liked to split off validation data and perform validation while training, hypertuning our many architecture parameters (learning rate, GCN depths, convolution filters...). We wanted to learn a yearlong prediction, but it may have also been worthwhile to split the data for more training (i.e. put some of 2024 in the training).

We would like to adjust our graph convolution implementation, perhaps making use of alternative GraphConv libraries to better exploit our detailed graph preprocessing. In our graph creation, we had stored edge values, including station distances and number of shared lines, that we believe would add relevant graph structure.

To augment our data, we might consider adding "events" to our external features; it is quite obvious that a big event, such as a Knicks game in MSG above Penn Station, will impact subway ridership. Another addition to our data could be connecting hourly data from previous weeks – e.g. when predicting Monday 8am ridership, we might want to consider the previous few Monday 8am counts. This makes sense given travellers' routines, such as morning commutes. It would also be beneficial to augment our static spatial data with more details about each station; some possible additions include station sizes (area, number of turnstiles, number of platforms...) and station construction status (in progress, recently updated...).

5.2 The Great Pivot

We started our project intending to follow Yu et al., 2021 in predicting traffic accidents in NYC. After hours and hours (and more

hours) scraping the available sources for NYC traffic data, however, we could not find suitable (or easily accessible) datasets to perform this task in our given timeframe (some datasets, for example, required weekslong request processes, others had sparse traffic readings). We had already been excited in employing graph concepts on some sort of traffic regression task, so we tried finding what public transportation was available. To our dismay, again NYC did not have what we wanted in this regard (not enough consistency in data, unreliable delay/schedule data). We tried a plethora of other metro areas, searching far and wide (Singapore, DC, Toronto...) to no avail. We did, however, find NYC subway turnstile counts – and here we are now.

5.3 Takeaways, Changes

As outlined in the pivot discussion, our biggest takeaway lies in the data steps involved in training a deep learning model. Our implementation of the actual architecture, showed us that the deep learning techniques we have grown through this course are now accessible to us and implementable in broad settings. The real brunt of the work, as suggested many times by our wonderful TAs, was in fact the data steps. We spent many hours trying to find suitable datasets, querying APIs, finding file downloads, and combing through columns to ensure continuous and correct data. Then, we poured many many more hours into preprocessing the data we had found into the fitting formats to be used in our architecture. Doing this project over again, we would have prioritized finding and confirming the correct datasets at an earlier stage. In order to do this, we would have to have in mind exactly what is required in our data (for example, to use datasets with differing station ids we needed a method to match stations from the two datasets). We would have mapped out the exact steps in preprocessing, and organized our tasks accordingly. We would then have executed this structural plan together, and added any steps where we realized additions were needed; this structured approach would make it easier to return to preprocessing if anything needed to be checked or changed.

Acknowledgements

This project received support during the CSCI 1470 course, instructed by Professor Eric Ewing, with TA support from Armaan Patankar.

References

Data.gov (2023). *New York City GTFS*. URL: https://catalog.data.gov/dataset/mta-general-transit-feed-specification-gtfs-static-data.

Metropolitan Transportation Authority (2020-2025). *MTA Daily Ridership Data*. URL: https://data.ny.gov/Transportation/MTA-Daily-Ridership-Data-2020-2025/vxuj-8kew/about_data?utm_source=chatgpt.com.

— (2020-2024). *MTA Subway Hourly Ridership*. URL: https://data.ny.gov/Transportation/MTA-Subway-Hourly-Ridership-2020-2024/wujg-7c2s/about_data.

National Weather Service (2023-2024). *NYC, Kennedy Airport Weather Observations*. URL: https://forecast.weather.gov/data/obhistory/KJFK.html.

Yu, Le et al. (2021). "Deep Spatio-Temporal Graph Convolutional Network for Traffic Accident Prediction". In: *Neurocomputing* 423, pp. 135–147. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2020.09.043. URL: https://doi.org/10.1016/j.neucom.2020.09.043.