Crowd Density Forecasting Using Graph and Recurrent Neural Networks

Lucas Childs
Statistics and Data Science
UC Santa Barbara
lucaschilds@ucsb.edu

Logan Detrick
Computer Science
UC Santa Barbara
logandetrick@ucsb.edu

Julia Novick
Computer Science
UC Santa Barbara
jnovick@ucsb.edu

Riona Pampati
Computer Science
UC Santa Barbara
riona@ucsb.edu

Eirini Schoinas
Computer Science
UC Santa Barbara
eirini@ucsb.edu

Abstract

Urban mobility forecasting plays a critical role in managing traffic, improving transportation efficiency, and anticipating crowd dynamics. This paper introduces a spatio-temporal graph-based deep learning model to predict crowd density, measured as the number of taxi drop offs in each zone of New York City. Utilizing the NYC Yellow Taxi trip records from 2017, we construct static graphs (both per hour and as a single co-visitation graph), embedding spatial and temporal attributes into the node features. Each node represents a taxi zone, and edges capture inter-zone traffic with trip counts and passenger volumes. We employ a Graph Neural Network (GNN) architecture coupled with a Recurrent Neural Network (RNN) architecture to model spatial dependencies, integrated with a temporal recurrent mechanism for hourly prediction. Our model forecasts the number of drop offs per zone for the subsequent hour, demonstrating robust performance against temporal fluctuations such as weekends, holidays, and rush hours. Our work builds on advances in spatio-temporal graph learning [1][2] and contributes a scalable framework for urban crowd forecasting.

1 Introduction

As urbanization intensifies, real-time understanding and forecasting of crowd dynamics have become essential for city planning, public safety, and transportation optimization. The increase of mobility data, especially from ride-hailing and taxi services, offers unprecedented insight into urban flow patterns. One of the primary challenges in leveraging such data lies in capturing both spatial correlations (i.e. how different regions influence each other) and temporal trends (i.e. how demand evolves over time).

Graph-based deep learning has emerged as a powerful framework for modeling such spatio-temporal data. Graph Neural Networks (GNNs), especially Graph Convolutional Networks (GCNs), have been widely adopted for their ability to represent non-Euclidean relationships among entities [3]. When combined with temporal modeling techniques such as Recurrent Neural Networks (RNNs) or sequence encoders, GNNs provide a natural structure for predicting the evolution of dynamic systems like traffic or crowd density [4].

Our project leverages this framework by constructing two approaches, both using graphs, each corresponding to one hour of 2017 Yellow Taxi dropoff data. Each node in the graph represents a zone defined by the NYC Taxi and Limousine Commission (TLC), and includes features like average

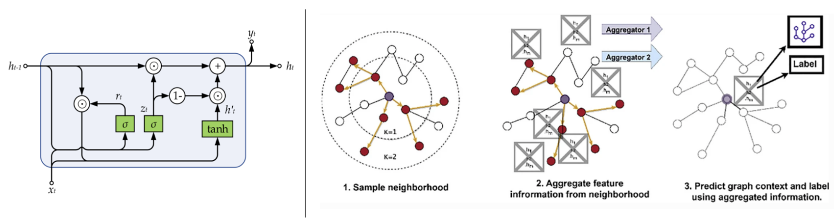

h_{t-1}
h_t
y_t
x_t
r_t
z_t
1-
h'_t
σ
σ
tanh
K=1
K=2
1. Sample neighborhood
Aggregator 1
Aggregator 2
2. Aggregate feature infrromation from neighborhood
Label
3. Predict graph context and label using aggregated information.

Figure 1: Left: GRU Architecture. Right: GraphSAGE Architecture. Images courtesy of [15]

fare, congestion surcharge, time-of-day encodings, and dropoff volume. Edges represent taxi trips between zones and are weighted by the number of trips and passengers. This representation enables our model to learn not only local zone behavior but also the influence of surrounding zones. While our first approach entails creating separate graphs for each hour using the GraphSage architecture and piping that through a Gated Recurrent Unit (GRU), our second approach involves creating a singular co-visitation graph using the GCN architecture and putting that through a GRU.

Notably, prior work such as ST-GCN [5] and DCRNN [6] has shown the effectiveness of spatio-temporal GNNs for traffic prediction tasks. Our approach is similar in spirit but tailored to the unique properties of human mobility patterns captured via taxi trips, where the focus is not speed or flow on roads, but rather dropoff density, which correlates with crowd presence in urban space. By addressing both spatial heterogeneity and temporal trends, our work provides a foundation for proactive crowd management and demand-responsive transportation systems.

2 Related Works

"Predicting Taxi Pickups in New York City" by Josh Grinberg (2014) used several methods, notably linear regression and decision tree regression to predict the amount of taxi pickups at a given time and location. This work relied on 2013 taxi trip data but did not capture more complex spatial or temporal relationships, given the limitations of its models.

"New York City Taxi Trip Duration Prediction Using Machine Learning" by Kapil Saini (2024) predicted the duration of taxi trips using data from New York, augmented with weather data and traffic information. Similar to Grinberg's work, this paper used regression models linear, ridge, and lasso regression models, however it also uses the K-Nearest-Neighbors algorithm to group data points before using the regression model.

Jinbao Zhao et. al. (2022) used a graph convolutional network model in order to predict the urban taxi travel demand using datasets from the New York City Taxi and Limousine Commission. This paper is similar to our own work, but differs in key areas. The paper dynamically constructs a graph using sampled traffic data, while our paper creates the graphs statically using the taxi zones as vertices and trips as edges. While both papers use a graph neural network, Jinbao Zhao et. al. then uses a convolutional neural network, while our paper uses a recurrent neural network.

Based on the current landscape, we noticed that there was a lack of temporal predictions for taxi data. In this manner, previous papers did not predict the effects of rush hours and holidays on trip density. Our paper addresses the need for a temporal prediction method, while also using graph neural networks to capture the spatial features of our dataset. While several papers used a similar dataset to our own, we specifically processed our data for temporal tasks, constructing graphs by dividing trips by hour and taxi zone.

3 Methodology

In this paper, the goal of taxi dropoff demand prediction, or crowd density forecasting, is to predict the taxi drop-off count in NYC taxi zones (referenced by the NYC Taxi and Limousine Commision (TLC)) in one hour time intervals based on various historical taxi trip data features. These features include pickup and drop-off counts, taxi zone IDs, datetimes, passenger counts, etc. We use two different spatiotemporal techniques to forecast.

3.1 GCN+GRU Methodology

We define $G = (V, E)$ to be a graph representing the NYC taxi zones. Each taxi zone is a node, or vertex, V, and an edge is formed between two zones if there has been at least one taxi trip from one zone to another. Let T be the set of all taxi-trip records in the training data. $Z = \{z_1, ..., z_N\}$ is the set of the top 50 most active taxi zones, based on the zones with the most drop-offs. For each trip $t \in T$, let $p_t \in Z$, $d_t \in Z$ denote its pickup and drop-off zone ID respectively. We group the remaining trips in T by (p_t, d_t) to count how many trips went from zone i to zone j. Denote this integer by w_{ij}. Thus, the directed weighted graph G has $V = Z$ and $E = \{(z_i, z_j) \in Z \times Z \mid w_{ij} > 0\}$, with an edge-weight function $w : E \to \mathbb{Z}^+$ given by $w(z_i, z_j) = w_{ij}$. Instead of storing this graph structure as a dense adjacency matrix $A \in \mathbb{R}^{N\times N},\ A_{ij} = w_{ij}$ (number of trips from zone i to j), we use PyTorch Geometric's sparse format. All nonzero entries (i, j) with $w_{ij} > 0$ are stored in a tensor of two rows, where each column $[u, v]^T$ represents a directed edge from u to v. So instead of storing a $N \times N$ matrix, we store only the M edges that exist between nodes.

Each node in the graph, z_i, a taxi zone, has a standardized feature vector including drop-off count, average fare amount, average passenger count, average congestion surcharge, and one-hot encoded day types, for each hourly timestamp. At each hour, t, the node feature matrix is $X^{(t)} \in \mathbb{R}^{N\times F}$, where $N = |Z|$, the number of nodes (the top 50 taxi dropoff zones) and $F = 4 + D_{day}$ is the number of features per zone, including drop-off count, average fare, passenger count, congestion surcharge, and one-hot encoded day type (of dimension 7).

Overview This section will describe how the graph convolution network (GCN) to gated recurrent unit (GRU) framework works for crowd density prediction in terms of taxi drop-off demand. Early spatiotemporal forecasting methods treated space as a grid in the Euclidean space, allowing convolution neural networks (CNNs) to learn the spatial dependencies of city data divided into regions. However CNNs are not designed for non-Euclidean spaces, like the irregular spatial graph formed by taxi zones connected by frequent travel between them. GCNs are able to perform graph convolution directly on data structured in the format of a graph, extracting useful spatial patterns from non-Euclidean spaces [10] [11]. Thus, research shows a shift towards graph based spatiotemporal modeling for higher accuracy in forecasting taxi demand or traffic flow.

The GCN-GRU we implemented is broken down into two components, the GCN and GRU. We input historical taxi data from the year 2017, and the GCN encodes spatial context in terms of taxi zones and their neighbors. Then, the sequence of hourly spatially encoded data is fed to the GRU which learns temporal dependencies. The traditional RNN architecture is designed for time series data but runs the risk of vanishing and exploding gradients during backpropagation due to the re-use of parameters across time steps. GCNs include gating mechanisms that regulate the flow of information over time, helping mitigate the vanishing gradient issues prevalent in sequential learning. Additionally, long short-term memory (LSTM) networks work similarly to GRUs, but require increased computational power due to their more complex architectures which use more parameters with three gate networks rather than the GRU's two [12].

Model Architecture Given the node feature matrix, $X^{(t)} \in \mathbb{R}^{N\times F}$, $X^{(t)}$ is passed through a single layer GCN. The resulting node embeddings are:

$$H = \mathrm{ReLU}\big(\hat{A}X^{(t)}W_{\mathrm{gcn}}\big) \in \mathbb{R}^{N\times H_{\mathrm{gcn}}}$$

where $\hat{A} = \tilde{D}^{-\frac{1}{2}}(A + I)\,\tilde{D}^{-\frac{1}{2}} \in \mathbb{R}^{N\times N}$ represents the symmetric normalized adjacency matrix, $\tilde{D}$ is the degree matrix $\tilde{D} = \sum_j (A + I)_{ij}$, $W_{gcn} \in \mathbb{R}^{F\times H_{gcn}}$ is the trainable GCN weight matrix, H_{gcn} is the GCNConv layer output dimension of 32, and ReLU() is the rectified linear unit activation function to introduce non-linearity. This operation is handled by GCNConv from PyTorch Geometric. Thus the GCN model spatially learns the taxi trip features. In the temporal learning segment, the node embeddings H are reshaped to H' and fed as input to the GRU. $H' \in \mathbb{R}^{1\times N\times H_{gcn}}$ is the inputted to the GRU, reshaping H and treating spatial features as a single timestep sequence rather than a whole temporal sequence. Then, let $h_{t-1} \in \mathbb{R}^{H_{rnn}}$ be the GRU's previous hidden state where H_{rnn} is the hidden layer dimension of 64. Letting x_t be the input at time step t, the t-th node's GCN embedding, the GRU calculates:

$$z_t = \sigma\big(W_z x_t + U_z\, h_{t-1} + b_z\big), \qquad r_t = \sigma\big(W_r x_t + U_r\, h_{t-1} + b_r\big), \tag{1}$$

$$\tilde{h}_t = \tanh\big(W_h x_t + U_h\,\big(r_t \odot h_{t-1}\big) + b_h\big), \qquad h_t = z_t \odot h_{t-1} + \big(1 - z_t\big) \odot \tilde{h}_t. \tag{2}$$

where $W_z, W_r, W_h \in \mathbb{R}^{H_{gcn} \times H_{rnn}}$ are input and gate weights, $U_z, U_r, U_h \in \mathbb{R}^{H_{rnn} \times H_{rnn}}$ are hidden and gate weights, $b_z, b_r, b_h \in \mathbb{R}^{H_{rnn}}$ are bias vectors, and $\tilde{h} \in \mathbb{R}^{H_{rnn}}$ is the candidate hidden state. $z_t, r_t \in \mathbb{R}^{H_{rnn}}$ are the update and reset gates of the GRU respectively, σ represents the sigmoid function, and $\odot$ is element-wise multiplication. Then, the GRU learns the temporal relations of the taxi trip features outputting: $\hat{y}_t = W_{out} h_t + b_{out} \in \mathbb{R}^F$, for each $t \in Z$.

3.2 GraphSAGE Method

In this method, we create multiple graphs based on their hourly taxi trip feature information where each distinct taxi zone is represented by a node on the graph. Each taxi zone feature vector includes the same features as the previous vectors plus a calendar vector that includes hour, day of week, and month times sinusoidally represented to capture the cyclical nature of times, as well as is_holiday. The graph remains directed, where edges represent a trip between one zone to another within the hour time interval. Each edge carries total trip count and passenger count. Each graph is then stored in a PyG data object. The graph is then inputted to a two layer GraphSAGE model, known for its aggregation function and inductive framework. GraphSAGE learns how to aggregate neighborhood information so that it can create embeddings that generalize to unseen nodes [13]. The output is then run through a 2-layer GRU to learn temporal trends.

Overview Although GCNs improve on CNN structure, enabling them to spatially learn graph-based features, GCNs aggregate node information from all nodes' neighbors, which created noise when we used every taxi zone in our dataset. Thus, we considered only the top 50 most active zones for the graph created for the GCN. GraphSAGE, however, aggregates features by sampling a fixed number of neighbors for each node. This method updates nodes' representations with decreased computational cost, using a mean, for example, and can yield higher performance over the fixed convolution used in GCNs. Thus, we used graphs with all taxi zones for GraphSAGE. GraphSAGE also enables scalability so that in the future, if new nodes appeared in the graphs, GraphSAGE would be able to generalize to these unseen nodes [13].

Model Architecture Input to the GraphSAGE takes each node's 11 dimensional feature vector $\mathbf{x}_v \in \mathbb{R}^{11}$ and aggregates all of v's neighbors' feature embeddings using mean-pool. Then v's feature vector is concatenated with the aggregated vector of the neighbor's features to form a 22 dimensional vector. Next, this fist hidden layer applies a linear map $W^{(1)} \in \mathbb{R}^{128\times 22}$ plus bias $b^{(1)} \in \mathbb{R}^{128}$, followed by ReLU. Outputted by this layer is a 128-dim embedding $h_v^{(1)} \in \mathbb{R}^{128}$ for each node. Hidden layer 2 takes $h_v^{(1)} \in \mathbb{R}^{128}$ for each node v and aggregates again using mean-pools for all neighbors' embeddings. v's embedding is then concatenated with the neighbor mean aggregated features to form a 256-dimensional vector. A linear map $W^{(2)} \in \mathbb{R}^{128\times 256}$ plus bias $b^{(2)} \in \mathbb{R}^{128}$ is applied, followed by ReLU. Outputted by this layer is a 128-dim embedding $h_v^{(2)} \in \mathbb{R}^{128}$ for each node. Explicitly, aggregation is computed in each layer like:

$$h_{\mathcal{N}(i)} = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} h_j^{(0)} \;\in\; \mathbb{R}^{11}.$$

The output from GraphSAGE, graph_emb, is a 128-dimensional embedding per graph for B total hourly graphs each with 265 nodes. This looks like:

$$z_g = \frac{1}{265} \sum_{i \in \{0,\ldots,B\cdot 265-1\}} h_i^{(2)}.$$

Embeddings are then inputted to the two layer GRU which uses hidden layer dimensions of 128 and outputs taxi drop-off count predictions for the next hour of size $B \times 265$.

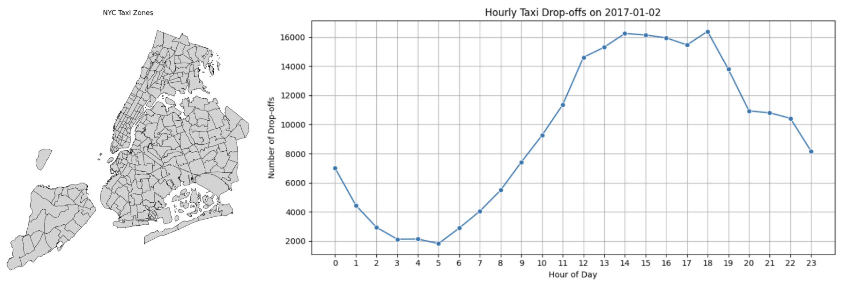


Figure 2: Left: New York City Taxi Zones. Right: Hourly taxi drop-offs for the whole of NYC during a given day

4 Forecasting

This section will cover each forecasting method and output predictions. T-GCN employs spatiotemporal methods to forecast the next hour's taxi drop-off count per zone. Additionally, we predict the dropoff count of the next 24 hours, $\hat{y}_{t+24}$, simulate day types to forecast dropoff counts for weekdays or weekends in single zones and multiple–to display zones' behavior during weekdays/weekends–and forecast drop-off count for all zones 3-steps ahead by recursively making a one hour ahead prediction. Thus, we are able to see zones behave under theoretical weekdays or weekends and compare these zone-to-zone. Furthermore, T-GraphSAGE also predicts drop-off count per zone for the next hour, but doesn't extend to simulating theoretical days or multi-step horizons. Both models were trained on TLC taxi data from all months of 2017, and are evaluated using distinct testing methods highlighted in the results section.

5 Dataset Description

The dataset used for traffic forecasting contains taxi data from all boroughs of New York City provided by the NYC Taxi and Limousine Commision (TLC) [14]. The dataset lists three types of taxis: yellow taxis, green taxis, and for-hire vehicles. We chose to focus on yellow taxi trip records as they are not restricted to any particular zones in NYC, like green taxis are. Each taxi or observation records 19 different data features like dropoff_datetime, DOLocationID, passenger_count, and trip_distance. For our purposes, we chose to use pickup and dropoff_datetime, PU and DOLocationID, passenger_count, fare_amount, and congestion_surcharge.

5.1 Preprocessing

During preprocessing, we found some irregularities in the dataset in which timestamps were in monthly datasets that they should not be in. We discarded these irregularities making sure timestamps were consistently within their defined bounds. Additionally, for node feature vectors inputted to GraphSAGE, we created sinusoidal time data to represent the cyclical nature of hourly, day-to-day, and week formats.

6 Results

Our team explored two distinct modeling approaches, so we employed distinct evaluation methods tailored to each framework. The GraphSAGE-GRU method, using a graph sequence approach with a GraphSAGE encoder followed by a GRU trained on graph embeddings, was evaluated on full-year data using raw and relative metrics (RMSE, MAE, R²). The GCN-GRU method, using a feature-level GNN+RNN approach, encoding per-zone spatial information and inputting it into a GRU to learn temporal dynamics, was evaluated through validation MSE, zone-wise multi-step forecasts, and simulations of daily patterns.

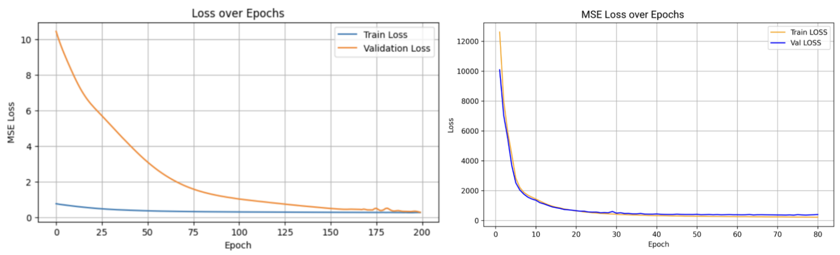


Figure 3: Left: GNN+RNN Loss. The model converges after 150 epochs, with validation MSE dropping below 0.2. Final Train MSE: 0.18, Final Validation MSE: 0.21. Right: GraphSAGE Loss. Progression of train and validation loss over 80 training epochs

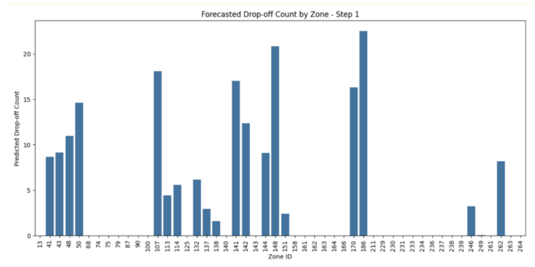


Figure 4: Predicted Taxi Drop-offs by Zone

6.1 GNN+RNN Results

We evaluate our GNN+RNN model's predictions of crowd density in NYC taxi zones based on historical drop-off data.

Quantitative Results We trained the model for 200 epochs using MSE loss. As shown in Figure 3, the validation loss consistently decreases and closely approaches the training loss, which suggests strong generalization and stable learning without overfitting. By epoch 190, the validation loss reaches 0.33, while training loss is 0.26.

Zone-Level Forecasting To measure the model's ability to correctly handle differences in crowd density across different locations, we visualize predictions for different zones in multi-step forecasts (1, 2, and 3 hours ahead). Figure 4 exhibit that the model accurately identifies high-traffic zones such as 107, 141, 148, and 186, where they consistently have higher predicted drop-off activity. These results confirm that the model preserves learned spatial patterns in demand across time. Zone 186, in particular, remains the highest across all steps, providing insight into its status as a consistently dense location. This is Manhattan Penn Station/Madison Sq West, a major transportation and event hub. This insight reflects the model's ability to learn and preserve zone-specific patterns in urban mobility.

Hourly Dynamics by Zone and Day Type We simulate 24-hour forecasts for individual zones under different day types. Figure 5 show consistent intraday trends, with predicted drop-offs gradually declining from midnight to midnight. Drop-off counts on Saturdays tend to be slightly lower overall than Mondays, especially during day-time and in mid-traffic zones like 107. High-traffic zones like 148 remain stable across day types. These results exhibit that the model generalizes well across both spatial and temporal dimensions. By adjusting the one-hot encoded weekday input, the model can simulate crowd density on a specific day, which supports applications like city planning or transit scheduling.

6.2 GraphSAGE-GRU Results

The model's performance on 2017 training data shows strong fit, with a high R^2 value of 0.9846 indicating that it explains nearly all the variance in dropoff counts. The training MAE of 5.57 and

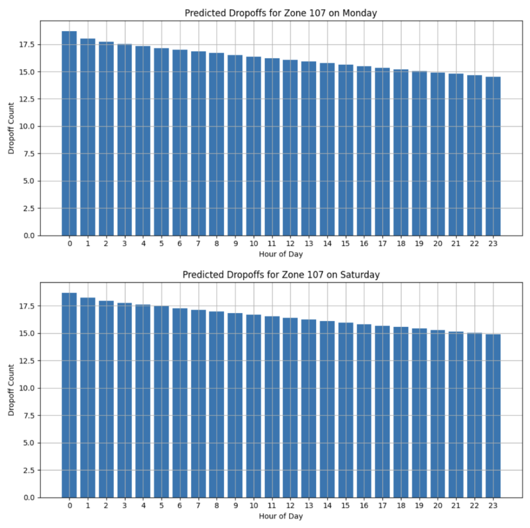


Figure 5: Comparing zones 107 and 148 hour-by-hour. The model captures both level differences and slope trends across zones and day types

Table 1: Average prediction metrics on 2017 (Train) and 2018 (Test) data. Relative metrics are percentages.

Metric	Train (2017)	Test (2018)
Loss (MSE)	220.0350	649.6746
MAE	5.5723	9.4267
RMSE	14.8336	25.4887
R^2	0.9846	0.9449
Avg. Target Value	48.9060	44.3331
Relative MAE (%)	11.39	21.26
Relative MSE (%)	9.20	29.60
Relative RMSE (%)	30.33	57.49

RMSE of 14.83 are relatively low compared to the average dropoff count of 48.91, reflected in fairly low relative errors . Also note, Root mean squared error (RMSE) and mean squared error (MSE) penalize large mistakes that come from zones with more trips more than mean absolute error (MAE).

When applied to 2018 test data, performance with all metrics declines. However, R^2 drops only to 0.9449, which still indicates a lot. This suggests the model somewhat overfits to 2017 patterns and generalizes less effectively to unseen 2018 data, possibly due to shifts in dropoff behavior or other external factors affecting demand.

Evaluation Metrics Let y_i be the true value, $\hat{y}_i$ the predicted value, and $\bar{y}$ the mean of the true values over n samples.

$$\text{Loss (MSE)} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad \text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3}$$

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \qquad R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4}$$

$$\text{Relative MAE (\%)} = 100 \times \frac{\text{MAE}}{\bar{y}} \qquad \text{Relative RMSE (\%)} = 100 \times \frac{\text{RMSE}}{\bar{y}} \tag{5}$$

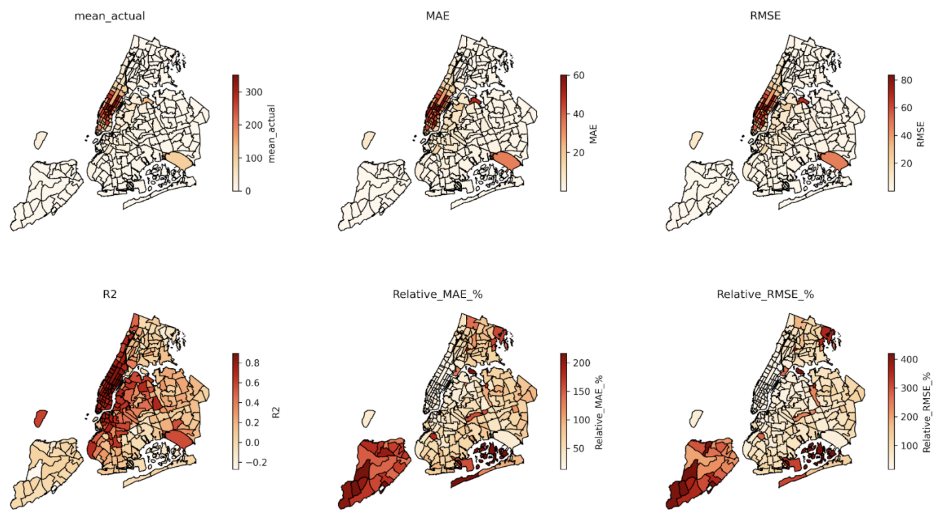


Figure 6: Each map displays a performance metric averaged over time per zone. Top row: mean actual trip counts, mean absolute error (MAE), and root mean squared error (RMSE). Bottom row: coefficient of determination (R^2), relative MAE (%), and relative RMSE (%). Color intensity indicates higher values. Metrics have been clipped at the 98th percentile to highlight spatial trends. High-traffic zones in the center of the city tend to show higher absolute errors and better R^2, while low-traffic zones exhibit disproportionately high relative error.

Loss over Training These results are from the initial training phase, where the first 80% of 2017 data was used for training and the final 20% was reserved for validation. Metrics shown reflect averages across all predictions during this phase. Afterward, the model was briefly fine-tuned using the full 2017 dataset, allowing it to learn from all parts of the year, before making predictions on the 2018 test set.

The loss used was a raw MSE. The loss improves rapidly at first, then gradually begin to plateau. Initially, performance on the validation set slightly exceeds that of the training set, but as training progresses, the training metrics continue to improve and eventually surpass the validation metrics. Data was not standardized, which is what led to such high raw loss values. In the last epoch, training loss was 229.92 and the validation loss was 410.03.

Comparing Metrics to Zones (on Test Set/2018) Using the final model, predictions were made on the test set. MAE and RMSE are in the same units as the target variable, which makes them much easier to interpret directly than MSE. MAE is more robust to outliers than RMSE. When plotted against actual target values, MAE is generally lower, but both metrics show a similar trend.

Relative Error Relative error show the inverse relationship between target and error. Even errors of only a few trips can inflate relative error tremendously in zones with very small single-digit target values.

6.3 Combined Results

Both models have strong predictive capabilities but are better in different aspects, judging from the results. The GraphSAGE-GRU (graph sequence approach) excels at learning from graph patterns and long-term generalization. In high-volume zones it performs well in predicting average demand levels accurately. In contrast, the GNN-RNN method (feature-level spatiotemporal GNN+RNN) excels in capturing fine-grained temporal and spatial variations and is effective in simulating intraday and zone-specific dynamics. This approach is well-suited for day-specific forecasting and short-term demand forecasting and visualization.

7 Conclusion

Accurate crowd density prediction has meaningful societal applications, such as improving public safety, optimizing city planning, and managing emergency responses. However, there are also potential risks, such as bias if models fail to generalize beyond data from affluent, urban areas like NYC.

We employed two methods to predict crowd density. Our GraphSAGE-GRU method succeeded at learning long-term patterns, while our GNN-RNN method worked better at intraday patterns. We did not have access to very much compute power, so we were only able to train on a subset of the dataset. This led to underfitting when we applied our model to the testing dataset, it struggled to capture variance on an intraday level. The dataset that we used lacked geographic diversity, only consisting of data from New York City, potentially creating city-specific patterns that may not generalize to other cities or countries.

In the future, we aim to combine the two methods to capture not just fine-grained details, but also large-scale features in one model. We assumed a static graph structure with dynamic weights, which may not reflect short-term events like road closures, events, or construction. It would be interesting to dynamically generate the graph used by our model based on these factors.

8 Code Availability

Our code is available through GitHub at: `https://github.com/JuliaNvck/crowd_density_forecasting`

Code used to visualize the data is available through GitHub at: `https://github.com/lucasc212/crowd-density-forecasting-data-visualizations`

9 Collaboration Statement

Lucas researched spatiotemporal forecasting methods, comparing GNN+RNN models and their mathematical formulations, generated descriptive dataset visualizations, and implemented a GAT-based model for comparison—ultimately excluded from the final paper. Besides literature review and conceptual design, Julia designed and implemented the GNN+RNN model with the single co-visitation graph, including data processing, tuning, and analysis of results, including simulations and evaluations of the model's performance. Logan researched and wrote the related works as well as the conclusion. Logan also formatted and compiled the entire paper into LaTeX. Eirini found the dataset and designed and implemented the GNN+RNN model, which utilized hourly graphs as input. This included data processing, pipeline, and combined model design, as well as visualizing, analyzing, and writing up results. Riona assisted with the GNN+RNN model with individual graphs per hour, researching different architectures and which attributes would be key in the spatio-temporal forecasting, as well as looking into related works and writing up the introduction and abstract sections.

References

[1] Wu, Zonghan, et al. "A comprehensive survey on graph neural networks." IEEE Transactions on Neural Networks and Learning Systems (2020). `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9046288`

[2] Gao, Hongyang, and Shuiwang Ji. "Graph U-Nets." ICML (2019). `https://proceedings.mlr.press/v97/gao19a/gao19a.pdf`

[3] Kipf, Thomas N., and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks." ICLR (2017). `https://arxiv.org/pdf/1609.02907`

[4] Seo, Youngjune, et al. "Structured Sequence Modeling with Graph Convolutional Recurrent Networks." ICONIP (2018). `https://arxiv.org/pdf/1612.07659`

[5] Yu, Bing, Haoteng Yin, and Zhanxing Zhu. "Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting." IJCAI (2018). `https://www.ijcai.org/proceedings/2018/0505.pdf`

[6] Li, Yaguang, et al. "Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting." ICLR (2018). `https://arxiv.org/pdf/1707.01926`

[7] Grinberg, Josh, Arzav Jain, and Vivek Choksi. Predicting Taxi Pickups in New York City. Final Paper for CS221, Autumn 2014, `https://www.vivekchoksi.com/papers/taxi_pickups.pdf`

[8] Saini, Kapil. "New York City Taxi Trip Duration Prediction Using Machine Learning." International Journal of Creative Research Thoughts (IJCRT), vol. 12, no. 7, July 2024, p. g663. `https://ijcrt.org/papers/IJCRT2407755.pdf`.

[9] Zhao, Jinbao, et al. "Prediction of Urban Taxi Travel Demand by Using Hybrid Dynamic Graph Convolutional Network Model." Sensors, vol. 22, no. 16, 2022, p. 5982. `https://doi.org/10.3390/s22165982`.

[10] LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (2002): 2278-2324. `https://ieeexplore.ieee.org/document/726791`

[11] Yu, Bing, Haoteng Yin, and Zhanxing Zhu. "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting." arXiv preprint arXiv:1709.04875 (2017). `https://arxiv.org/abs/1709.04875`

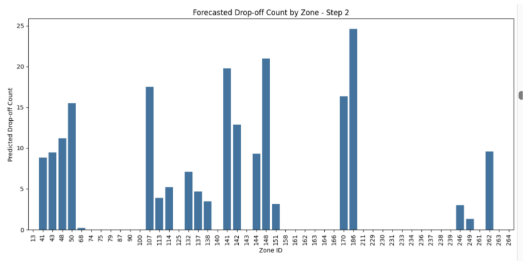
[12] Dey, Rahul, and Fathi M. Salem. "Gate-variants of gated recurrent unit (GRU) neural networks." 2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS). IEEE, 2017. `https://ieeexplore.ieee.org/abstract/document/8053243`

[13] Hamilton, Will, Zhitao Ying, and Jure Leskovec. "Inductive representation learning on large graphs." Advances in neural information processing systems 30 (2017). **?**

[14] "TLC Trip Record Data." TLC Trip Record Data - TLC, www.nyc.gov/site/tlc/about/tlc-trip-record-data.page. Accessed 9 June 2025. `https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page`

[15] Khemani, Bharti, et al. "A Review of Graph Neural Networks: Concepts, Architectures, Techniques, Challenges, Datasets, Applications, and Future Directions." Journal of Big Data, vol. 11, no. 1, 2024, article 18, `https://doi.org/10.1186/s40537-023-00876-4`.

10 Appendix



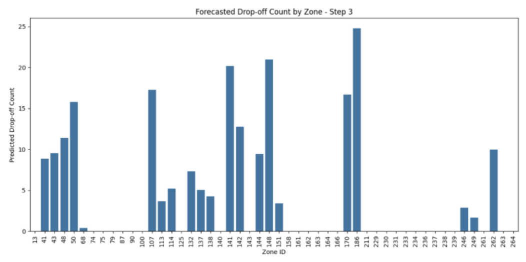


Figure 7: Drop-off count by Zone, 2 and 3 hour prediction respectively