Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Master's Degree in Artificial Intelligence

Master Thesis

Application of Graph Neural Networks for Urban Mobility Demand Forecasting

Author: Simon Markmann
Tutor: Damiano Zanardini

Madrid, July - 2023

Master Thesis
Master's Degree in Artificial Intelligence

Title: Application of Graph Neural Networks for Urban Mobility Demand Forecasting

July - 2023

Author: Simon Markmann
Tutor: Damiano Zanardini
Department of Artificial Intelligence
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

El continuo crecimiento de la urbanización en los últimos años ha incrementado la necesidad de herramientas avanzadas para la gestión de la movilidad urbana. Esta tesis aborda la tarea de predecir la demanda de movilidad urbana empleando redes neuronales gráficas (GNN) para utilizar la información estructural incorporada en los datos de viajes en taxi y estaciones de alquiler de bicicletas, y mejorándola con fuentes de datos externas como datos meteorológicos o series temporales. Se presentan sistemáticamente los trabajos relacionados, la metodología detallada, la discusión de los resultados y las conclusiones, junto con una perspectiva.

El problema de la predicción de la demanda de movilidad se formula cuidadosamente en esta tesis, integrando datos espaciotemporales y características externas de forma innovadora. Se introduce un modelo novedoso, el ST-GAT* (Redes de Atención Gráfica Espacio-Temporal con integración de datos externos), que aumenta el modelo ST-GAT original con fuentes de datos adicionales. La metodología incluye varios pasos, desde la recogida y el procesamiento de datos, pasando por la extracción de características y la representación gráfica, hasta la selección del modelo, el entrenamiento, la evaluación y la implementación.

Se ofrece una visión general de las GNN, que son la base del modelo ST-GAT*, junto con un examen en profundidad de los detalles del modelo y las motivaciones para la selección del mismo. Este trabajo también aborda las consideraciones éticas y los problemas de privacidad de los datos inherentes al uso de datos de movilidad pública.

Los resultados de esta tesis ponen de relieve el potencial de las GNN, y en concreto del modelo ST-GAT* propuesto, para predecir con precisión la demanda de movilidad urbana. Este enfoque innovador podría constituir una nueva vía para ampliar las estrategias de gestión de la movilidad. Se discuten los resultados, las implicaciones y las posibles direcciones futuras. Las perspectivas subrayan la amplia aplicabilidad de este enfoque, con posibles extensiones a otros retos urbanos y posibles desarrollos en la investigación de las GNN.

Abstract

The continued growth in urbanisation in recent years has increased the need for advanced tools for urban mobility management. This thesis addresses the task of predicting urban mobility demand by employing Graph Neural Networks (GNNs) to utilise the structural information embedded in taxi trips and rental bicycle stations data, and further enhancing it with external data sources such as weather or time-series data. The related work, detailed methodology, discussion of the results, and conclusions, along with an outlook, are all systematically presented.

The problem of mobility demand prediction is carefully formulated in this thesis, integrating spatio-temporal data and external features in an innovative way. A novel model, the ST-GAT* (Spatio-Temporal Graph Attention Networks with external data integration), which augments the original ST-GAT model with additional data sources, is introduced. The methodology includes several steps, from data collection and processing, through feature extraction and graph representation, to model selection, training, evaluation, and implementation.

An overview of GNNs, which are the basis of the ST-GAT* model, is provided, together with an in-depth examination of the details of the model and the motivations for model selection. This work also touches the ethical considerations and data privacy concerns inherent in the use of public mobility data.

The findings of this thesis highlight the potential of GNNs, and specifically the proposed ST-GAT* model, to predict urban mobility demand accurately. This innovative approach could be a new way for extending mobility management strategies. The findings, implications, and potential future directions are discussed. The outlook underscores the wide applicability of this approach, with potential extensions to other urban challenges and possible developments in GNN research.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Research Objectives . . . 2
- 1.3 Structure . . . 3

2 Related Work **5**
- 2.1 Graph Neural Networks . . . 5
- 2.2 Urban Demand Mobility Prediction . . . 6
- 2.3 GNNs for Spatio-Temporal Prediction . . . 8

3 Methodology **9**
- 3.1 Problem Formulation . . . 10
- 3.2 Data Collection . . . 11
- 3.3 Data Processing and Feature Extraction . . . 14
- 3.4 Graph Representation . . . 16
- 3.5 Overview of Graph Neural Networks (GNNs) . . . 18
- 3.6 Model Selection and Design . . . 20
- 3.7 Training the Model . . . 23
- 3.8 Model Evaluation and Validation . . . 25
- 3.9 Implementation Details . . . 26
- 3.10 Ethics and Data Privacy . . . 27

4 Results and Discussion **29**
- 4.1 Performance of the ST-GAT* Model . . . 29
- 4.2 Examination of Predictions . . . 35
- 4.3 Input Feature Comparison . . . 40
- 4.4 Graph Structure Comparison . . . 41
- 4.5 Baseline Model Comparison . . . 42
- 4.6 Comparison with Previous Research . . . 46
- 4.7 Robustness Analysis . . . 47
- 4.8 Sensitivity Analysis . . . 49
- 4.9 Discussion . . . 50

5 Conclusion and Outlook **53**

Bibliography **58**

Appendix **59**
- .1 List of Test Runs . . . 59

.2 Additional Baseline Comparison Evaluations 63

Chapter 1

Introduction

As the urban environments become more complex and densely populated, understanding and predicting urban mobility demand is increasingly crucial. The ability to predict how, where, and when people move through a city can help city planners, policymakers, and transport service providers, trying to improve urban infrastructure, manage traffic congestion, and reduce environmental impact. Traditionally, such predictions were made based on static, tabular datasets which often lacked the dynamic context of city life. This thesis introduces a novel approach to this challenge by leveraging Graph Neural Networks (GNNs) and additional data sources to analyse and predict urban mobility demand using spatio-temporal data.

This method expands on the traditional statistical or deep learning methods by incorporating the complex structure of relationships between different urban areas. GNNs, a powerful tool in artificial intelligence, enable the architecture proposed in this thesis to process and learn from this network data effectively. They interpret cities as interconnected graphs, where each node represents a different area and each edge signifies a connection between those areas, such as a road or a public transportation route, or even non-physical connections.

By applying GNNs to spatio-temporal data and afterwards combining it with meteorological and time-series data, the aim is to capture the dynamic nature of urban mobility. These insights can then be used to build more accurate, real-time prediction models, potentially improving the urban mobility situation.

This thesis presents the complete process of constructing such a predictive model: from problem formulation, data collection and processing, to feature extraction, model selection and design, training and validation of the model, and implementation details.

Through this research, the aim is to explore the potential of GNNs in urban mobility demand prediction, demonstrating their practical applications and guiding future studies in this field of artificial intelligence and urban planning.

1.1 Motivation

In every city around the world, we often encounter a common situation: roads crowded with cars, many of which have only one or two passengers inside [3]. This observation exposes a significant inefficiency within the current transport system: a huge number of cars that transport a minimal amount of people. This reality of our everyday urban life leads to considerations of the long-term sustainability of the current methods of transportation. It is evident that if we wish to create greener, more liveable cities, there is an urging need to rethink the approach to urban mobility.

One solution that has been considered is the creation of car-free cities [19]. This includes the idea of urban centres alive with pedestrians, cyclists, and efficient public transportation, but without private cars. Not only would this dramatically reduce emissions, but it would also open up space, creating opportunities for social interaction and community building. However, moving towards car-free cities is not as simple as it sounds. It involves changing the traditional transportation habits of millions of people, a task that is very complex.

This significant change in the urban lifestyle can not happen overnight. Cities are evolving entities, and with them, the methods and modes of transportation evolve as well. As individuals undergo these transformations, they simultaneously encounter new sets of challenges and opportunities. Predicting and understanding these changes is crucial in avoiding potential problems down the line.

For example, if we know how people move around the city and when they are most likely to do so, one can effectively plan the urban landscapes and transportation systems. By efficiently managing transportation demand, traffic congestion could be reduced, making cities more accessible. More importantly, it supports a more sustainable future, using resources more efficiently, reducing pollution, and improving the overall quality of life for city residents.

Understanding and predicting urban mobility demand, therefore, carries huge value. It provides the knowledge to proactively shape the urban landscapes in a way that is not only efficient but also sustainable. This thesis is driven by this motivation. The aim is to use the power of Graph Neural Networks to understand and predict urban mobility demand, thereby contributing to creating smarter, greener, and more sustainable cities. This thesis hopefully inspires further research and practical initiatives, pushing towards a future where the means of urban mobility are less dominated by congested streets.

1.2 Research Objectives

The primary objective of this research is to apply Graph Neural Networks (GNNs) in a novel context: predicting urban mobility demand using heterogeneous spatio-temporal data.

To fully understand the objectives of this thesis, several key research goals are outlined in the following paragraphs:

1. Firstly, the aim is to construct a predictive model using GNNs. Spatio-temporal data from a variety of urban locations will be collected, with the ambition to construct a model that can effectively process and interpret this information. By applying GNNs, the goal is to understand the complex nature of urban spaces, which traditional models may struggle to capture. Through this, the aim is to predict the demand for urban mobility with greater accuracy.

2. Secondly, the influence of additional datasets will be explored, specifically meteorological and time-series data. Weather is a key variable that influences the movement patterns of people, while time-series data can provide insights into temporal changes in urban mobility. These data types will be incorporated into the GNN model, and it will be assessed on how they impact the results.

3. Thirdly, the behaviour of different graph structures within the GNN model will be examined. Graph structures represent the connections between different locations in the urban environment, and the manner of these connections might influence how well mobility patterns can be predicted. By testing various structures, it is intended to identify which types can most effectively model the urban environment and its mobility needs.

4. Ultimately, the goal of this research is not just to build a predictive model, but also to generate a broader understanding of how advanced machine learning techniques like GNNs can be effectively employed in the context of urban planning and mobility prediction.

1.3 Structure

The thesis is structured as follows:

- Introduction: As the introduction to this thesis, this chapter presents the motivation for the research, outlines the research objectives, and explains the structure of this research.
- Related Work: This chapter discusses relevant existing literature and research in the field. First, a brief overview to the history of GNNs is given and traditional methods for urban mobility prediction are laid out. Finally, the application of GNNs on this task is presented.
- Methodology: This chapter is the core work of this thesis. In several sections, the complete process of this research is presented, from problem formulation and data collection to model design and evaluation. It also discusses the ethical considerations and data privacy aspects.
- Results and Discussion: Here, the outcomes of the proposed model are presented, compared with existing approaches, and its implications and potential applications are discussed.
- Conclusion and Outlook: The thesis is concluded by summarising the findings and providing an outlook on future research directions in this area.

The intention behind structuring the thesis in this specific way is to offer a comprehensive analysis of the role of Graph Neural Networks (GNNs) in forecasting urban mobility demand. This thesis also tries to help to understand how such forecasts might effectively shape upcoming strategies for urban planning.

Chapter 2

Related Work

In this chapter, an overview of related existing literature is provided to construct the foundations of this thesis. It is structured into three principal segments: Graph Neural Networks (GNNs), Urban mobility demand prediction, and the utilisation of GNNs for spatio-temporal prediction. The GNN architecture used in this project will be further investigated in chapter 3 Methodology.

2.1 Graph Neural Networks

Graphs have proven their potential to be a powerful representation of data and high dimensional relationships in different fields of science like Biology, Chemistry or Informatics. The history of Neural Networks (NNs) and graphs intertwines starting with the introduction of Recurrent Neural Networks (RNNs) to handle acyclic graphs, as documented by Sperduti and Frasconi [30, 6]. As the field evolved, it began to address more complex forms of data such as cyclic graphs.

One of the first works that marked the beginning of Graph Neural Networks was published by Scarselli et al. [27] in 2004. Until that time, most approaches transformed graph data into simpler structures such as vectors. However, this often resulted in a loss of information when it comes to the relationships between nodes. As an attempt to preserve the structure of graph data, Recursive Neural Networks were introduced. These networks aimed to maintain the graph structure as long as possible before preprocessing them. Simultaneously, random walks were employed for node-focused approaches, with the goal of keeping the graph's context intact. Scarselli et al. [29] and Micheli [21] furthered this work by incorporating RNNs and Feed-Forward Neural Networks (FFNNs) to tackle cyclic graph structures. The concept of Graph Neural Networks witnessed huge improvements with the integration of Convolutional Neural Networks (CNNs), as proposed by LeCun et al. [14]. The convolutional layer's capacity to extract local features from data greatly enhanced the performance of NNs on graph-structured data.

Graph Neural Networks were proposed to unify these approaches into one model, thereby enhancing their capabilities. The groundbreaking work of Scarselli et al. [29] in 2008 established the mathematical basis for modern GNNs, based on their prior research [27]. However, the proposed model was limited to static graphs and did not handle dynamic changes. Additionally, they pointed out the issues related to unknown relationships between nodes and learning a graph without prior knowledge, indicating future research directions.

After their first discovery, GNNs slowly gained attention in various domains, thanks to their ability to capture complex relationships between data points connected in non-euclidean domains. This ability offered a major change for many real-world applications that inherently exhibit graph-structured data, such as social networks, biological networks, citation networks, and transportation networks.

One of the critical milestones in the development journey of GNNs was the invention of the Graph Convolutional Network (GCN) introduced by Kipf & Welling in 2017 [13]. This work, which draws inspiration from CNNs, proposed the concept of graph convolutions, introducing an efficient and scalable approach for learning on graphs. By performing convolution operations directly on graph data, GCNs allowed localised first-order approximations of spectral graph convolutions, making them suited for semi-supervised learning tasks on graph data.

Velickovic et al. [32] proposed the Graph Attention Networks (GATs) in 2018, an advancement that used the attention mechanism to handle graph-structured data. The GAT model introduced a layer-wise attention mechanism that helped assign different importances to different nodes in a graph, enabling the model to focus on the most relevant parts of the input and thereby enhancing its capacity to capture underlying structures.

Building upon this line of research, Hamilton et al. [9] introduced GraphSAGE in 2017, a new algorithm that leverages node feature information to generate low-dimensional embeddings for nodes in large graphs. The framework allows inductive learning on large graphs, enabling the generation of embeddings for previously unseen data. This stands in contrast to earlier transductive approaches, where all the nodes have to be present during the training phase.

Temporal dynamics is a crucial aspect to consider when dealing with real-world graph data, as many applications involve graphs that change over time. To tackle this, Rossi et al. [25] introduced Temporal Graph Networks (TGNs) in 2020. TGNs bring the ability to handle dynamic graphs by combining both graph-structured and temporal data. The framework incorporates a memory module that captures the temporal dependencies of the nodes, leading to better performance on tasks that involve time-changing graph data. This work marked a big step towards the use of GNNs in applications where both the graph structure and feature information change over time.

2.2 Urban Demand Mobility Prediction

The task of forecasting urban mobility demand is important due to its wide ranging implications for urban planning, traffic management, policy-making, and sustainable development. Over the years, researchers have proposed and applied a lot of different approaches for urban mobility demand prediction, including statistical methods, traditional machine learning, and deep learning-based methods.

Statistical models have traditionally been used to tackle the prediction problem. For instance, Zhang et al. [40] performed prediction of traffic flow, a problem similar to mobility demand prediction. They explored several statistical techniques including

Auto-Regressive Integrated Moving Average (ARIMA) family models. However, these methods often ignore the spatial relationships between different points in a city where traffic is measured, which can lead to less accurate predictions.

Li et al. [15] took a different statistical approach to predict human mobility in a city using taxi GPS data. The resulting model helped taxi drivers locate their next passengers, effectively addressing hot-spot prediction. Nevertheless, the model's temporal data was grouped, causing loss of information such as for different days of the week. This approach can limit the model's capability to capture granular temporal variations in mobility patterns.

Similarly, Moreira et al. [22] employed time series forecasting techniques, with a specific focus on a 30-minute time horizon. They manually modelled various factors like seasonal demands and weather influence, yet, the data was still grouped. This method also overlooked spatial dependencies, resulting in significant information loss.

In a Bayesian approach [33], factors like time of day, day of the week, weather conditions, and area type were considered. Despite these factors, they found that only 54% of taxi trips were predictable, suggesting that nearly half of individual taxi trips could be random. Again, spatial relationships across the city were neglected, which could be a critical factor contributing to the model's unpredictability.

Lastly, Chen et al. [2] implemented a Generalised Additive Model (GAM) to predict the mobility of shared bicycles. While the paper provided valuable insights into shared bicycle mobility, the work did not elaborate on how well the model addressed spatial and temporal aspects of mobility prediction.

Traditional methods for urban demand mobility prediction often lack the ability to model complex spatial-temporal dependencies, which could result in inaccurate predictions. It is essential to incorporate these dependencies for a more effective and realistic prediction model.

With the arrival of machine learning, a new class of models was introduced for the task. Decision trees [4] and Support Vector Machines [12] have been used for predicting urban mobility demand. Despite providing better accuracy than traditional statistical methods, they come with their own set of limitations. These models often struggle when dealing with high-dimensional data and require significant feature engineering, which can be work-intensive and may not generalise well to unseen data.

In order to capture these complex relationships, more sophisticated models like Gradient Boosting Machines (GBM) and XGBoost have been employed [36, 5]. These algorithms have the ability to handle high-dimensional, sparse data and have proven to be robust against overfitting.

The K-nearest neighbours (K-NN) method has also been utilised in urban demand prediction [39]. K-NN models are particularly effective when there is a strong spatial correlation in the data, as is often the case in urban mobility demand datasets.

In the era of deep learning, researchers have experimented with advanced models like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) for the task [18, 8, 31, 24]. These models have the advantage of automatically extracting high-level features from raw data. In particular, RNNs, with their temporal dynamics capturing capability, have shown promising results in demand prediction tasks where temporal patterns play a crucial role. However, these methods often fail to fully capture the spatial dependencies inherent in urban mobility data, leading to sub-optimal performance in many cases.

2.3 GNNs for Spatio-Temporal Prediction

Recently, the application of Graph Neural Networks (GNNs) for spatio-temporal prediction tasks has gotten a lot of attention, especially in the field of urban mobility. GNNs' ability to capture both spatial and temporal dynamics in data have made them really effective for tasks such as traffic forecasting, demand prediction, and anomaly detection in traffic data.

One of the pioneering works using GNNs for spatio-temporal prediction is by Li et al. [16]. In this study, the researchers developed the Diffusion Convolutional Recurrent Neural Network (DCRNN), a deep learning framework that integrates both spatial and temporal data in traffic forecasting. This model was one of the first to apply GNNs for spatio-temporal prediction, and it set a good example for the use of this technology in predicting complex spatio-temporal data.

Yu et al. [37] proposed Spatio-Temporal Graph Convolutional Networks (ST-GCN) for citywide passenger demand prediction. The model was specifically designed to capture both spatial correlations and temporal patterns in the demand data. This work showed the effectiveness of GNNs for spatio-temporal prediction, as it outperformed several baseline models.

According to a comprehensive review by Zhou et al. [41], there exist different methodologies in GNNs for spatio-temporal predictions. Some approaches favor learning spatial relations first and then temporal patterns, while others learn both concurrently. A detailed categorisation by Huang et al. [11] provides an extensive framework to understand different methodologies for heterogeneous graphs.

Most studies on GNNs can be criticised for assuming that the underlying graph structure of the data is well-known. Wu et al. [35] were the first ones to address this problem by proposing an extra layer in their MTGNN model which first learns the structure of the graph. Furthermore, in the training phase of this model the graph structure gets adjusted.

Considering the strengths and effectiveness of GNNs in capturing spatio-temporal dependencies, as shown in the mentioned studies, it provides the foundation for the methodology used in this thesis. By combining these insights, the aim is to develop a GNN-based approach to predict urban mobility demand more accurately. The resulting model should outperform traditional statistical models and be at least on level with other machine learning techniques, offering a more effective tool for urban planning and mobility management.

However, it is important to note that while GNNs have shown promise, their implementation in this context is still in its early stages, and the field continues to evolve. In the next section, the proposed methodology will be presented, which aims to integrate the latest advancements in GNNs to better capture and forecast urban mobility demand.

Chapter 3

Methodology

In this chapter, the methodology employed in this research to predict urban mobility demand using Graph Neural Networks (GNNs) is presented. A step-by-step procedure is outlined, guiding the reader through the series of strategies and techniques that were executed in the development of this research. The goal is to ensure transparency and replicability of this research.

Firstly, the data collection will be presented, describing the source and nature of the spatio-temporal data that is the foundation of this research. The process of data processing and feature extraction will then be addressed, detailing how the raw data was refined and transformed into a suitable format as input for the GNN model.

Next, the core concept of graph representation is discussed, showing how the urban mobility features got encapsulated into a graph structure and how this representation leverages the underlying characteristics of the data. The reader will also be introduced to the fundamental mathematical principles of Graph Neural Networks, providing a foundation of understanding upon which the specifics of the model design and selection can be better comprehended.

Following that, the specifics of the selected Graph Neural Network architecture will be studied, including a detailed explanation of its layers and functionalities. The process of training the model is also described, analysing the choices of loss function, optimisation methods, and other crucial aspects that contributed to the learning process.

Furthermore, the model evaluation and validation techniques will be laid out, presenting the metrics used to assess the performance of the GNN model, as well as the procedure of data splitting for training and testing. Insights into the software, hardware, and libraries used to implement the model will also be provided, alongside any challenges faced during the implementation phase and the corresponding solutions.

Finally, ethical considerations and data privacy issues that arose during this research will be discussed, raising awareness for the importance to responsible and ethical researching.

3.1 Problem Formulation

Considering the temporal dimension of the spatio-temporal problem, the task in this research can be categorised as a time series forecasting or a sequence-to-sequence prediction. This means, that a time-ordered sequence of fixed length input will be used to predict a fixed length, temporally successive output sequence. In this context, the sequences have a chronological order to preserve the temporal dependencies.

As denoted by Prado-Rujas et al. [24], in the context of time series modelling, the size of the input window is referred to as the **timestep**, represented as n_x. In a similar way, the size of the output window or the **number of horizons** is denoted by n_y.

The variable s signifies the **shift** or the distance (in terms of time intervals) between the input and output windows. Whenever required, the parameters n_x, s, and n_y are indicated as subscripts of the model name to simplify interpretation.

For instance, ST-GAT*$_{8,4,4}$ is a notation for the ST-GAT* model where $n_x = 8$, $s = 4$, and $n_y = 4$.

Temporal granularity, or resolution, refers to the span of time into which time intervals are grouped. In the context of this research, the temporal granularity is 15 minutes. This can be represented as $t =$ [12:00, 12:15). For clarity, the start of the time period (e.g., 12:00) is used as a label for that particular interval.

The **temporal coverage** is the time duration over which certain data is collected. In this study, the temporal coverage for the mobility data spans from January 1, 2013 to March 1, 2020. This time span was chosen for two reasons: first, the COVID pandemic led to big changes in taxi and rental bicycle demand, therefore the resulting data would not be suitable for training or testing due to a high bias. Secondly, other research, especially Prado-Rujas et al. [24], use a similar time span to train and test their approaches, which makes it possible to compare parts of the results.

For every time interval a graph is constructed. In the context of GNNs, a graph $\mathcal{G}$ is described as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes (or vertices), and $\mathcal{E}$ is a set of edges. Nodes represent entities and each is associated with a feature vector $\mathbf{x}_v \in \mathbb{R}^d$, where d is the dimension of the feature space. Edges represent relationships or interactions between entities.

In the case of taxi rides, each centroid location present in the dataset corresponds to a node, solely derived from the data itself. Logically, the fixed bicycle stations are used as nodes when the model treats bicycle data. Edges are constructed separately which will be discussed in detail in section 3.4.

Additionally, there are two sets of data which are used alongside the input graphs: $\mathcal{W}$ and $\mathcal{T}$. $\mathcal{W}$ is the set of meteorological weather data and $\mathcal{T}$ is a set of time-series data. These will be important as demands change depending on the time of the day, week or year, and depending on the temperature or weather condition.

Concluding, if the input window of the model is $n_x = 8$, then for each sequence there will be eight input node sets $\mathcal{V}_{t_1}, \ldots, \mathcal{V}_{t_8}$ (here: the amount of taxi or bicycle rides per station in one time period), an adjacency matrix $\mathbf{A}$ which represents the edges and the additional weather and time-series data. The input window in this case would cover two hours of data. With $s = 4$ and $n_y = 4$ the model predicts the node values for a time window of one hour for every possible sequence, beginning one hour

($s = 4 * 15min = 60min$) after the last input timestamp. In this case, the four predicted sets of node values would be $\mathcal{V}_{t_{12}}, \ldots, \mathcal{V}_{t_{15}}$, where the nodes and edges are the same as for the input but only the node values are calculated.

Mathematically, the problem can be stated in the following way:

$$\{\mathcal{V}_{t_1}, \ldots, \mathcal{V}_{t_{n_x}}, \mathcal{E}, \mathcal{G}, \mathcal{W}, \mathcal{T}\} \mapsto \{\hat{\mathcal{V}}_{t_{n_x+s}}, \ldots, \hat{\mathcal{V}}_{t_{n_x+s+n_y-1}}\}$$

Figure 3.1 summarises the problem by portraying the graphs as slices in a time axis. However, it is important to note that the model does not predict the whole graph structure. The graph structure stays the same for every time interval, as the taxi pick-up centroids and bicycle stations are fixed for the data. Experimenting with changing edges could be interesting for future work. For example, a different graph adjacency for different times of the day or months in a year.

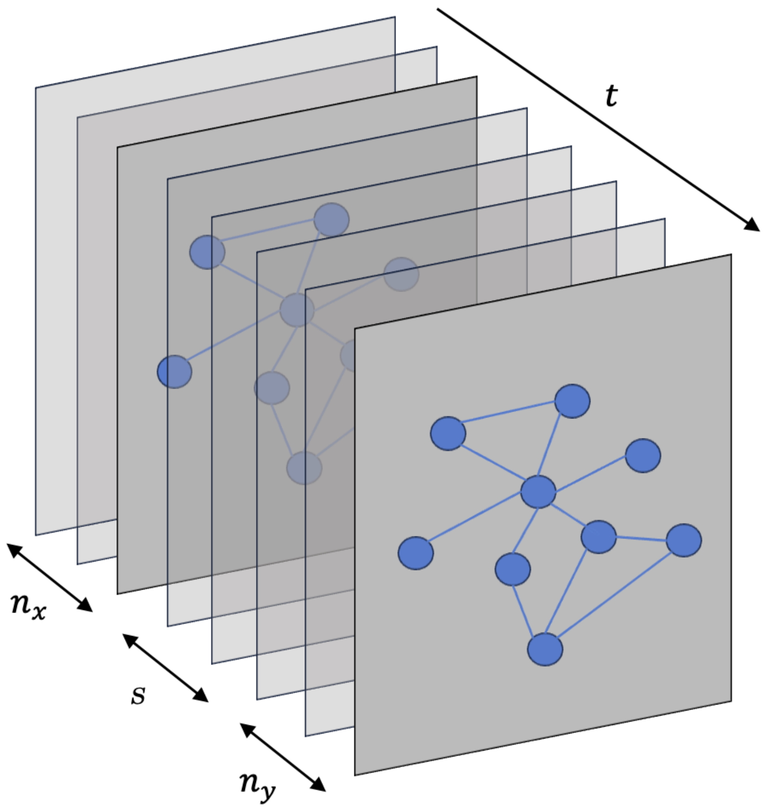


Figure 3.1: A full window sequence consists of n_x, s and n_y graphs for a time window $t_{n_x}, \ldots, t_{n_x+s+n_y-1}$. The graphs and its node values of n_x time frames are used to predict the node values of the n_y graphs, s time intervals in the future.

3.2 Data Collection

In the context of Graph Neural Networks (GNNs), data collection has to be approached with great attention to detail, given that these networks need structured and interconnected data. The data should ideally provide a good coverage of the domain in question, ensuring that the resultant model is trained on a representative sample.

Also, how well the data is linked can greatly affect how well the GNN can learn and make right guesses. Therefore, gathering data in this area is like trying to find the right balance between the amount of data, the detail of the data, and how well the data instances are connected to each other.

Taxi Trips

The city of Chicago provides hundreds of public datasets, one of them is the Taxi Trips dataset [1]. It is an extensive set of data and is collected by the City of Chicago's Business Affairs and Consumer Protection Department, which licenses all taxi drivers and vehicles operating in the city. The data is gathered through digital systems for each taxi, often referred to as 'taximeters', which record a variety of data during every trip.

Every time a taxi ride starts and ends, these taximeters capture key information about the ride, such as the start and end times, trip duration, pick-up and drop-off location, distance travelled, and fare charged. This data is then transmitted to the city's data systems.

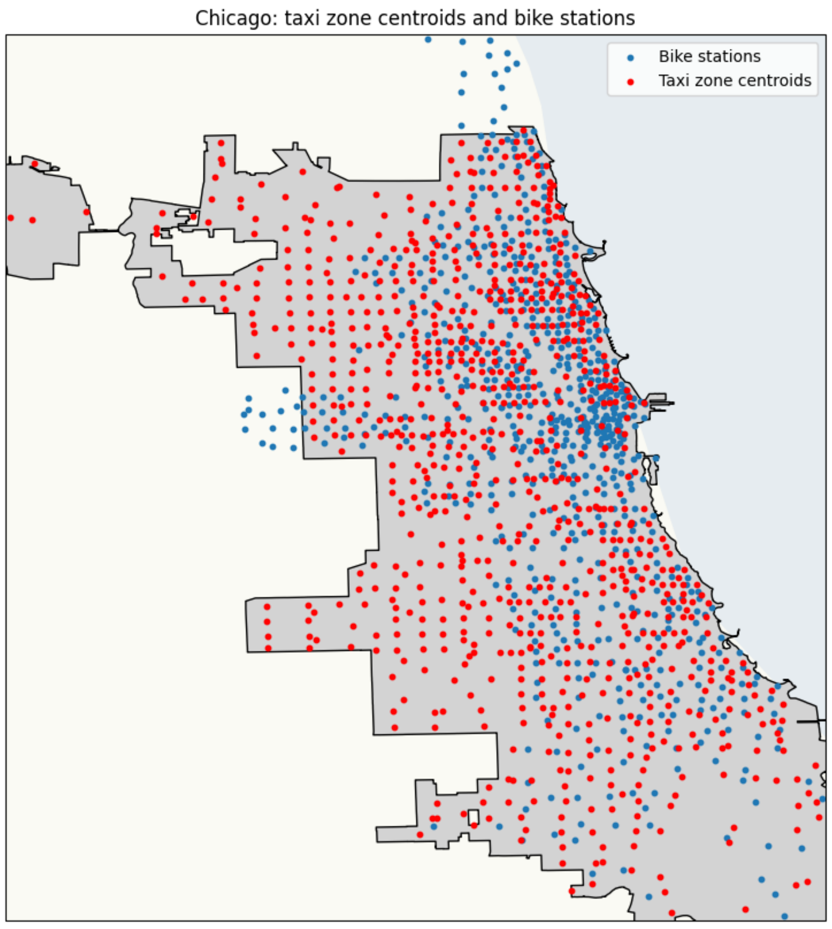


Figure 3.2: The image illustrates a map of Chicago, highlighting the distribution of 861 taxi pickup locations and 684 bicycle stations. It also shows the location of the weather station which provided the meteorological data.

[1]https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew

This dataset is anonymised for privacy, with no personally identifiable information about taxi drivers or passengers included. The different pick-up and drop-off locations were clustered using a centroid based approach, which will later be used as the nodes of the taxi data. The dataset is updated regularly, providing a continually evolving snapshot of taxi usage in Chicago.

Bicycle Rides

The Divvy Bicycle Trips dataset is a public resource provided by Divvy, the bicycle sharing system of Chicago [2]. This dataset is updated monthly and contains anonymised historical data of all bicycle trips taken using the Divvy system.

Users of Divvy need to sign up and provide information about themselves in their account. Once they unlock a bicycle at a station, a record with metadata regarding the trip is created. Each record includes details such as the start and end times of the trip, the locations of the start and end stations, user information like age and gender, and the rider type which could be a Member, Single Ride, or Day Pass.

As stated by the provider, the data has been cleaned and processed to remove any trips taken by Divvy staff during system servicing and inspection, as well as trips that were under 60 seconds in duration, which may represent false starts or instances of users trying to ensure a secure bike docking.

Time-Series Data

The time-series dataset was developed inspired by the implementation of Prado-Rujas et al. [24]. Every individual time interval within this temporal range is assigned eight characteristics. The first two are boolean indicators, specifying whether the given time interval falls on a weekend or a public holiday. Furthermore, two metrics are provided to denote the specific time of day, represented through the mathematical sine and cosine functions. Similarly, a pair of values, derived using the same sine and cosine functions, represent the day of the week. Finally, two additional values, derived in the same mathematical manner, signify the specific period in the annual cycle.

Weather Data

As there is no public dataset of meteorological data for the city of Chicago on an hourly basis and in the temporal coverage, a dataset was acquired from the weather data provider VisualCrossing [3]. The temperature ($°C$), wind speed (km/h) and a description of the weather condition were considered as important factors for the model of this research. While other measurements such as the humidity, wind direction or UV index were also accessible, only these three characteristics were chosen as they are most present for people using a taxi or bicycle.

Figure 3.2 shows a map of Chicago with the 861 taxi pickup zone centroids and the initial 684 bicycle stations in the temporal coverage across the city. Due to changes in the city's limits over the years, a few stations lie beyond the city's boundary.

[2]https://divvybikes.com/system-data

[3]https://www.visualcrossing.com

3.3 Data Processing and Feature Extraction

Data processing is a fundamental step in any data-driven research, enabling the extraction of meaningful insights from raw data. In the context of predicting urban mobility demand using spatio-temporal data, data processing plays a critical role in transforming the collected information into a format suitable for analysis and modelling. Once it is processed, the features can be extracted which are the inputs of the model. This section provides an overview of the data processing techniques employed in this study, outlining the steps involved in preparing the data for following stages of the methodology.

Taxi Trips

For the purpose of this work, only the timestamp and the location (latitude and longitude) of each taxi ride is needed. Furthermore, the trips were grouped and counted by timestamp and location. This can be done separately in a preprocessing step after downloading all the data, but the data portal of the city of Chicago provides a SQL-like querying method to filter and group data. The downloaded file then contains four columns: the timestamp at the beginning of a time period, the centroid's latitude, the pick-up centroid's longitude and the amount of taxi rides in this period and at this location.

Derived from these locations a lookup table for the centroids was constructed, containing a unique ID, the latitude and the longitude of each taxi pick-up centroid. This lookup table served to enhance the taxi trips with node IDs.

In a next step, the taxi trips file got increased to be a matrix M of dimension $t \times n$, where t is the amount of time periods in the temporal coverage and n the amount of nodes. From January 2013 to March 2020, a span of 2617 days was considered. On each of these days, there are 96 time windows, each lasting 15 minutes.

The amount of nodes was reduced from 861 to 258 due to computing memory restrictions. This was done by calculating the 70th percentile of the sums of all node values. Every node whose sum of all taxi trips was smaller than the 70th percentile was removed. Therefore, the final matrix M has the dimensions $(2617 * 96 =)251232 \times 258$. Despite the significant reduction of nodes from 861 to 258, amounting to a 70% decrease, it is important to note that this reduction only resulted in the removal of 0.13% of the total number of taxi rides. A majority of the centroid pick-up locations had minimal or no taxi trips recorded throughout the eight-year coverage period, making them less relevant to this forecasting task.

To extract the feature representation, the Speed2Vec approch by Zhang et al. [38] was used. In this case, the amounts of taxi trips was transformed into input vectors of size n_x and output vectors of size n_y. The node values (here: amount of taxi trips) were normalised using the z-score.

Bicycle Rides

Prado-Rujas et al. [24] provide a preprocessed file in the format and time frame that is needed for this research [4]. Like for the taxi trips, the data first was grouped by timestamp and location. The format of the matrix is again $t \times n$, where t is the amount of time periods in the temporal coverage and n the amount of nodes, i.e.

[4] 15m_flat_bike_count.h5

$(2617 * 96 =)251232 \times 428$. The original file contains 857 locations, but again due to memory restrictions 50% of the stations were removed. In total, only 0.2% of all bicycle rides were removed by this operation. Later on, the temporal coverage for the bicycles was reduced, starting now on the 28th of June. Before that day, there were no recorded bike rides and therefore no significant data.

Again a feature extraction in the style of Speed2Vec was performed in order to be able to feed the model with data and the amount of bicyle rides was normalised using the z-score.

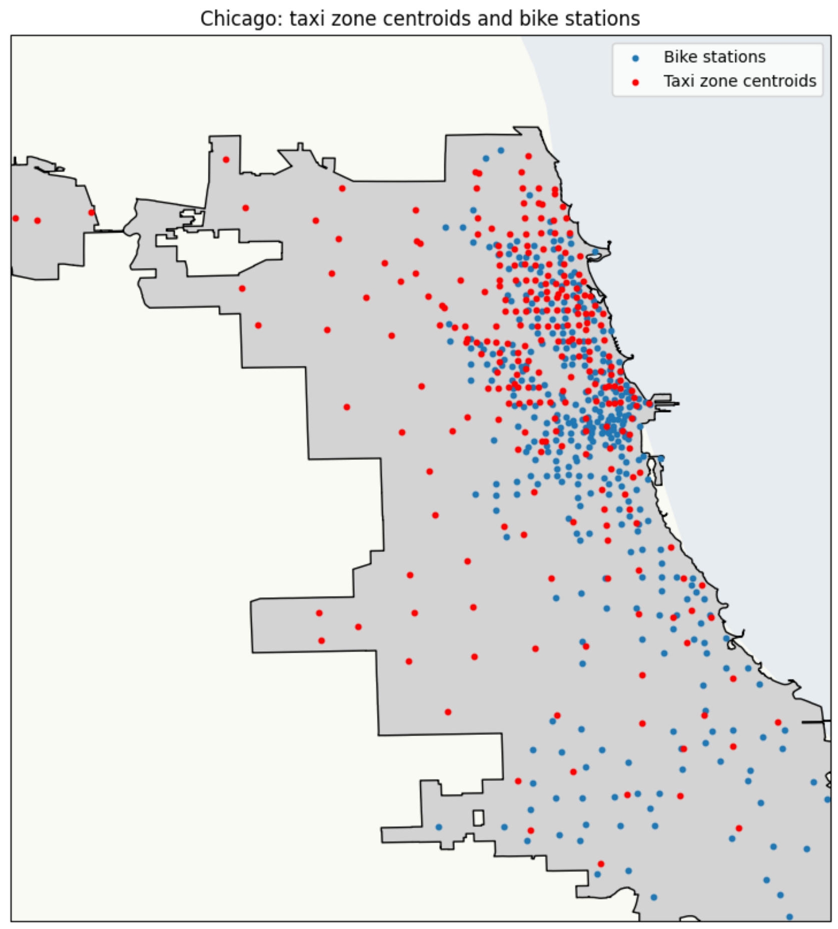


Figure 3.3: Another representation of a map of Chicago with the taxi pickup zone centroids and bicycle stations scattered onto it. Here the amount of centroids and stations was reduced to 258 and 342 respectively.

Figure 3.3 portrays the centroids and bike stations that remain after the reduction. The map is a lot more sparse than in figure 3.2, but it still captures captures more than 99% of all taxi trips and bicycle rides in the temporal coverage. The amount of bicycle stations in the map and in the used data do not match because after 2018 Divvy included the locations with each ride which led to a lot more unique locations. Before they would use lookup table for the stations and map them to them. So portrayed here are only the stations before 2018.

Time-Series Data

The dataset was created in the right format and therefore needed no more processing.

However, there were considerations for the feature extraction. Instead of considering all n_x records for each sequence, only the time-series data for the last time frame was saved with the timestep. The reason behind this is that in a realistic scenario the prediction would be made in the present moment, therefore only the specific features of that timestamp are important. Another reason was that this would reduce the amount of data being saved to the machines memory.

Weather Data

To prepare the dataset for the training phase, the z-score function was applied to the temperature and wind speed as means of normalisation. The description of the weather condition was grouped into four categories:

- Sunny: Indicating zero to slight coverage of the sky
- Cloudy: A sky full of clouds
- Rainy: Ranging from drizzling to heavy rainfall
- Snowy

Every meteorological record was then grouped into one these categories, using boolean values. The final dataset therefore contained 6 features: temperature, wind speed and the four weather conditions. Like for the time-series data, only the latest record out of the n_x observations was saved.

PyTorch Geometric Dataset

Before the preprocessed data can be used by the ST-GAT model it has to be processed into a PyTorch Geometric dataset, which is a python class by PyTorch to load data more efficiently. The before described matrices furthermore need to be transformed into sequences with x (input) and y (output) values, in order for the model to be trained on it. This step is also important for the training process in batches: a PyTorch Geometric dataset can be used by a PyTorch dataloader which efficiently divides the workload on the different cores of the GPU.

Per day there are $96 - (n_x + s + n_y) + 1$ possible full window sequences. One full window sequence can be used for training, validation or testing as it contains the input and output sequence. All the sequences which cover two days were not considered, as this makes it harder to divide the dataset or use different batch sizes. After all these steps, the data is completely processed and ready for the training phase.

3.4 Graph Representation

In order to analyse the urban mobility demand using spatio-temporal data, the data must be represented in an appropriate form that can capture the underlying spatial relationships and dependencies. Graphs serve as an effective tool for this task, allowing for a robust mapping of data points as nodes, and their relationships as edges. The strength of these relationships can be quantified using various metrics. In this research, three approaches have been used to represent these relationships: correlation matrices, cosine similarity matrices and the thresholded Gaussian kernel method .

Correlation Matrix Approach

In the correlation matrix approach, the Pearson correlation coefficient is used to determine the similarity between each pair of nodes in the graph. For a graph with N nodes, the correlation matrix **C** is an $N \times N$ symmetric matrix where each entry c_{ij} represents the correlation between nodes i and j.

The Pearson correlation coefficient between nodes i and j is calculated using the formula:

$$c_{ij} = \frac{\sum_{k=1}^{T}(x_{ik} - \bar{x_i})(x_{jk} - \bar{x_j})}{\sqrt{\sum_{k=1}^{T}(x_{ik} - \bar{x_i})^2 \sum_{k=1}^{T}(x_{jk} - \bar{x_j})^2}} \tag{3.1}$$

where T is the total number of observations, x_{ik} and x_{jk} are the k-th observations of nodes i and j respectively, and $\bar{x_i}$ and $\bar{x_j}$ are their respective mean values [23].

For both the taxi and bicycle data, the correlation matrix was reduced before serving it as an adjacency matrix, because the amount of $N \times N$ edges would drastically increase computed calculations later on. Therefore, different settings of sparsity were tested.

By keeping the n highest absolute values of the correlation matrix, nodes will only be connected if they correlate positively or negatively with one another. For this research, n was set to 10, 30 and 50.

Cosine Similarity Matrix Approach

The cosine similarity approach, on the other hand, focuses on the angle between two vectors of node values to quantify their similarity. Like the correlation matrix, the cosine similarity matrix **S** is also an $N \times N$ symmetric matrix, where each entry s_{ij} represents the cosine similarity between nodes i and j.

The cosine similarity is calculated using the formula:

$$s_{ij} = \frac{\mathbf{x_i} \cdot \mathbf{x_j}}{||\mathbf{x_i}||_2 \times ||\mathbf{x_j}||_2} \tag{3.2}$$

where $\mathbf{x_i}$ and $\mathbf{x_j}$ are the vectors representing the time series of nodes i and j respectively, $||\mathbf{x_i}||_2$ and $||\mathbf{x_j}||_2$ represent their respective Euclidean norms, and $\cdot$ is the dot product operator.

Again, an initial adjustment by decreasing the size of the cosine similarity matrix was made, which was then used as an adjacency matrix. This was done to prevent a potential outburst in calculations associated with the $N \times N$ edges that would appear further in the process.

As with the correlation matrix, different levels of sparsity were tested. This involved keeping the n strongest positive or negative relationships from the similarity matrix, which made sure that a connection between points only happened if they were either strongly similar or dissimilar. 10, 30, and 50 were used as values for n.

Thresholded Gaussian Kernel Method

The Thresholded Gaussian Kernel method is a technique used for constructing adjacency matrices in graph-based machine learning tasks, particularly for Graph Neural

Networks (GNNs). The method can be used to build a weighted adjacency matrix to describe the strength of connections between nodes in a graph. [38]

Firstly, a Gaussian Kernel is used to compute similarities between pairs of nodes. The Gaussian Kernel function is defined as follows:

$$k(x_i, x_j) = e^{-\frac{||x_i - x_j||^2}{\sigma^2}} \tag{3.3}$$

where x_i and x_j represent two different nodes, $||.||^2$ denotes the Euclidean distance, and σ is a parameter that controls the width of the Gaussian Kernel. In case of $i = j$, $k(x_i, x_j) = 1$ to include a self-connection for the nodes. In this research, the Euclidean distance was measured for the coordinates of each node with one another.

The Gaussian Kernel computes a similarity value between pairs of nodes. These values are then used to create an adjacency matrix $\mathbf{A}$, where a_{ij} represents the similarity between nodes i and j. As such, $\mathbf{A}$ becomes a weighted adjacency matrix, where the weights correspond to the strength of relationships between the nodes.

However, using this approach might lead to a dense graph, where almost all nodes are connected to each other with different weights. This can be a problem when dealing with large-scale problems, as dense graphs lead to computational inefficiency. This is where the concept of thresholding comes into play.

In the thresholding step, a cut-off value is determined. Any similarity score below this cut-off value is set to zero, which effectively removes the corresponding edge from the graph. This process is also known as sparsification, which results in a sparser, more computationally efficient graph.

3.5 Overview of Graph Neural Networks (GNNs)

Graph Neural Networks (GNNs) are a class of deep learning methods designed to perform inference on data described by graphs. GNNs operate by learning to aggregate information from a node's local neighbourhood, thus enabling them to leverage both the features of the nodes and the structure of the graph.

The GNN operates by first applying a neighbourhood aggregation function, or propagation rule, to each node, which results in each node v receiving an aggregated representation of its neighbours' features. [34]

The process of node aggregation in GNNs, as depicted in figure 3.4, can be broken down into several key steps. This process is also often referred to as the 'message passing' paradigm in graph-based learning.

1. Message Generation: Each node in the graph generates a message, which is typically a function of its current state (i.e., its feature vector), and possibly the states of its immediate neighbours. The message summarises what the node 'wants to communicate' to its neighbours. This process is often governed by a shared message function, which ensures that the generation of messages is consistent across the entire graph.

2. Message Aggregation: Next, every node collects and aggregates the messages sent by its neighbouring nodes. This aggregation step helps to summarise the

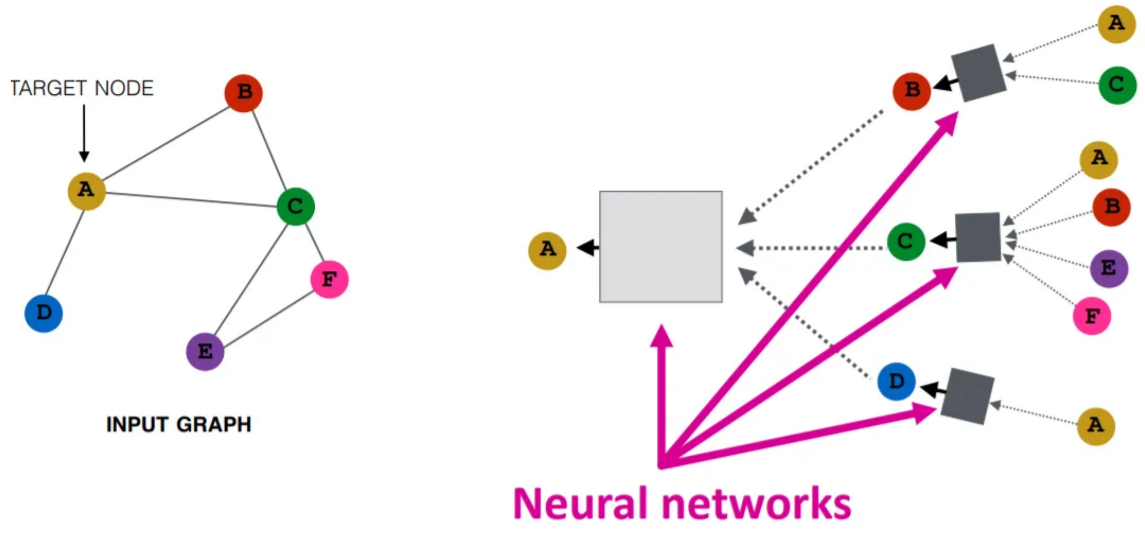


Figure 3.4: This figure shows the key steps involved in the operation of a Graph Neural Network (GNN) for a node in a graph. It illustrates the concept of 'message passing' in the network, wherein each node receives and sends information to its immediate neighbours. (Image copied from P. Mehta [20])

information from the local neighbourhood. The aggregation can be a simple function like the sum, average, or more complex functions such as LSTM or max pooling. The goal of this function is to be permutation invariant, meaning it shouldn't matter in which order the messages are received.

3. Node Update: The aggregated message, also known as the 'neighbourhood representation', is then used to update the node's own state. This state update is usually performed through a function that takes the node's current state and the aggregated message as input, and produces a new state as output. This function is typically a neural network, which allows GNNs to learn complex patterns.

4. Readout: After several rounds of message passing, a readout function is used to generate the final output of the GNN. The readout function aggregates information across all nodes in the graph to produce a graph-level output. This is especially useful in tasks where the prediction needs to be made for the entire graph (e.g., graph classification tasks).

These steps are iteratively performed for a number of rounds (also known as graph convolution layers), allowing information to propagate through the graph. As the number of iterations increases, nodes have access to information from further away in the graph, enabling them to 'see' larger parts of the overall structure. This iterative process helps the GNN to capture both local and global properties of the graph.

It should be noted that the design of the message generation, message aggregation, state update, and readout functions are all critical aspects of GNN design, and many different choices can be made for these functions depending on the specific application or task at hand.

A significant advantage of GNNs over other types of neural networks is their ability to naturally handle irregular data. They are invariant to the order of the nodes in the graph, and they can process graphs of different sizes and shapes. This makes them particularly suitable for many real-world data, which often comes in the form of graphs, such as social networks, biological networks, or transport networks.

3.6 Model Selection and Design

The ST-GAT* model constructed in this research is a hybrid of the architecture of Prado-Rujas et al. [24] and the ST-GAT model from Zhang et al. [38]. It incorporates the spatio-temporal data of taxi and rental bicycle rides with an additional time-series and weather module.

ST-GAT Model

The spatio-temporal Graph Attention Network (GAT) model uses a powerful architecture for the task of predicting urban mobility demand using spatio-temporal data. The choice of this model was informed by its ability to process the spatial structure of the data while also taking into account the temporal aspects, thereby encapsulating the essence of the data. It was also selected because it performed well on urban mobility data. In their research, Zhang et al. predicted traffic speed across the city of Los Angeles.

The spatio-temporal GAT model is a combination of a spatial GAT block and two Long Short-Term Memory (LSTM) units. Each of these components play a significant role in the functionality and performance of the model. Figure 3.5 provides an overview of the model.

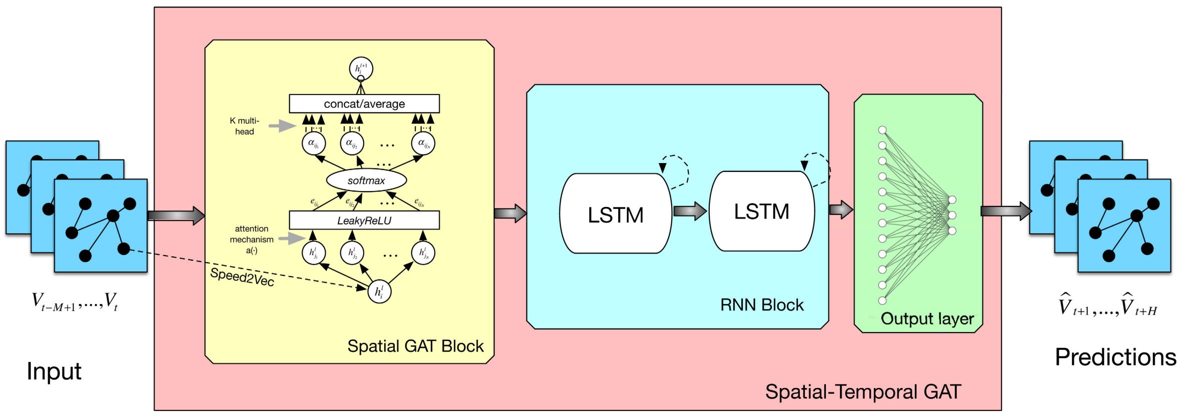


Figure 3.5: This figure shows the architechture of the spatial-temporal graph attention networks (ST-GAT). (Image copied from Zhang et al. [38])

Spatial GAT Block

The spatial GAT block makes up the first mechanism of the spatio-temporal GAT model, responsible for processing the spatial structure of the data. This block consists of a GATConv layer from the PyTorch framework, leveraging the Graph Attention Networks (GAT) mechanism.

The propagation rule used in Graph Attention Networks (GATs) is originally formulated as [32]:

$$\mathbf{x}'_i = \alpha_{i,i}\mathbf{\Theta}\mathbf{x}_i + \sum_{j\in\mathcal{N}(i)} \alpha_{i,j}\mathbf{\Theta}\mathbf{x}_j$$

where the attention coefficients $\alpha_{i,j}$ are computed as

$$\alpha_{i,j} = \frac{\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^\top\left[\mathbf{\Theta}\mathbf{x}_i \| \mathbf{\Theta}\mathbf{x}_j\right]\right)\right)}{\sum_{k\in\mathcal{N}(i)\cup\{i\}}\exp\left(\mathrm{LeakyReLU}\left(\mathbf{a}^\top\left[\mathbf{\Theta}\mathbf{x}_i \| \mathbf{\Theta}\mathbf{x}_k\right]\right)\right)}.$$

Here, first the attention coefficient between two nodes is computed with LeakyReLU as the activation function, and $\mathbf{a}^\top$ is a shared learnable weight vector. $||$ denotes the concatenation operation. The denominator normalises the coefficients across all choice of neighbours, ensuring that they sum up to 1. Finally, $\mathbf{x}'_i$ is the updated feature of node i which is a weighted sum of its neighbours' features, with weights determined by the normalised attention coefficients.

GATs have several layers of propagation, enabling the model to gather information from an increasingly larger part of the graph and learn complex, multi-node patterns.

The spatial GAT block in the ST-GAT* model consists of 32 attention heads. Multiple attention heads allow the model to focus on different parts of the input data, which is like viewing it from different perspectives. As such, the GATConv layer with 32 attention heads enables the model to extract multiple, different, and complementary spatial feature representations from the data, significantly boosting the model's ability to understand the spatial aspects of the urban mobility demand.

LSTM Units

To capture the temporal relationships in the data, two LSTM units were deployed. The first LSTM unit has one hidden layer of size 256, and the second LSTM unit has one hidden layer of size 512.

After the input runs through the spatial GAT block, its output enters the first LSTM. For each element in the input sequence, each layer computes the following function:

$$\begin{aligned}
i_t &= \sigma\left(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}\right) \\
f_t &= \sigma\left(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}\right) \\
g_t &= \tanh\left(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}\right) \\
o_t &= \sigma\left(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}\right) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh\left(c_t\right)
\end{aligned}$$

where h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the input at time t, h_{t-1} is the hidden state of the layer at time $t-1$ or the initial hidden state at time o, and i_t, f_t, g_t, o_t are the input, forget, cell, and output gates, respectively. σ is the sigmoid function, and $\odot$ is the Hadamard product. [26]

LSTMs are a type of Recurrent Neural Network (RNN) designed to remember temporal dependencies for long periods of time, making them ideal for handling temporal data. In the context of this problem, the temporal aspect is critical since the demand for urban mobility is heavily influenced by time patterns.

The big layers in each LSTM unit enable the model to extract high-level temporal features from the data. The first LSTM, with its 256-sized hidden layer, processes the initial temporal patterns in the data. Its outputs are then fed into the second LSTM, which, with its larger 512-sized hidden layer, is capable of capturing more complex temporal patterns based on the features extracted by the first LSTM.

This chosen spatio-temporal GAT module leverages the strengths of GAT and LSTM architectures to process the spatial and temporal aspects of urban mobility demand data respectively. The design choices regarding the number of attention heads in the GATConv layer and the sizes of the LSTM hidden layers have been made to balance the complexity of the model with its ability to accurately capture the intricate patterns in the data.

ST-GAT*

Following the architecture established by Prado-Rujas et al. [24], the model developed for this research was built. The entirety of the model is visually represented in Figure 3.6. The model is composed of three key components: the ST-GAT block, the weather block, and the time-series block.

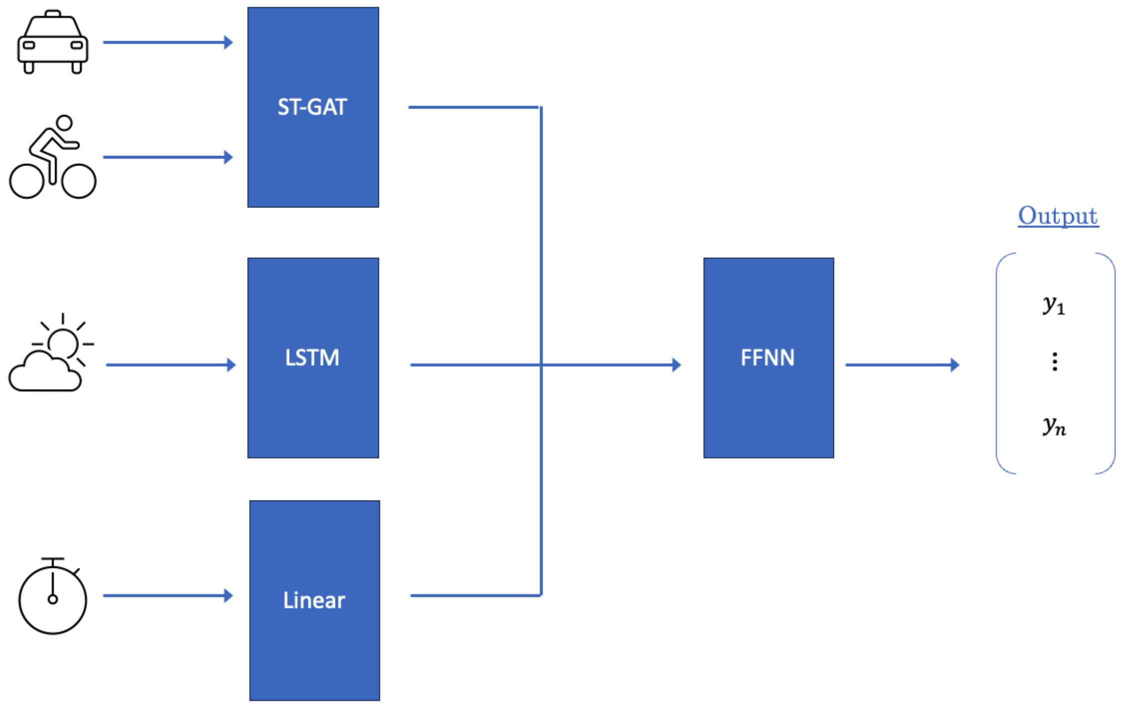


Figure 3.6: An abstraction of the ST-GAT* model as proposed in this thesis.

The weather block employs a Long short-term memory (LSTM) unit, which is responsible for learning the underlying weather patterns that show up in the data. LSTMs are designed to understand both short-term and long-term patterns in a sequence, which makes it ideal for weather patterns that can change rapidly or evolve over a long period. The inputs to this LSTM unit are the six distinct meteorological features that are detailed earlier in section 3.3. It produces n features corresponding to the number of nodes that are present in the input data. Concluding, for every input node there will be one value from the weather unit, representing what influence the current meteorological conditions will have on the predictions.

Next up is the time-series block. Its function is to apply a linear transformation to the eight time characteristics associated with each input sequence. This block, like the weather block, also outputs n features for the number of predicted node values. This one value for each node will determine, whether the current timestamp leads to higher or lower predictions. E.g., the demand for taxis could be higher at rush hours, but if it is 04:00 AM in the morning the value should represent that and not lead to predictions that are too high.

Finally, these three blocks come together. This is achieved using a feed-forward neural network (FFNN). It has a hidden layer of size 128 to learn how the three units influence each other. This step takes in $n_y + 2$ vectors, each of size n, and returns the predicted horizons for each node. It's a simple way of integrating the outputs from the individual blocks into a final, coherent prediction.

Weight Initialization

In the design phase of an AI model, a choice has to be made about how the initial weights of the model are set. The type of weight initialisation can impact the effectiveness of training. For example, if weights are initialised to be too large or too small, this can lead to issues with the learning process, such as vanishing or exploding gradients.

Common initialisation methods are Xavier/Glorot, He or Orthogonal initialisation. The GATConv layer and LSTMs traditionally use the Xavier initialisation. The idea behind Xavier/Glorot initialisation is to make the variance of the outputs of a layer to be equal to the variance of its inputs, in order to combat the vanishing and exploding gradients problem that can occur during training. In the original paper by Glorot and Bengio where Xavier initialisation was proposed [7], the authors note that this form of initialisation is beneficial for activation functions that are symmetric around zero and have outputs inside the range [-1, 1] like the hyperbolic tangent (tanh), which is commonly used in LSTMs.

In the case of this thesis, both the bicycle rides and taxi trips ST-GAT* model performed better with the Xavier initialisation method than the Orthogonal initialisation, which is also commonly used in LSTMs. He initialisation is designed specifically for rectified linear units (ReLUs) and variants thereof. ReLUs are asymmetric and have an output range of [0, ∞]. If He initialisation is used with an activation function like tanh, it could lead to issues with the variance of the outputs and that is why it was not used.

3.7 Training the Model

In the attempt to construct an effective Graph Neural Network (GNN) model that can predict urban mobility demand utilizing spatio-temporal data, it is fundamental to effectively train the model to perform optimally. This section discusses the main components of the model training process, detailing the selected loss function, optimisation method, regularisation technique, and batch size. These elements are critical to the model's learning process, the optimisation of the weights and biases, and the overall performance of the model.

Loss Function: Mean Squared Error (MSE)

The chosen loss function for the model is the Mean Squared Error (MSE) loss, as opposed to the Mean Absolute Error (MAE) loss. The MSE loss function, which is the squared difference between the actual and the predicted values, has proven to be more effective for this task, leading to better predictions. This might be due to its property of heavily penalising larger errors more than smaller ones, thus forcing the model to pay more attention to larger differences, which is particularly beneficial in the case of this research where the aim is to predict mobility demand with a high precision.

Training the model for ten epochs with the two different loss functions showed that the MSE constantly outperformed the MAE loss, not just for the evaluation metrices which are mentioned in section 3.8 but also when looking at example predictions for the taxi and bike stations with the highest activity.

Optimisation Method

The optimisation method utilised for most of the training of the GNN is the Adaptive Moment Estimation (Adam), which was selected for its stability and robustness. Adam was initialised with a learning rate between 1×10^{-4} and 5×10^{-4}, which was a good balance between learning speed and convergence stability. In addition, a learning rate decay of 1×10^{-5} was employed, helping the model to refine the learning as the epochs progress and avoid overshooting the global minimum.

This combination has shown to outperform the Stochastic Gradient Descent (SGD) optimizer in terms of minimizing the loss function. With SGD, the loss function fluctuated significantly, which can indicate a struggle to find the global minimum in the loss landscape.

For the bicycle data, the RMSprop (Root Mean Square Propagation) optimisation algorithm also yielded good results. RMSprop adjusts the learning rate based on the root mean square of recent squared gradients, scaling it down for large gradients and up for small ones. Especially with a momentum of 0.9 it performed better than Adam on the bicycle data.

Other optimisation methods that were tested for at least one training cycle of ten epochs were: Rprop (Resilient Backpropagation), ASGD (Averaged Stochastic Gradient Descent), Adamax (a variant of Adam based on the infinity norm), AdamW (implementation of weight decay in Adam), Adadelta (adaptive learning rate method in Adam) and NAdam (Adam with Nesterov momentum). All of them performed worse than Adam or RMSprop with momentum and were not further considered.

Regularisation Technique: Dropout

For the regularisation technique, dropout was used with a rate of 0.1 to 0.3 to prevent overfitting. Dropout is a simple yet effective method to mitigate overfitting by randomly setting a proportion of the input units to 0 during training, which helps to make the model more robust and generalisable by preventing complex co-adaptations on training data.

Batch Size and Number of Epochs

The model was trained with a relatively large batch size of 2048 for the bicycle rides data and 256 for the taxi trips data. A smaller batch size was initially tested, but it led to a highly fluctuating loss function, suggesting that the model was unable to generalise well across the dataset. While a larger batch size requires more computational resources, it has the advantage of giving a more accurate estimate of the gradient over the entire dataset, resulting in a more stable learning process.

Given the high batch sizes, a higher number of epochs was needed to ensure sufficient iterations through the entire dataset. After various trials, 30 epochs were chosen as an optimal number, balancing the need for the model to learn the data patterns and the risk of overfitting. This configuration was found to provide a stable decrease in the loss function over the epochs, suggesting that the model was learning effectively without memorising the training data.

3.8 Model Evaluation and Validation

Model evaluation and validation is an essential phase in any AI project, offering important feedback on the model's performance. This step ensures that the chosen model is reliable and robust enough to capture the underlying patterns in the data and make accurate predictions on unseen data. In this section, there will be an analysis on how the evaluation and validation of the ST-GAT* model was executed, highlighting the dataset partitioning, metrics used, and how the model's performance was measured.

Dataset Partitioning

To accurately assess the performance of the model,the dataset was divided into three distinct parts: training, validation, and testing sets.

- **Training Set**: The largest subset of the data, covering the period from January 1, 2013, to December 31, 2017. The bicycle dataset covered the time period from June 28, 2013 to December 31, 2017, because there was no data for the first months of 2013. The training set was used to build and calibrate the model. Through this iterative learning process, the model adjusted its parameters to minimise the error on the training data and capture the underlying data patterns.
- **Validation Set**: This subset was used to fine-tune the model parameters and architecture while ensuring that the model does not overfit the training data. This period spans from January 1, 2018, to December 31, 2018.
- **Test Set**: After the model was fully trained and optimised, the test set was used to evaluate the model's final performance. This independent data subset, ranging from January 1, 2019, to March 1, 2020, was previously unseen by the model and offered an unbiased estimation of how the model might perform in a real-world context.

Therefore, the taxi trips data is divided into a 70%-14%-16% split while the bicycle rides data is divided into a 68%-15%-17% split, due to the missing data.

Evaluation Metrics

For the model evaluation, three statistical error metrics were used: Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Underestimate Penalising Error (UPE). These metrics provide a quantitative measure to compare the performance of the model against different benchmarks and alternative models.

- **Mean Absolute Error (MAE)**: This metric quantifies the absolute difference between the model predictions and the actual values. It is an interpretable metric since it provides the average error in the same units as the original data.

$$\mathrm{MAE}(v, \hat{v}) = \frac{1}{N}\sum_{i=1}^{N} |v_i - \hat{v}_i| \tag{3.4}$$

- **Root Mean Square Error (RMSE)**: RMSE calculates the square root of the average squared differences between the predicted and actual values. Compared to MAE, RMSE gives more weight to larger errors, making it more sensitive to outliers. It is a useful metric when large errors are particularly undesirable.

$$\mathrm{RMSE}(v, \hat{v}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (v_i - \hat{v}_i)^2} \tag{3.5}$$

- **Underestimate Penalising Error (UPE)**: UPE is a metric proposed in this work. While similar approaches have the same intention, UPE aims to penalise underestimations in the predictions. This metric is more practical, as the ultimate goal of a model like this would be to provide enough mobility for urban citizens. The penalty factor p has been set to $p = 3$.

$$\mathrm{UPE}(v, \hat{v}) = \frac{1}{N}\sum_{i=1}^{N} |v_i - \hat{v}_i| \times \begin{cases} p & \text{if } v_i > \hat{v}_i \\ 1 & \text{otherwise} \end{cases} \tag{3.6}$$

The MAPE (Mean Absolute Percentage Error) is also a common metric for regression problems, however, it falls short to its necessity of dividing by the ground truth. In the case of this study, a lot of true values are 0. A workaround would be to use a really small value and add it to 0, but then the MAPE grows by a lot.

Validation Procedure

After splitting the data and determining the metrics, the traditional model validation procedure was followed. Firstly, the model was trained on the training set, adjusting its parameters to learn the inherent patterns in the data. Following this, the validation set was used to fine-tune the model parameters and prevent overfitting.

Finally, the trained and optimised model was tested on the test set. This last step provided an unbiased evaluation of the final model, offering insights into how the model would likely perform on unseen real-world data. The MAE, RMSE, and UPE metrics were calculated on the test set to quantify the model's predictive performance.

The dataset partitioning and model validation were carried out in such a way that the model was prevented from 'seeing' future data during the training phase. This approach mimics real-world situations and helps to ensure that the model is not just fitting the data it has seen but is capable of making reliable predictions about future urban mobility demand.

3.9 Implementation Details

The implementation of the model and its associated processes is an essential part of this thesis, not only to ensure transparency and reproducibility, but also to provide insights into the challenges encountered and the solutions developed to overcome them. The implementation includes a series of steps ranging from data cleaning and preprocessing to model training and evaluation. In this section, the software, libraries, and hardware used in implementing the ST-GAT* model for predicting urban mobility demand is laid out.

Software and Libraries

Python was used exclusively throughout the project. The choice of Python was driven by its flexibility and the robustness of its scientific computing libraries, like Pandas, PyTorch and NumPy. Pandas, a software library for data manipulation and analysis, was used in the cleaning and preprocessing of the spatio-temporal data. Complementing it, NumPy adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

The development and implementation of the dataset, model, training, and evaluation were achieved using PyTorch and PyTorch Geometric (PyG). PyTorch is an open-source machine learning framework that helps researchers by providing a rich set of functionalities for tensor computation with GPU acceleration support. PyTorch Geometric, an extension library of PyTorch, was used to implement the Graph Neural Network model. It simplifies the process of implementing graph-based neural network architectures and datasets, providing efficient and easy-to-use tools for working with graph-structured data.

Environment and Hardware

The model was developed and run in a Python 3 environment in Google Colab. Google Colab is a cloud-based Python development environment that provides GPU support for machine learning and deep learning applications. It supports multiple popular libraries and frameworks, including PyTorch and PyTorch Geometric.

For the model training and evaluation, an NVIDIA Tesla T4 GPU with 16GB memory was used. This GPU is provided by Google Colab and is designed to accelerate deep learning workloads.

Challenges and Solutions

The primary challenge faced during the implementation was the processing of large amounts of data and managing the density of the graph. These factors occasionally led to memory issues, either during the data processing stage or when training the model. The high density of the graph and the large volume of data were memory-intensive, causing the system to use all the available GPU memory. However, as discussed in section 3.3 Data Processing, this issue was avoided by reducing the number of nodes in the graph drastically. This reduction allowed the system to handle the dataset and perform the training process without expanding the GPU memory.

3.10 Ethics and Data Privacy

As this study takes into account sensitive elements such as urban mobility data, it is important to address and discuss the considerations that have been made with regard to the ethics and data privacy aspects in the applied methodology.

Ethics

From the ethical standpoint, it needs to be noted that biases may exist in the data obtained from taxi and bicycle rides. These biases, which will not be discussed in this thesis, can potentially stem from a range of socioeconomic factors, including e.g. economic status, geographical preferences, age, and lifestyle habits of the individuals using these ways of transportation.

These biases are inevitable in any form of data collection process, as they reflect the realities of the environment where the data is collected. Moreover, it is important to note that the ST-GAT* model, which operates on the principle of learning from the training data, is therefore also likely to reflect these biases in its predictions. However, the aim is not to establish the functionality of GNNs solely on this particular dataset, but to demonstrate the efficiency of GNNs in handling any dataset, regardless of its inherent biases.

Therefore, while the probability of bias is noted, this methodology is still an effective approach. The adaptability of GNNs such as ST-GAT* makes sure that if there is a less biased or differently biased dataset available, the model can learn from and adapt to it, offering valuable and insightful results.

Data Privacy

Privacy concerns are important to consider, particularly when dealing with data that could potentially be traced back to individuals. However, in the case of this research, the data used has been anonymised beforehand, ensuring that no personal or private information can be traced back to the original data owners. This not only ensures the privacy of the individuals but also aligns with global data privacy regulations and standards.

Furthermore, the data is grouped and generalised during the process of building the ST-GAT* model. This level of aggregation ensures that even if an attempt is made to de-anonymise the data, it would be impossible to trace it back to any single individual.

Chapter 4

Results and Discussion

In this chapter, the results obtained from the application of the proposed Spatio-Temporal Graph Attention Network (ST-GAT*) model on the urban mobility dataset are presented and discussed. The model's effectiveness has been evaluated by investigating its predictive performance, comparing it with other baselines and similar models, and analysing the specifics of its predictions. Additionally, the model's performance across different time horizons was assessed to determine its efficacy in short-term and long-term demand prediction.

The robustness and sensitivity of the model were evaluated to measure its stability and response to changes in the dataset and hyper-parameters, respectively. The findings suggest that the ST-GAT* model shows significant promise in predicting urban mobility demand, as detailed in the following sections of this chapter. These findings, their implications, and their alignment with previous research in the field are further discussed to provide a comprehensive understanding of the potential of graph neural networks in predicting urban mobility demand.

4.1 Performance of the ST-GAT* Model

In this section, the two most efficient implementations of the ST-GAT*$_{4,4,4}$ model for the taxi and bicycle demand forecasting problem will be presented. Their performance, as measured by the metrics described in section 3.8, will be examined. In addition, an analysis and explanation of these metrics along with their configurations will be provided. Finally, a few exemplary graphs will showcase their performance.

Performance of ST-GAT* for taxi demand prediction

Three tables in the appendix .1 provide a detailed listing of all the model iterations that were performed to determine the optimal settings. One configuration that produced good results for the taxi driving data consisted of:

- **Learning Rate (LR)**: 0.0005
- **LR decay**: 0.00001
- **Optimiser**: Adam
- **Batch size**: 1024

- **# Epochs**: 60
- **Edge weights**: Correlation Matrix
- **# Edges per node**: 30

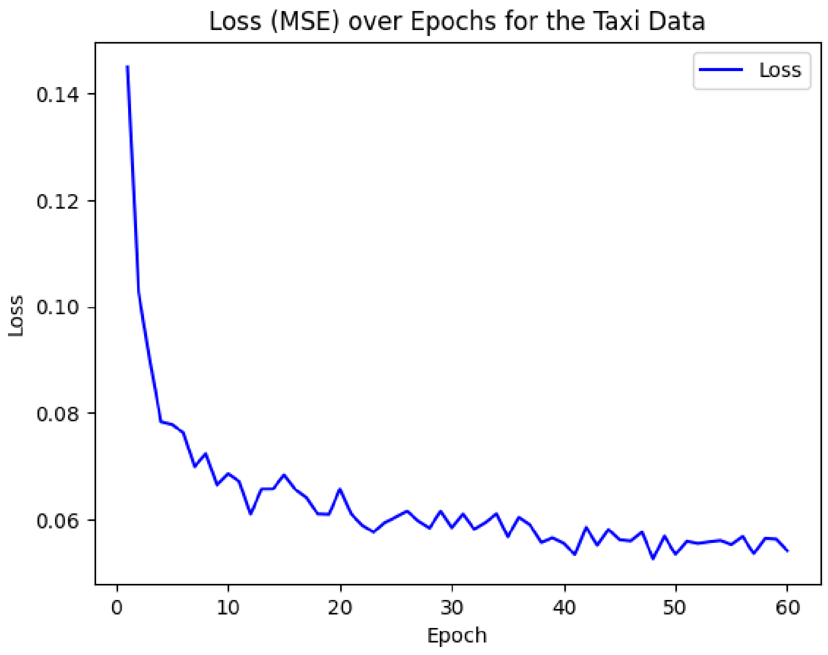


Figure 4.1: The MSE loss of the model on the taxi rides dataset, calculated in every epoch.

This particular arrangement of settings resulted in the loss function shown in Figure 4.1. Other configurations often resulted in an unstable loss function, a trend that was more noticeable for smaller batch sizes. The final model, when run on the test data set, produced the following values for the proposed metrics:

- **MAE**: 0.678
- **RMSE**: 2.237
- **UPE**: 1.213

However, these values are heavily biased due to cases where no demand for taxi services was observed. In such cases, the model usually produces a prediction close to zero, resulting in lower values for all three metrics. This presents a bias problem as a significant proportion of taxi centroids are characterised by periods of no demand. Figure 4.2 shows the predictions and actual demand one hour into the future for a random taxi centorid and for the centroid that has the 50 highest counts of taxi trips throughout the temporal coverage on June 25, 2019. The model under-performs in low demand scenarios as it is difficult to predict the demand for a taxi after extended periods of no demand.

Based on these observations, the metrics were changed to include Top50-RMSE, Top10-RMSE, Top5-RMSE, Top50-UPE, Top10-UPE and Top5-UPE. The 50 taxi centroids with the most taxi trips account for 90.04% of all taxi trips. The corresponding values are shown in the table below:

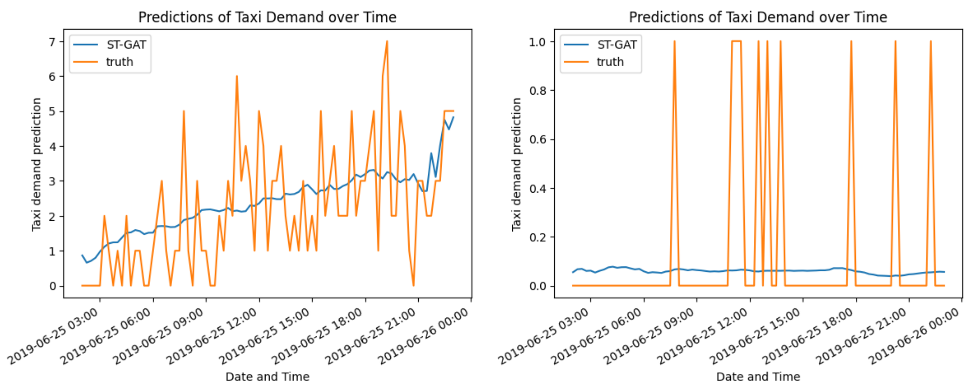


Figure 4.2: The predictions and ground truths for two different taxi centroids on June 25, 2019, one hour into the future.

	Top50	Top10	Top5
RMSE	3.799	8.311	10.44
UPE	5.844	12.321	14.963

These values are much larger, but more accurately reflect the predictions. Figure 4.3 shows four different snapshots of the same model for the same centroid and day. This specific centroid has the maximum taxi trips over the entire temporal coverage. The dates, 25 and 26 June 2019, were chosen arbitrarily, but to ensure that it was a day with relatively high taxi demand. As $n_y = 4$, the model predicted demand for four different time periods. The top two graphs show predictions for one hour in the future (first prediction, but $s = 4$), while the two lower diagrams show predictions for 01:45 hours in the future. The left-hand diagrams refer exclusively to 25 June, while the right-hand diagrams depict a period of two days. It can be observed that the predictions become less accurate the further the forecast period extends into the future. However, apart from a few outliers, the predictions are close to the actual demand.

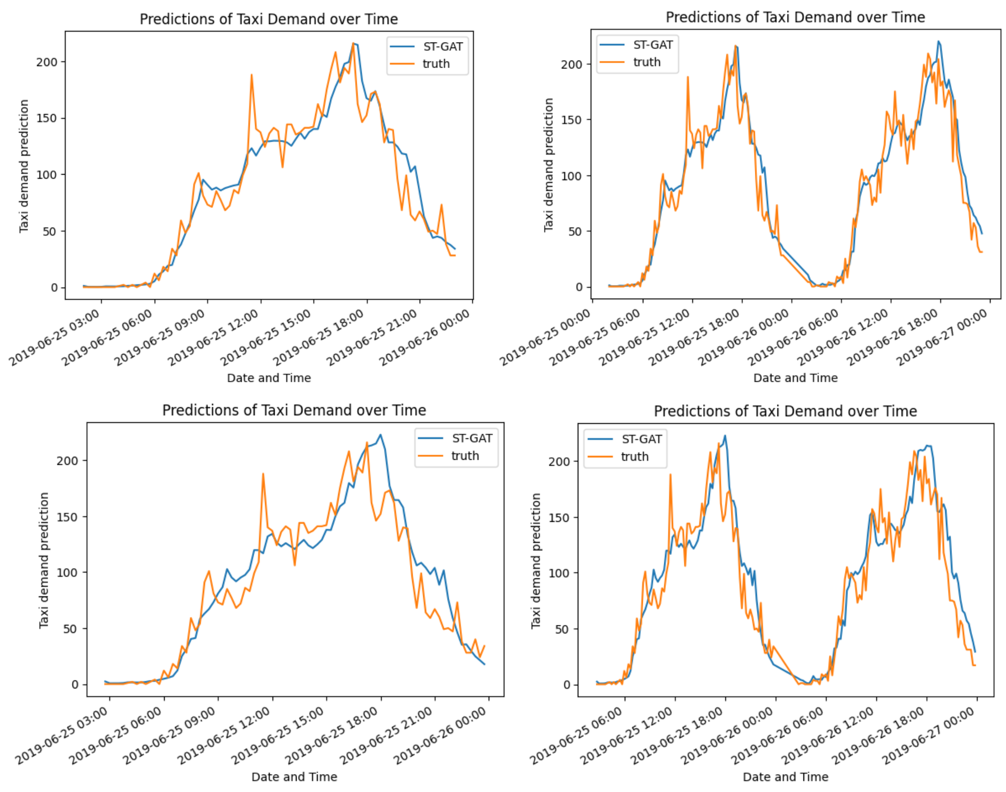


Figure 4.3: The predictions and ground truths of the taxi centroid with the most demand all-time, on June 25 and 26, 2019. The graphs on the left hand side cover June 25: the upper one, one hour into the future and the lower one, 01:45 hours. The graphs on the right hand side cover June 25 and 26: the upper one, one hour into the future and the lower one 01:45 hours.

Performance of ST-GAT* for bicycle demand prediction

As for the taxi rides demand, the three tables in the appendix .1 provide a comprehensive overview of the numerous model runs that were run to optimise the model parameters. An example configuration that was successful when applied to bicycle rental data is detailed below:

- **Learning Rate (LR)**: 0.0001
- **LR decay**: 0.00001
- **Optimizer**: RMSprop
- **Momentum**: 0.9
- **Batch size**: 4096
- **# Epochs**: 60
- **Edge weights**: Cosine Similarity Matrix
- **# Edges per node**: 50

This specific set of parameters resulted in the loss function as shown in Figure 4.4. It is worth noting that alternative configurations often resulted in a fluctuating loss

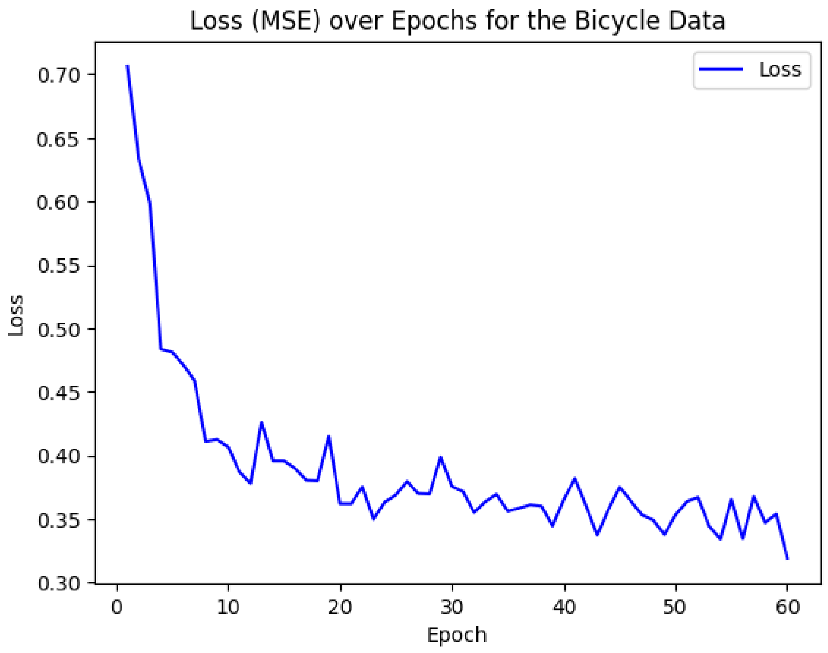


Figure 4.4: The MSE loss of the model on the bicycle dataset, calculated in every epoch.

function, especially for smaller batch sizes or a different optimiser. Once completed, the model was evaluated on the test data set, with the proposed metrics having the following values:

- **MAE**: 0.2493
- **RMSE**: 0.692
- **UPE**: 0.520

The UPE, a penalty metric for underestimated demand, appeared to be even lower than the RMSE. This effect may be due to the impact of instances where no demand for bicycles was recorded. This appeared even more often in the context of bicycle data. The mean ($\mu = 0.215$) and standard deviation ($\sigma = 0.998$) of demand across all stations and time periods suggest a general low demand for bicycles. Figure 4.5 visualises the predicted and actual demands one hour ahead for a random bicycle station, as well as for the top 50 station in terms of total bicycle rentals throughout the temporal coverage. For a station within the top 50, the demand is still low, despite this particular day being comparatively busy. The model's performance is worse when faced with low demand scenarios, as the prediction of bicycle demand becomes challenging in periods of low to no demand.

To address this issue, the metrics were changed to focus on high demand scenarios: Top50-RMSE, Top10-RMSE, Top5-RMSE, Top50-UPE, Top10-UPE, and Top5-UPE. The top 50 stations in terms of bicycle rentals account for 58.87% of all bicycle rentals. The new metrics yield the following values:

While these values are higher, they better reflect the predictive power of the model.

Figure 4.6 illustrates four different instances of model projections for the same cen-

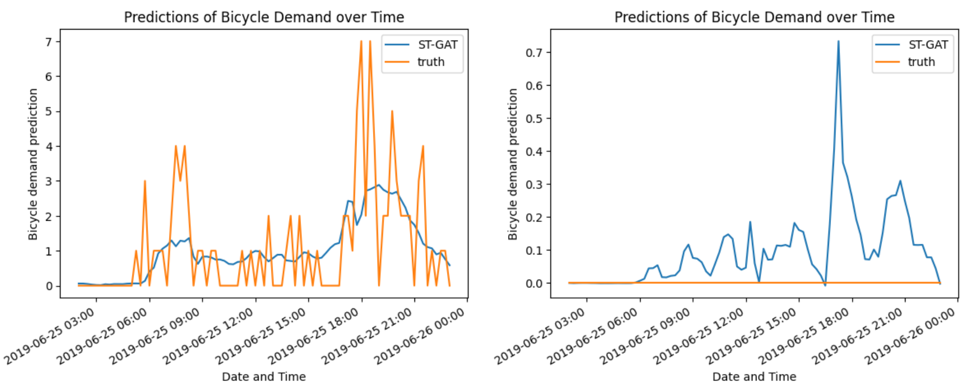


Figure 4.5: The predictions and ground truths for two different bicycle stations on June 25, 2019, one hour into the future.

	Top50	Top10	Top5
RMSE	1.651	2.939	3.288
UPE	1.759	2.975	3.287

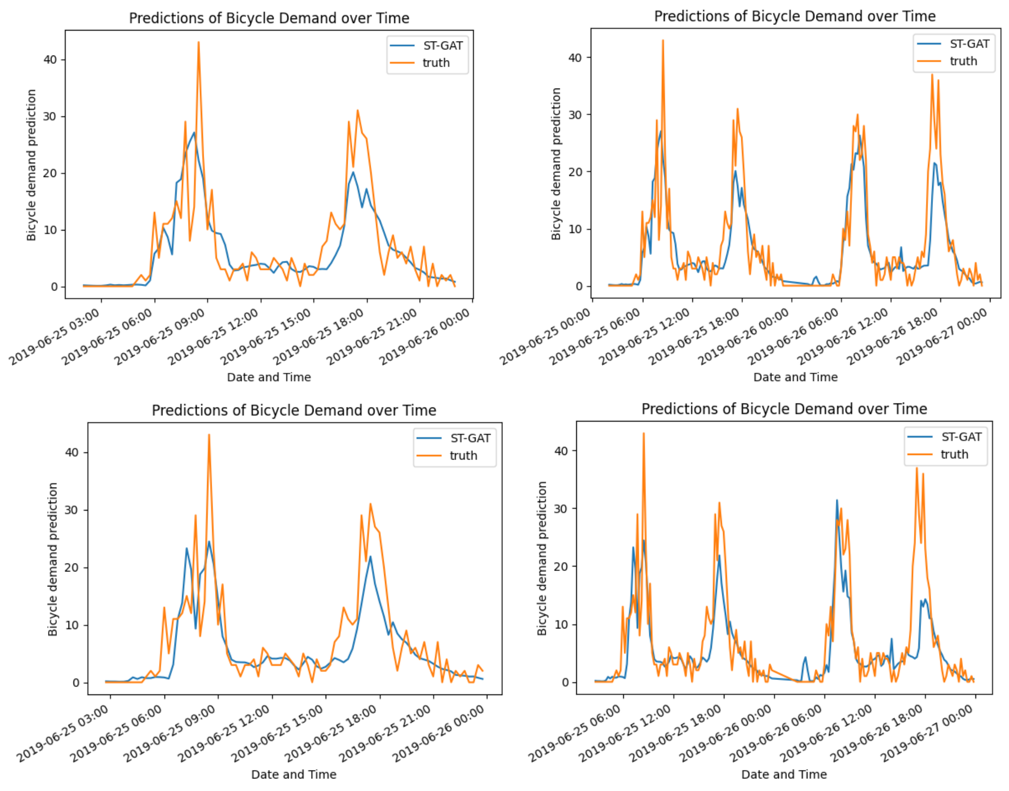


Figure 4.6: The predictions and ground truths for the bicycle station with the most demand all-time, on June 25 and 26, 2019. The graphs on the left hand side cover June 25: the upper one hour into the future and the lower one 01:45 hours. The graphs on the right hand side cover June 25 and 26: the upper one, one hour into the future and the lower one 01:45 hours.

troid and day. This centroid, identified as the one with the most bike rentals throughout the observation period, is shown for 25 and 26 June 2019. With $n_y = 4$, the model consistently predicts demand for four separate time intervals. The top two plots show forecasts for one hour ahead (the first forecast, but with $s = 4$), while the bottom two show forecasts for 1:45 hours ahead. The charts on the left are for 25 June, while the charts on the right cover two days. It is obvious that the accuracy of the model decreases as it predicts further into the future, but apart from a few outliers, the model catches the underlying patterns of higher or lower demands.

4.2 Examination of Predictions

This analysis includes various approaches for assessing prediction quality. The models explored here have been optimised to reduce the mean squared error of the prediction. However, for practical mobility considerations, predictions should ideally meet demand, which may be better represented by measures such as the Underestimation Penalty Error (UPE), penalising under-predictions.

The focus could either be on absolute or relative error when evaluating model performance. This raises an ethical question: is it more crucial to achieve maximum mobility, or optimise mobility relative to the area? This analysis has chosen the first aspect.

The ST-GAT*$_{4,4,4}$ model produces these predictions. Bias in error measures caused by zero-demand instances was corrected by considering only cases with demand equal to or greater than one. The errors were then categorised based on demand. Given the focus on absolute errors, it's clear that error increases with demand due to higher deviations and increased noise. Still, some visible patterns can explain certain model errors.

Performance of Taxi Demand

The UPE was evaluated across 11 distinct categories for the taxi rides data, as shown on the x-axis in the following graphs. Figure 4.7 reveals the distribution of all demand values in these categories on a logarithmic scale. This categorisation highlights differences in performance, which might get lost if the UPE was aggregated like in other parts of the results chapter.

Although the first category might appear most critical due to its high count, all categories are significant. For instance, a shortfall of 30 out of 120 taxi rides is a more considerable problem regarding mobility congestion than if one out of four people doesn't get a ride.

Examining the factors that could influence taxi demand prediction, holidays versus non-holidays and weekends versus weekdays, in figure 4.8 shows that the model generally performs better on non-weekend, non-holiday days. This might be due to the availability of more data on these days, which helps the model in learning and understanding patterns.

The substantial performance difference between weekdays and weekends when demand exceeds 80 people may be due to data scarcity. On weekdays, rush hours often lead to high demand, while weekends rarely have such high demand within 15-minute intervals, resulting in insufficient data for the model to predict effectively.

In lower demand categories, UPE values are quite close. In the first category, which constitutes most of the test, training, and validation datasets, UPE values are vir-

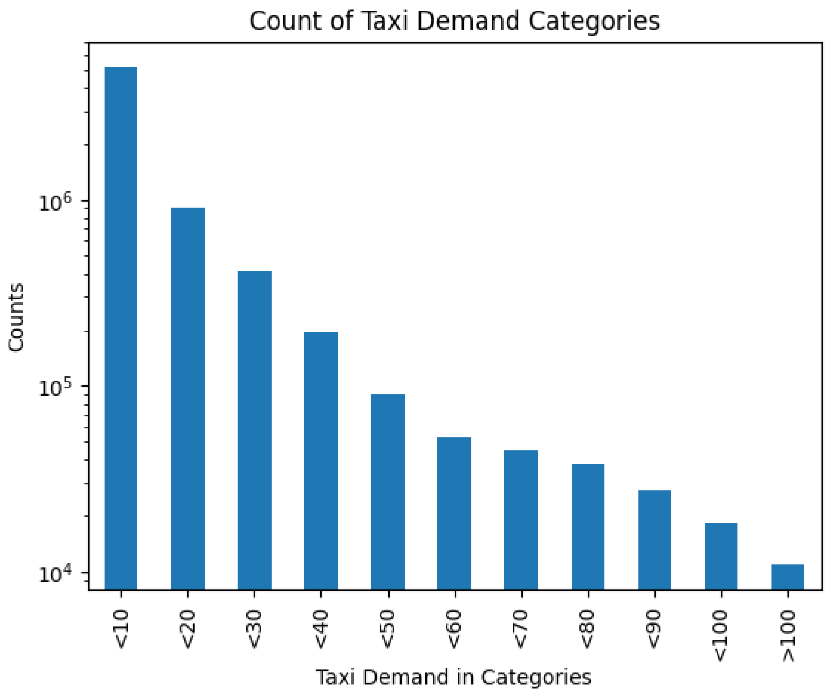


Figure 4.7: The distribution of all taxi rides demand values in 11 distinct categories on a logarithmic scale.

tually identical for weekends (3.127) and weekdays (3.133). For holidays in the first category, it's 3.106 and 3.132 for non-holidays.

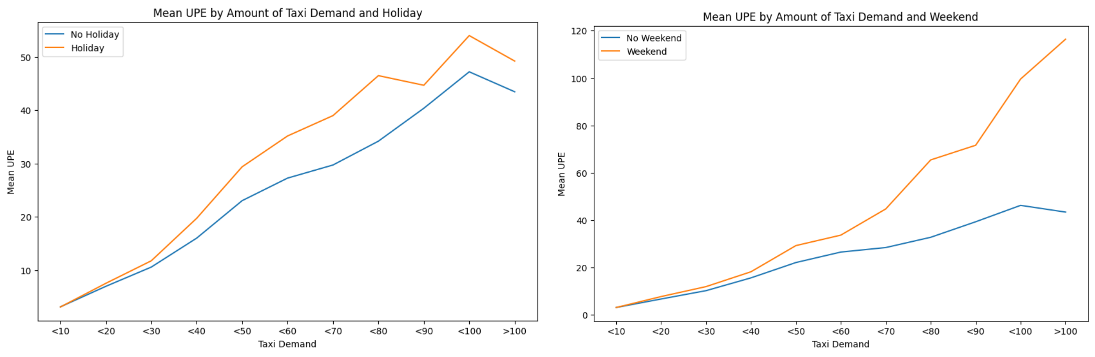


Figure 4.8: On the left: The mean UPE for holidays and no holidays in 11 distinct categories of taxi rides demand. On the right: The mean UPE for weekends and weekdays in 11 distinct categories of taxi rides demand.

Time-series data such as the time of the day or the season can also affect the model's performance. Figure 4.9 demonstrates how these factors impact the model's performance. The model generally performs worse during night hours and during winter and spring for higher demands, possibly due to higher standard deviations.

Weather conditions heavily influence taxi demand. Figure 4.10 demonstrates how these factors impact the model's performance across different demands. The model performs comparably across all weather conditions for the lowest category of demand, with a slightly better performance under snowy conditions.

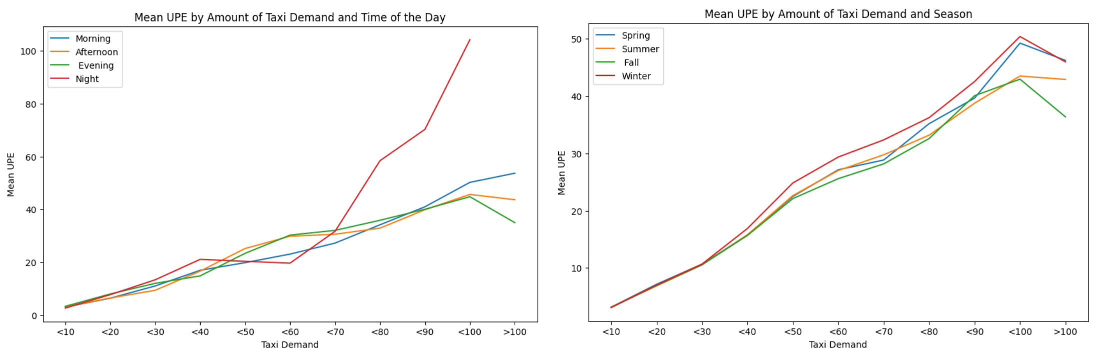


Figure 4.9: On the left: The mean UPE for different times of the day in 11 distinct categories of taxi rides demand. On the right: The mean UPE for different seasons of the year in 11 distinct categories of taxi rides demand.

In higher categories, the model performs worse when raining or snowing, likely due to insufficient data under these conditions. However, it seems the model incorporates various weather conditions effectively into its predictions, with no single weather condition standing out.

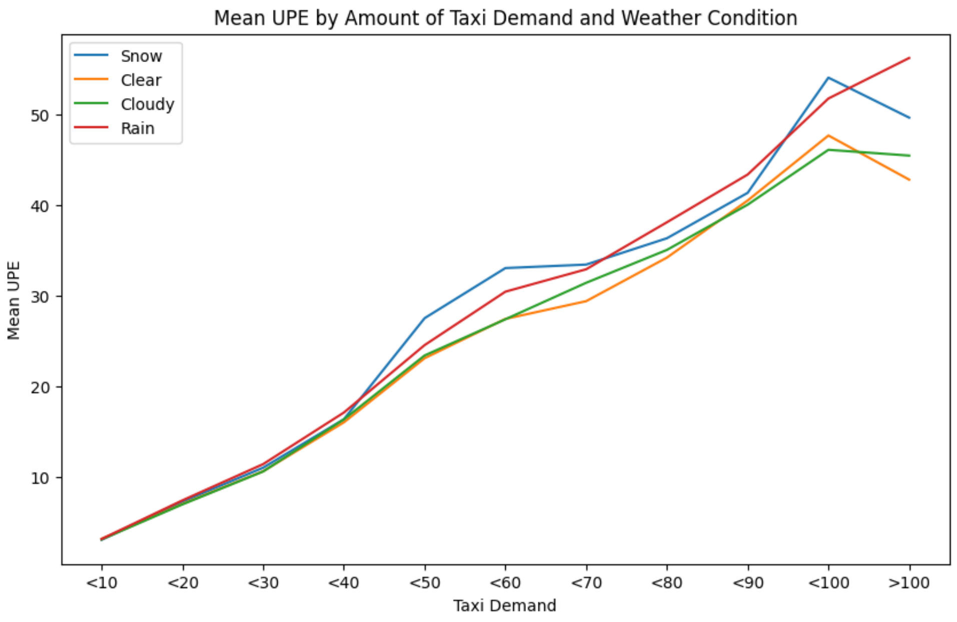


Figure 4.10: The mean UPE for different weather conditions in 11 distinct categories of taxi rides demand.

The model has also been trained on temperature and wind speed data, as changes in either could affect taxi demand. Figure 4.11 presents the performance across four different temperature and wind categories. It appears that the model performs better for higher demands under hotter conditions, but this could be due to a lack of data and therefore an overperformance bias. Wind speed doesn't seem to affect the model's performance unless it is gale-force, which is likely due to a lack of data under these conditions.

Performance of Bicycle Demand

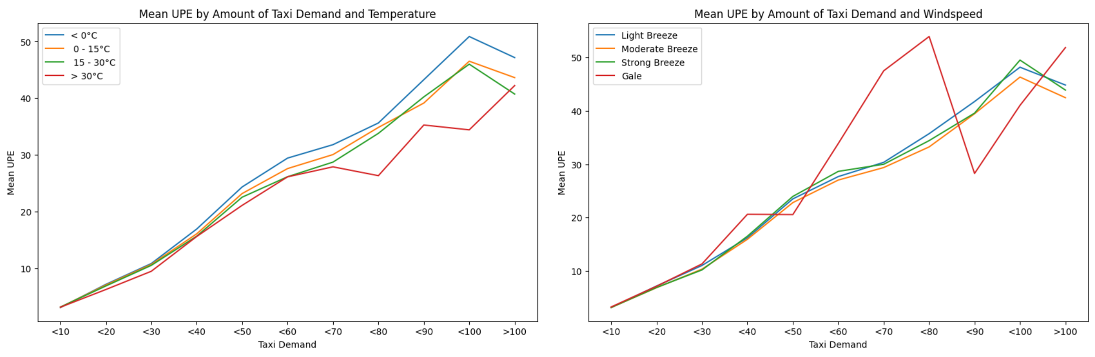


Figure 4.11: On the left: The mean UPE for different temperature categories in 11 distinct categories of taxi rides demand. On the right: The mean UPE for different wind categories in 11 distinct categories of taxi rides demand.

Similar to the taxi demand analysis, the performance of the ST-GAT* model has been evaluated for bicycle rental data using the mean UPE across 11 distinct categories of demand. As the demands for bicycles were significantly lower than for taxis, the categories differ. Figure 4.12 depicts the distribution of all predictions for these categories on a logarithmic scale.

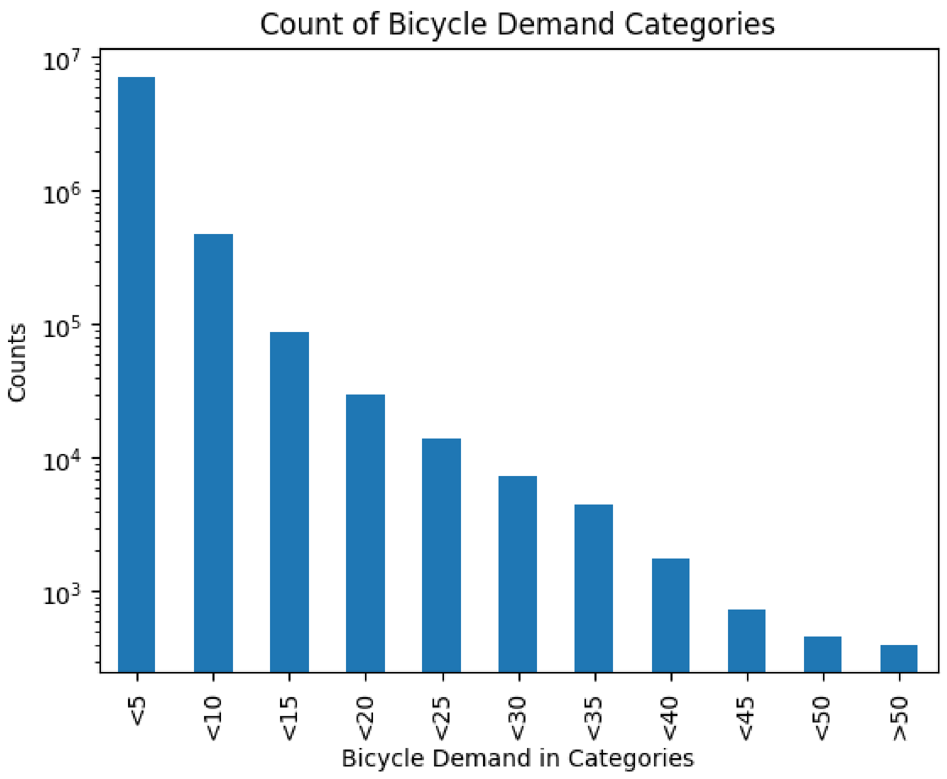


Figure 4.12: The distribution of all bicycle demand values in 11 distinct categories on a logarithmic scale.

Figure 4.13 displays the model's performance for holidays and weekends. In the lowest three categories of bicycle demand, the model's performance is quite similar across all days. In contrast, performance diverges for higher categories, likely due to data scarcity. The model performs better on weekends than weekdays for demands over 20, possibly due to a different standard deviation in demands.

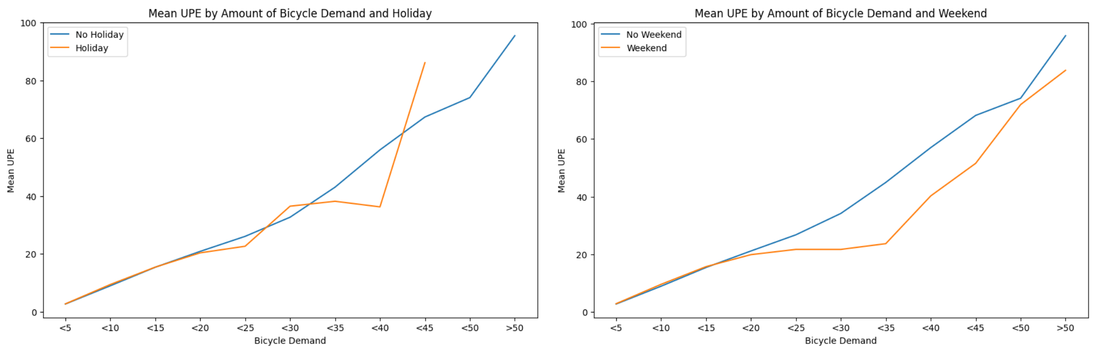


Figure 4.13: On the left: The mean UPE for holidays and no holidays in 11 distinct categories of bicycle demand. On the right: The mean UPE for weekends and week-days in 11 distinct categories of bicycle demand.

In terms of time of the day, as shown in figure 4.14, the model performs better in the morning and afternoon for all categories. However, a comparison is challenging due to a lack of data for night times.

The same issue applies to bicycle rentals in winter, as fewer people ride bikes. Otherwise, the model performs comparably across the remaining three seasons.

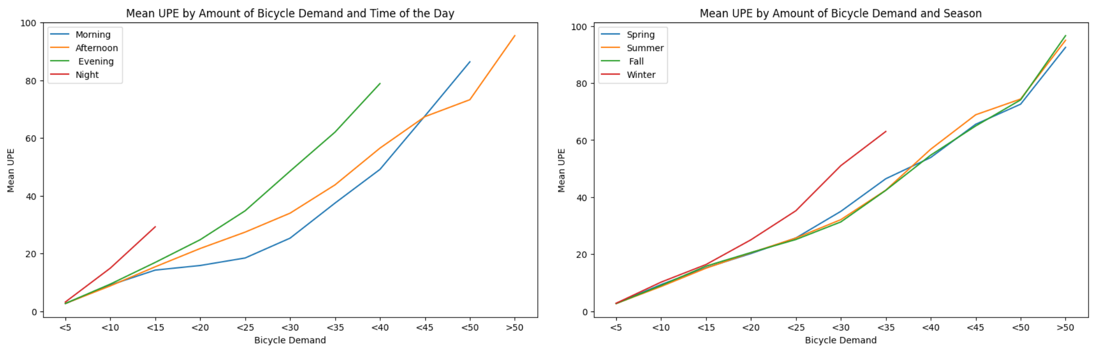


Figure 4.14: On the left: The mean UPE for different times of the day in 11 distinct categories of bicycle demand. On the right: The mean UPE for different seasons of the year in 11 distinct categories of bicycle demand.

Figure 4.15 presents the mean UPE values for different weather conditions, revealing that the model performs slightly better in non-raining conditions. In the case of snowfall, there's never been a demand over 20 at one station within 15 minutes.

Figure 4.16 illustrates the performance across different temperature and wind categories. The pattern continues: extreme weather conditions lead to low to no demand for bicycles. Therefore, there's minimal data, and the model doesn't perform well. As with taxi demand, the model performs better with high temperatures. Different wind-speed categories do not seem to influence the model's performance.

In conclusion, it's important to note that UPE values for bicycles reach much higher values than for taxis, particularly in categories with higher demands. This may be

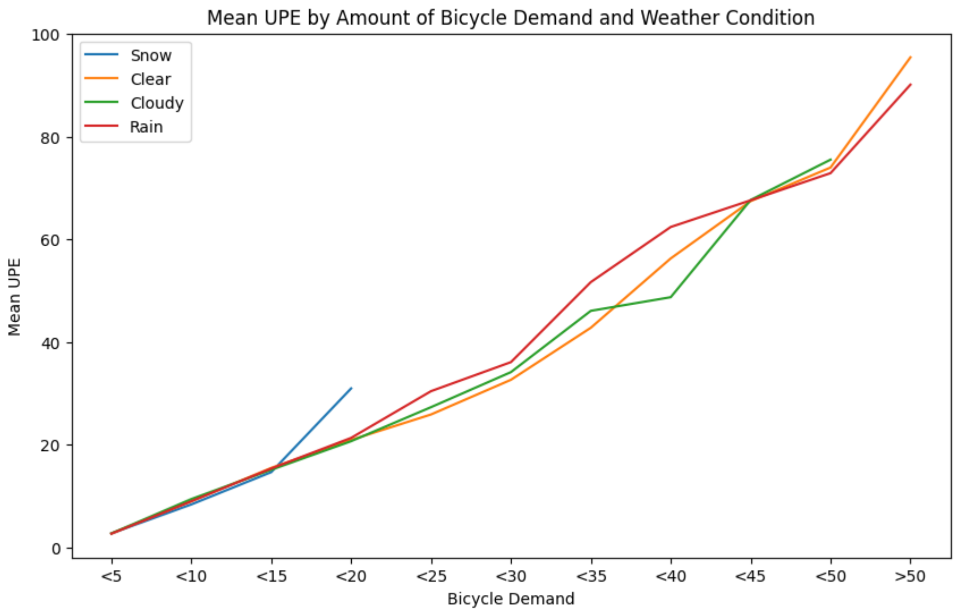
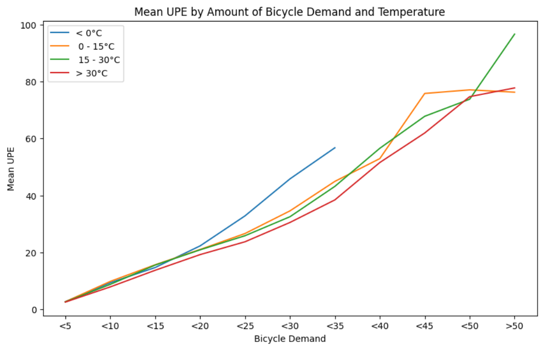


Figure 4.15: The mean UPE for different weather conditions in 11 distinct categories of bicycle demand.



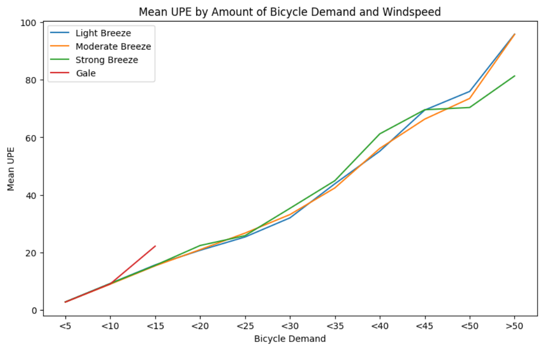


Figure 4.16: On the left: The mean UPE for different temperature categories in 11 distinct categories of bicycle demand. On the right: The mean UPE for different wind categories in 11 distinct categories of bicycle demand.

due to the predominance of zero-demand instances in the training data set, biasing the model towards higher accuracy for lower demands. This is a significant concern, as it may improve desirable metrics but lead to bad results for higher demands.

4.3 Input Feature Comparison

To discover the performance of the ST-GAT* model relative to the ST-GAT model with specific datasets, an in-depth analysis of the results generated by each model is necessary. As in previous sections, the input, shift, and output configurations are set to 4. All models underwent identical hyper-parameter training over a span of 10 epochs.

A comparison was made between the ST-GAT* model and the ST-GAT model, the latter lacking both time-series and weather modules. Additionally, the ST-GAT* model was compared to a variant with only the time-series module, denoted as ST-GAT$_t$, and another variant incorporating just the weather module, named ST-GAT$_w$.

As indicated in table 4.1, both the Root Mean Square Error (RMSE) and UPE scores for each mobility module and model were documented. Values for the Top 50, 10, and 5 stations were included to avoid potential bias from stations that consistently have low demand.

In the context of taxi ride demand predictions, the ST-GAT* model outperforms other model versions on the Top 50, Top 10, and Top 5 UPE metrics. There are no notable improvements besides these. However, the Top 5 UPE score is interesting as it shows the model's potential to tackle high-demand problems.

In contrast, the ST-GAT* model does not perform better than other models when predicting bicycle rental demand; it actually underperforms based on the metrics presented. This could suggest the need for a different training approach to achieve better results, or it could indicate that this model may not be ideally suited to this specific dataset.

#	Mobility Module	Model	RMSE	Top50 RMSE	Top10 RMSE	Top 5 RMSE	UPE	Top50 UPE	Top10 UPE	Top5 UPE
1	Taxi	ST-GAT*	2.237	3.799	8.311	10.44	1.213	5.844	12.321	14.963
2		ST-GAT$_w$	2.274	3.953	8.62	10.607	1.246	6.311	13.768	17.05
3		ST-GAT$_t$	2.266	3.953	8.56	10.583	1.242	6.54	13.953	17.587
4		ST-GAT	2.36	4.062	8.872	11.017	1.37	6.529	14.36	18.614
5	Bicycle	ST-GAT*	0.692	1.651	2.939	3.288	0.52	1.759	2.975	3.287
6		ST-GAT$_w$	0.685	1.645	2.903	3.219	0.529	1.902	3.19	3.599
7		ST-GAT$_t$	0.682	1.641	2.9	3.205	0.53	1.693	2.882	3.2
8		ST-GAT	0.681	1.637	2.875	3.19	0.535	1.753	2.881	3.181

Table 4.1: An overview of different evaluation metrics for different variations of the ST-GAT model and for different mobility modules.

4.4 Graph Structure Comparison

The construction of the graphs used three different methods, as presented in section 3.4. The application of the Gaussian Kernel, with $\sigma = 0.1$ and $\epsilon = 0.9$, created connections between all nodes within approximately a one kilometer radius. This was only done for the taxi data, given that the bicycle data had been preprocessed at the point of download, and no coordinates were available.

Table 4.2 demonstrates the ST-GAT*$_{4,4,4}$ model's performance across various graph structures. The evaluation metrics used were RMSE, Top50 RMSE, UPE, and Top50 UPE. Yet, the table suggests no discernible impact of the edges per node quantity or the way of their construction on the RMSE or UPE. The Gaussian Kernel shows a slightly superior performance over the Cosine Similarity, though the difference is not substantial.

Similar findings are observable for the bicycle data in table 4.3. The selected measurements do not show a performance differences. This could mean that the GAT-Conv layer of the ST-GAT* model may not significantly influence the performance, or alternately, other model components could compensate for variations in these set-

Edgetype	# Edges per Node	RMSE	Top50 RMSE	UPE	Top50 UPE
Correlation	10	2.195	3.808	1.211	6.045
	30	2.17	3.779	1.245	5.831
	50	2.194	3.815	1.215	6.128
Cosine Similarity	10	2.243	3.924	1.201	6.659
	30	2.212	3.841	1.247	5.989
	50	2.24	3.881	1.235	6.163
Gaussian Kernel	$\sigma = 0.1$, $\epsilon = 0.9$	2.193	3.802	1.281	5.882

Table 4.2: The performance of different graphs, constructed with the same set of nodes but different edges and quantity of edges for the taxi rides dataset.

tings. Another possibility might be that neither choice is optimal for the graph structure. Additionally, these results could be due to the conformity in training duration (10 epochs) and hyper-parameters across all approaches.

Edgetype	# Edges per Node	RMSE	Top50 RMSE	UPE	Top50 UPE
Correlation	10	0.683	1.661	0.526	1.832
	30	0.68	1.645	0.524	1.784
	50	0.688	1.673	0.52	1.855
Cosine Similarity	10	0.68	1.653	0.523	1.928
	30	0.677	1.643	0.524	1.797
	50	0.682	1.654	0.525	1.77

Table 4.3: The performance of different graphs, constructed with the same set of nodes but different edges and amount of edges for the bicycle dataset.

4.5 Baseline Model Comparison

This segment evaluates the effectiveness of the Spatio-Temporal Graph Attention Network (ST-GAT*) model in forecasting urban mobility demand. The comparison is drawn with a few benchmark models. These models have been chosen due to their wide application in demand prediction and their inherent capability to handle the time-sensitive nature of the data. The decision to use these models was influenced by Prado-Rujas et al. [24]. The selected benchmark models include Long Short-Term Memory (LSTM), Bidirectional LSTM (BiLSTM), in addition to a persistence and a naive approach.

The choice to limit the selection to these models also came from practical considerations. Implementing more complex or innovative models, as suggested in recent studies, can be difficult due to the absence of easily accessible public code. As a result, the focus was placed on LSTM, BiLSTM, a persistence, and a naive approach.

For the LSTM and BiLSTM, the architecture of the ST-GAT* model, including the weather and time-series module, was used. Therefore, only the ST-GAT module was swapped out. The LSTM had a hidden layer of size 16, whereas the BiLSTM had a bidirectional layer of size 8.

The persistence approach assumes the demand from the last recorded timestamp of the input continues for the n_y predictions. The naive approach calculates the mean of the n_x input values and uses that value for the n_y predictions.

Taxi Trips Prediction

For the taxi data, five different temporal settings were tested. Hyper-parameters such as learning rate, optimiser, or batch size were consistent for the ST-GAT*, LSTM, and BiLSTM model, and were not optimised for any model. Again, the issues inherent in the metrics were addressed by showing only the RMSE and UPE values of the top 50 taxi centroids with the highest number of taxi rides of all time. The RMSE and UPE were then calculated for each of the n_y timesteps to achieve a more detailed analysis. RMSE values for all centroids are presented in the appendix, in table .2.

Table 4.5 portrays the RMSE for the top 50 taxi centroids for the various models and temporal settings. The ST-GAT* model consistently outperforms the other methods. Although this might suggest that the predictions are more precise, it could also be attributed to the nature of the RMSE. Since there is often no demand or very little, a model that closely approximates zero in such cases performs better with a larger data set. Although there are less no-demand cases in the top 50 stations, it still represents a large proportion. However, initial learning on the graph structure appears to lead to better predictions.

#	Model	n_x, s, n_y	Training	RMSE Top50 Taxi Centroids							
			time	t1	t2	t3	t4	t5	t6	t7	t8
1	ST-GAT*		10:05	3.659	3.753	3.832	4.001				
2	LSTM		7:22	3.988	4.090	4.200	4.303				
3	BiLSTM	4, 4, 4	7:51	4.131	4.199	4.287	4.422				
4	Persistence		-	5.209	5.628	5.998	6.321				
5	Naive		-	5.178	5.572	5.941	6.284				
6	ST-GAT*		13:37	3.724	3.827	3.932	4.023	4.088	4.164	4.249	4.325
7	LSTM		6:56	3.973	4.042	4.092	4.142	4.224	4.307	4.401	4.501
8	BiLSTM	4, 4, 8	7:12	3.988	4.074	4.15	4.223	4.324	4.447	4.577	4.717
9	Persistence		-	5.197	5.613	5.981	6.303	6.662	6.989	7.276	7.532
10	Naive		-	5.16	5.55	5.917	6.26	6.58	6.879	7.165	7.438
11	ST-GAT*		13:50	3.71	3.795	3.871	3.964				
12	LSTM		9:02	3.9	3.966	4.036	4.122				
13	BiLSTM	8, 4, 4	9:26	4.021	4.102	4.192	4.296				
14	Persistence		-	5.269	5.692	6.066	6.393				
15	Naive		-	5.721	6.074	6.408	6.724				
16	ST-GAT*		14:02	3.816	3.935	4.049	4.113	4.205	4.298	4.351	4.394
17	LSTM		8:55	4.167	4.212	4.252	4.317	4.405	4.493	4.583	4.721
18	BiLSTM	8, 4, 8	9:14	3.995	4.066	4.124	4.181	4.264	4.341	4.432	4.536
19	Persistence		-	5.259	5.679	6.05	6.376	6.739	7.068	7.358	7.617
20	Naive		-	5.687	6.035	6.364	6.677	6.973	7.25	7.521	7.783
21	ST-GAT*		13:45	4.151	4.219	4.272	4.334	4.446	4.537	4.611	4.688
22	LSTM		8:40	4.48	4.518	4.543	4.614	4.711	4.782	4.884	5.001
23	BiLSTM	8, 8, 8	8:37	4.401	4.441	4.491	4.528	4.6	4.695	4.798	4.892
24	Persistence		-	6.719	7.043	7.329	7.584	7.862	8.127	8.372	8.594
25	Naive		-	6.9	7.168	7.432	7.689	7.938	8.178	8.418	8.657

Table 4.4: The training time and RMSE Top50 calculated for all n_y predictions for different models and temporal settings of the taxi rides data.

The UPE penalises prediction underestimations by a factor of three, making it a more practical method for mobility planning, if the goal is to maximise mobility in a city. Table 4.5 presents the UPE for the top 50 taxi centroids. It is apparent that the ST-GAT* model performs significantly better than the other baselines, though it is also clear that the UPE increases the further into the future the prediction extends.

In general, examining these two metrics shows that the ST-GAT* model outperforms the baselines in predicting taxi ride demand.

#	Model	n_x, s, n_y	Training time	UPE Top50 Taxi Centroids							
				t1	t2	t3	t4	t5	t6	t7	t8
1	ST-GAT*		10:05	5.658	5.874	6.013	6.325				
2	LSTM		7:22	6.426	6.757	7.084	7.343				
3	BiLSTM	4, 4, 4	7:51	7.241	7.412	7.643	8.042				
4	Persistence		-	6.808	7.346	7.838	8.269				
5	Naive		-	6.735	7.245	7.747	8.231				
6	ST-GAT*		13:37	5.698	5.896	6.158	6.291	6.541	6.762	7.067	7.301
7	LSTM		6:56	6.404	6.561	6.638	6.819	7.111	7.401	7.734	8.057
8	BiLSTM	4, 4, 8	7:12	6.502	6.625	6.809	7.051	7.358	7.714	8.107	8.504
9	Persistence		-	6.689	7.199	7.667	8.075	8.572	9.029	9.446	9.821
10	Naive		-	6.576	7.056	7.541	7.99	8.431	8.851	9.272	9.691
11	ST-GAT*		13:50	5.88	6.034	6.136	6.373				
12	LSTM		9:02	5.884	6.015	6.157	6.361				
13	BiLSTM	8, 4, 4	9:26	6.623	6.81	7.052	7.319				
14	Persistence		-	6.909	7.453	7.951	8.388				
15	Naive		-	7.378	7.85	8.32	8.779				
16	ST-GAT*		14:02	6.438	6.692	6.932	7.021	7.224	7.427	7.509	7.528
17	LSTM		8:55	7.284	7.329	7.408	7.604	7.859	8.082	8.294	8.669
18	BiLSTM	8, 4, 8	9:14	6.824	6.981	7.106	7.264	7.461	7.564	7.773	7.992
19	Persistence		-	6.788	7.305	7.777	8.19	8.696	9.159	9.585	9.966
20	Naive		-	7.124	7.561	8	8.431	8.85	9.256	9.673	10.089
21	ST-GAT*		13:45	7.175	7.335	7.436	7.569	7.898	8.11	8.271	8.434
22	LSTM		8:40	7.883	7.969	7.96	8.176	8.474	8.631	8.94	9.309
23	BiLSTM	8, 8, 8	8:37	7.657	7.787	7.948	8.064	8.256	8.517	8.757	8.956
24	Persistence		-	8.473	8.905	9.301	9.651	10.068	10.479	10.868	11.226
25	Naive		-	8.386	8.752	9.135	9.52	9.903	10.282	10.678	11.075

Table 4.5: The training time and UPE Top50 calculated for all n_y predictions for different models and temporal settings of the taxi rides data.

Bicycle Rides Prediction

For the taxi data, each model was trained with the same unoptimised hyper-parameters for 10 epochs to ensure a valid comparison. As the demand for bicycles is often even closer to zero than for taxi rides, again only the top 50 stations with the most overall demand were evaluated. The RMSE and UPE values for all stations are available in the appendix, in table.2.

For the Top50 RMSE, table 4.6 displays how each model performed with different temporal input settings. Inputs and predictions are either one or two hours long, and the gap between them is also either one or two hours. Due to computational memory limitations when creating the dataset, the ST-GAT*$_{8,4,8}$ model, which was featured for taxi rides, could not be created.

Unlike the taxi data, the model does not show significantly better results than the other baselines here. Especially for long prediction horizons, it performs worse than the model with the LSTM or BiLSTM layer. As suggested in previous sections, this could be due to this dataset having low values and the RMSE being lower for such values. If a model underestimates the demands, it will generally score better for the RMSE with such low values as zero is the lower limit.

The UPE addresses this issue. However, the model performs comparably to the other two models, shown in table 4.5. In general, the three models outperform the persistence and naive approaches. This result could mean that for bicycle demand, the spatial dependencies are not as crucial as for taxi rides. The training time of the

#	Model	n_x, s, n_y	Training time	RMSE Top50 Bike Stations							
				t1	t2	t3	t4	t5	t6	t7	t8
1	ST-GAT*	4, 4, 4	13:09	1.628	1.654	1.678	1.701				
2	LSTM		6:05	1.623	1.647	1.677	1.702				
3	BiLSTM		6:11	1.637	1.665	1.698	1.728				
4	Persistence		-	2.431	2.546	2.642	2.722				
5	Naive		-	2.277	2.377	2.462	2.534				
6	ST-GAT*	4, 4, 8	12:29	1.623	1.656	1.695	1.729	1.768	1.798	1.823	1.847
7	LSTM		6:27	1.622	1.636	1.658	1.68	1.703	1.722	1.748	1.775
8	BiLSTM		6:12	1.639	1.659	1.688	1.717	1.747	1.771	1.797	1.822
9	Persistence		-	2.473	2.591	2.69	2.773	2.847	2.91	2.961	3.001
10	Naive		-	2.319	2.422	2.51	2.584	2.647	2.7	2.744	2.782
11	ST-GAT*	8, 4, 4	16:20	1.67	1.696	1.717	1.732				
12	LSTM		6:40	1.676	1.696	1.723	1.747				
13	BiLSTM		6:31	1.674	1.7	1.728	1.751				
14	Persistence		-	2.491	2.609	2.707	2.79				
15	Naive		-	2.373	2.453	2.521	2.578				
16	ST-GAT*	8, 8, 8	15:48	1.787	1.797	1.818	1.84	1.862	1.884	1.901	1.912
17	LSTM		6:04	1.812	1.817	1.833	1.847	1.86	1.868	1.871	1.878
18	BiLSTM		5:58	1.829	1.841	1.86	1.88	1.896	1.903	1.904	1.907
19	Persistence		-	2.963	3.029	3.083	3.127	3.166	3.199	3.227	3.253
20	Naive		-	2.712	2.757	2.796	2.831	2.863	2.894	2.924	2.953

Table 4.6: The training time and RMSE Top50 calculated for all n_y predictions for different models and temporal settings of the bicycle data.

ST-GAT* model is much longer than for the LSTM or BiLSTM, which could also be an important factor when developing a prediction model.

#	Model	n_x, s, n_y	Training time	UPE Top50 Bike Stations							
				t1	t2	t3	t4	t5	t6	t7	t8
1	ST-GAT*	4, 4, 4	13:09	1.857	1.883	1.879	1.887				
2	LSTM		6:05	1.784	1.826	1.852	1.873				
3	BiLSTM		6:11	1.789	1.802	1.828	1.851				
4	Persistence		-	2.514	2.645	2.761	2.867				
5	Naive		-	2.446	2.568	2.679	2.776				
6	ST-GAT*	4, 4, 8	12:29	1.699	1.725	1.739	1.76	1.754	1.77	1.801	1.815
7	LSTM		6:27	1.763	1.778	1.777	1.781	1.802	1.826	1.856	1.868
8	BiLSTM		6:12	1.849	1.854	1.875	1.893	1.917	1.931	1.941	1.953
9	Persistence		-	2.567	2.704	2.824	2.935	3.035	3.122	3.199	3.263
10	Naive		-	2.492	2.618	2.732	2.833	2.921	2.999	3.069	3.131
11	ST-GAT*	8, 4, 4	16:20	1.929	1.945	1.968	1.938				
12	LSTM		6:40	1.811	1.854	1.899	1.92				
13	BiLSTM		6:31	1.821	1.82	1.841	1.863				
14	Persistence		-	2.629	2.767	2.89	3.001				
15	Naive		-	2.653	2.762	2.857	2.942				
16	ST-GAT*	8, 8, 8	15:48	2.01	1.991	1.983	1.964	1.958	1.965	1.958	1.963
17	LSTM		6:04	1.949	1.945	1.956	1.971	1.995	2.017	2.03	2.035
18	BiLSTM		5:58	1.969	1.964	1.98	1.976	1.987	1.994	2.001	1.996
19	Persistence		-	3.221	3.315	3.398	3.467	3.535	3.591	3.641	3.668
20	Naive		-	3.036	3.106	3.169	3.228	3.284	3.339	3.391	3.442

Table 4.7: The training time and UPE Top50 calculated for all n_y predictions for different models and temporal settings of the bicycle data.

4.6 Comparison with Previous Research

Prado-Rujas et al.[24] explain in their research that it's challenging to compare the outcomes of a model like the one under review to similar research that use different data or models. This problem occurs mainly when using metrics such as the Root Mean Square Error (RMSE) for comparison, underlining the crucial role that data plays in the outcome. For instance, a very fine temporal granularity might result in smaller values and predictions, consequently leading to a lower RMSE.

This section dives into the comparison of the RMSE of the predictions from the ST-GAT* model against other notable works involving taxi data. It's important to highlight that the STGCN, DCRNN, LSTM, ARIMA, and HA models were all formulated by Liu et al. [17] for data belonging to the cities of New York and Chengdu.

- ST-MDF* by Prado-Rujas et al.[24]: The same architecture as ST-GAT* but a Conv-LSTM module instead of the ST-GAT module
- CACRNN by Liu et al. [17]: An attention-based convolutional recurrent neural network
- STGCN by Han et al. [10]: A spatial–temporal graph convolutional network
- DCRNN by Li et al. [16]: A diffusion convolutional recurrent neural network
- LSTM: The long short term memory model
- ARIMA: The autoregressive integrated moving average
- HA: The historical average

Table 4.8 shows the RMSE for different models that used distinct datasets and forecast horizons. Although such comparison may not effectively portray the superiority of one method over another, it conclusively reveals that the ST-GAT* model holds its ground against other published models. An accurate comparison with the ST-MDF* model is not possible due to the latter's use of all taxi centroids while ST-GAT* only uses 30% after removing 70%.

#	Model	Evaluated service	Forecast horizon	City	RMSE
1	ST-GAT*	Taxi rides	1 h	Chicago	2.24
2	ST-MDF*				2.43
3	CACRNN	Taxi rides	15 min	NYC	3.17
4	STGCN				3.48
5	DCRNN				3.50
6	LSTM				3.65
7	ARIMA				9.53
8	HA				5.84
9	CACRNN	Taxi rides	15 min	Chengdu	2.73
10	STGCN				2.77
11	DCRNN				3.14
12	LSTM				4.30
13	ARIMA				5.75
14	HA				7.10

Table 4.8: An overview of the RMSE values of different models which predict the demand of taxi rides.

In a similar way, the prediction of bicycle rental demand has been explored in multiple research studies. Several models will be compared with the ST-GAT* model using the bicycle rental dataset.

- ST-MDF* by Prado-Rujas et al.[24]: The same architecture as ST-GAT* but a Conv-LSTM module instead of the ST-GAT module
- Multi-graph CNN by Chai et al. [1]: A multi graph convolutional neural network
- LSTM: The long short term memory model
- GBRT : A gradient boosting regression tree
- SARIMA: A seasonal ARIMA
- ARIMA: The autoregressive integrated moving average
- HM: The historical mean

The models in the list after the Multi-graph CNN were also implemented and tested by Chai et al. [1]. The forecast horizon is the same for each of the models: one hour of input data predicts one hour of output demand, one hour into the future ($n_x = 4$, $s = 4$, and $n_y = 4$). The dataset is the same, but besides the ST-MDF* model, they only cover the period from 2013 to the end of 2017. Still these results are more comparable than for the taxi data. For the ST-GAT* model the amount of bicycle stations considered was reduced by 50%, but the top 5 and top 10 stations are the same.

Table 4.9 presents the RMSE for the predictions of the bicycle data and includes the RMSE for the top 5 and 10 stations concerning all-time demand. Notably, the ST-GAT* model has the lowest RMSE for all three measures by a considerable margin. This could be attributed to several factors but fundamentally, it suggests that the ST-GAT* model's performance is at least on par with the other models.

#	Model	Evaluated service	RMSE		
			Top 5 stations	Top 10 stations	Average
1	ST-GAT *	Bike trips	3.288	2.939	0.692
2	ST-MDF *		5.331	4.646	1.702
3	Multi-graph CNN		5.177	4.930	3.658
4	LSTM		6.231	5.853	4.405
5	GBRT		5.945	5.738	4.410
6	SARIMA		6.797	6.175	4.608
7	ARIMA		9.853	8.535	6.163
8	HM		7.078	6.179	4.347

Table 4.9: An overview of the RMSE values of different models which predict the demand of rental bicycles.

4.7 Robustness Analysis

Assessing the robustness of a model is crucial, particularly as training happens in a controlled environment with preprocessed offline data. When predicting urban mobility demand for the immediate future, such as the next one or two hours, it is expected that the model will successfully handle live data. Live data, however, can be challenging due to missing information, incorrect formatting, or skewed data sets. Therefore, evaluating the model's performance under stress, mimicking real-life situations, is both interesting and crucial.

For Graph Neural Networks (GNNs), the robustness of the model can be assessed in two ways: either by eliminating all the edges of a certain number of nodes in a graph, thereby excluding them from the GATConv layer computation, or by removing a certain amount of input data during prediction testing. The latter approach was selected for testing the resilience of the ST-GAT* model.

A random selection of inputs was replaced with the average of all inputs to simulate missing data. However, to realistically represent missing data, the model would need to be modified to accommodate varying input sizes. The model is currently incapable of handling NaN or Null values, and extreme high or low values are misinterpreted because they are not part of the training set. Another possible approach would be to replace missing values with the last recorded value at that node, although this becomes problematic if all inputs are missing or the first one of a sequence is missing. Consequently, the decision was made to use the average of all nodes for comparison.

Table 4.10 demonstrates how the RMSE changes when certain amounts of input data are missing and replaced by the average. The model performs well when small amounts of data are missing (up to 10%), but the RMSE rises significantly as more data is replaced, particularly for taxi rides. The difference between the two mobility modules can be attributed to the standard deviation. The standard deviation for taxi rides is 10.5, while that of bicycle rides is only 1.0. Replacing 50% of the values with the mean results in large RMSEs at those taxi centroids with high demand. Despite this, the model should recognise that these values are unrealistic. The model's performance noticeably worsens for long-term predictions, regardless of the quantity of missing input.

#	Model	Mobility	n_x, s, n_y	RMSE worsening [%] when missing				
				10%	20%	30%	40%	50%
1	ST-GAT*	Taxi	4, 4, 4	6.0	12.6	21.1	31.6	44.8
2			4, 4, 8	6.1	14.1	25.0	37.7	52.3
3			8, 4, 4	7.0	16.0	26.0	37.8	52.8
4			8, 4, 8	5.4	14.3	26.1	43.0	63.4
5			8, 8, 8	6.8	17.2	36.4	62.9	84.7
6		Bicycle	4, 4, 4	0.4	1.6	2.8	4.4	6.7
7			4, 4, 8	1.0	3.5	6.3	9.5	13.0
8			8, 4, 4	0.6	1.6	2.9	5.1	8.4
9			8, 8, 8	1.8	4.7	6.7	8.5	11.4

Table 4.10: The RMSE values of the ST-GAT* model for the two mobility modules and different temporal settings when a certain percentage of input test data is masked.

For the UPE, the model appears to be more resilient for taxi rides than the RMSE, particularly when the input sequence size is 8. The UPE values for bicycle rides closely resemble their RMSE values. Generally, the model performs well for bicycle rides when the output horizon is one hour. The performance declines strongly as predictions reach further into the future.

Figure 4.17 shows the model's performance for the same day and setting as figure 4.3 and 4.6 in section 4.1 for the taxi and bicycle demand prediction. In this example, 30% of the inputs are missing for each of the mobility modules. While there are more outliers compared to the scenario with complete data, the predictions remain generally robust, capturing the various peaks throughout the day.

#	Model	Mobility	n_x, s, n_y	UPE worsening [%] when missing 10%	20%	30%	40%	50%
1	ST-GAT*	Taxi	4, 4, 4	2.9	6.8	12.1	19.5	30.4
2			4, 4, 8	2.4	6.2	12.1	20.0	31.3
3			8, 4, 4	3.3	7.4	11.9	17.0	23.3
4			8, 4, 8	2.6	6.9	13.0	22.9	40.7
5			8, 8, 8	3.2	8.5	16.3	32.8	58.4
6		Bicycle	4, 4, 4	0.6	1.7	2.9	4.4	6.5
7			4, 4, 8	1.4	3.6	6.0	8.7	11.7
8			8, 4, 4	0.6	1.3	2.2	3.7	5.9
9			8, 8, 8	1.3	3.2	4.8	6.3	8.8

Table 4.11: The UPE values of the ST-GAT* model for the two mobility modules and different temporal settings when a certain percentage of input test data is masked.

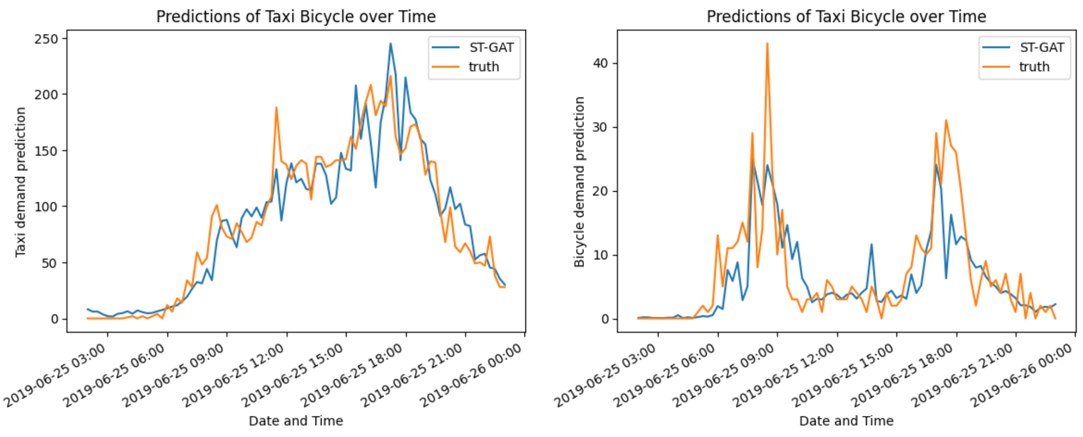


Figure 4.17: Two examples of predictions over the day of June 15, 2019. On the left for the taxi rides dataset and on the right for the rental bicycle dataset. 30% of the input values of these predictions were masked.

While the metrics and the way of replacing a missing value are not optimal, this shows that the model can be robust, even with large amounts of data missing. The predictions are definitely worse but also still comparable with other benchmarks that are not missing data.

4.8 Sensitivity Analysis

One of the main factors evaluated in this sensitivity analysis were the hyper-parameters of the GNN. Changing the hyper-parameters led to minimal differences in the selected metrics, as shown in the tables in the appendix .1. This observation can be linked to the robust nature of the ST-GAT* model, which has an inherent ability to absorb minor changes in hyper-parameters without significant impact on the overall performance. However, it's essential to note that different configurations of hyper-parameters could potentially lead to variations in performance, which might not be reflected in the chosen metrics but become evident in practical applications.

Incorporating an additional station into the model was found to be a complex task.

It does not just involve the addition of a new node in the graph, but it also means changes in the preprocessing steps. With the addition of a new station, the model has to be retrained as certain settings of the model can't be changed after the initial training. This process could potentially result in a significant change in the model's predictions and overall performance.

An alternative approach considered was to aggregate the data from the new station with that of the closest existing station. This method would indeed simplify the integration process and avoid the need for model retraining. However, this approach comes with its drawbacks as some important information from the new station could be lost during the aggregation process, potentially leading to less accurate predictions.

4.9 Discussion

The ST-GAT* model shows promise for predicting urban mobility demand. This thesis demonstrates its capacity to effectively integrate diverse data sources like taxi trips, rental bicycle stations, weather, and time-series data, to generate useful predictions.

However, there is a remaining challenge in measuring the ST-GAT* model's performance. The Root Mean Square Error (RMSE), commonly used across literature for performance evaluation, does not fully capture the complexity of the model's prediction accuracy. While RMSE provides a general metric of error measurement, it might not account for the complex aspects of the model's performance in different situations and demand levels.

The results show that ST-GAT* performs differently depending on the level of demand. Its predictions are better suited at higher demand levels. This observation could be attributed to the randomness and volatility observed in low-demand situations, where demands fluctuate between 0 and 1. On the contrary, in high-demand scenarios, the patterns are more consistent and predictable, allowing the model to make more accurate predictions.

Identifying the optimal graph structure for the ST-GAT* model presents another difficulty. While efforts have been made to develop an effective structure in this thesis, further research is necessary. The right graph structure can significantly impact the model's performance and ability to predict urban mobility demand.

The ST-GAT* model shows good performance under common input patterns such as rush hours. However, it struggles in less common scenarios like snow, gale, or holidays. This issue suggests the need for more diverse training data to allow the model to handle these less frequent events better.

The ST-GAT* model holds its own when compared to previous research. Its predictive accuracy is comparable, suggesting it could be a good alternative or complement to existing models. Further, it outperforms several baseline models tested in this study, underlining its potential for this predictive task.

In terms of sensitivity, the ST-GAT* model shows resistance to the addition of new nodes. This resistance indicates the potential challenge of scaling up the model to use an increasing number of nodes or expanding the spatial scope of prediction.

Robustness analysis shows that the model handles missing data well. It continues to provide reasonable predictions despite data gaps. However, it falters when presented with data that falls outside the normal range, indicating a weakness in dealing with anomalies.

Chapter 5

Conclusion and Outlook

The journey through this theses has shown various dimensions of Graph Neural Networks' (GNNs) potential and limitations in predicting urban mobility demand. The examination of diverse data types, performance metrics, model tuning complexities, and graph construction approaches have contributed to a more comprehensive understanding of the subject.

The primary conclusions drawn from the study are:

- GNNs, particularly the ST-GAT* model, have demonstrated considerable potential in tackling the complex task of predicting urban mobility demand. Nevertheless, a significant step was the construction of an optimal graph that captures the intricacies of urban mobility. Defining the ideal nodes and edges, maintaining the trade-off between complexity and computational feasibility, and ensuring the graph adequately represents the reality of urban mobility were some of the challenges encountered.
- Incorporating diverse data sources such as weather and time-series data into the GNN improved the model's ability to predict mobility demand. This reveals the need to account for various influencing factors that show the complex dynamics of urban environments. Understanding how to appropriately weigh and integrate these variables for maximum model performance is an interesting observation for future research.
- The crucial role of the right performance metrics became apparent. Conventionally used measures like RMSE, employed to benchmark against existing models or push the model's results, may not fully reflect the model's practical performance. The difficulty lies in finding a metric that can account for the specificities of the problem and the unique characteristics of the model's prediction, while remaining interpretable and comparable.
- The art of fine-tuning a model surfaced as an complex task. With numerous hyper-parameters to adjust, each affecting the model's learning and predictions in different ways, finding the optimal combination was not trivial. This aspect undermines the time-intensive and complex nature of model tuning.
- The ST-GAT* model could potentially be applied to any kind of urban mobility data from any city. This is particularly relevant in regions with high mobility

demands, as it could result in more accurate forecasts, thereby improving the coverage for various modes of transportation.

While working on the thesis a lot of restrictions and problems became apparent. Many of them have been addressed earlier on, but some remain subject to future research:

- The role of city planning in determining the model's targets and loss function should be further considered. Whether it is more beneficial to prefer the number of people or the density of people per area is a question that requires more in-depth exploration and could affect the model's performance and objectives significantly.
- Exploring alternatives to the Mean Squared Error (MSE) loss function could provide more balanced training. MSE has known limitations, particularly in tasks where the prediction targets have a large range or where the distribution of errors is not symmetrical. New loss functions could avoid these issues and enhance the model's predictive capacity.
- A different approach to testing robustness might involve selectively excluding nodes from the graph. This could provide insights into how the model behaves under varying graph structures, highlighting its adaptability or lack thereof.
- The use of other Spatial layers like Graph Convolutional Networks (GCN) or Graph SAGE could yield different, possibly improved, results. Each of these has unique characteristics and assumptions about the data, which may be more suitable for certain scenarios or data types.
- Investigating different graph structures, particularly learning from approaches in studies that dynamically learn the graph structure, like Wu et al. [35], may provide significant performance boosts. This could allow for a more flexible and adaptable model that can better handle the complexities of urban mobility.
- The exploration of dynamic graphs, where different graphs are used for different times of the day, seasons, or weather conditions, could lead to more nuanced and accurate predictions. This would reflect the fact that mobility patterns and influencing factors can change substantially depending on the context.
- Applying data augmentation techniques could be a potential strategy to address the model's limitations in dealing with rare input patterns like snow or heavy wind. By artificially creating more examples of these rare events, the model may be able to better generalise to them.

To sum up, the potential of GNNs in forecasting urban mobility is definetly apparent, yet this thesis has highlighted numerous areas requiring improvement and additional research. A vast amount of effort is still required to understand how various mobility patterns interact with each other. In the pursuit of creating greener, less congested cities, further investigation in this field is essential. It is my hope that this study will benefit the collective understanding of this subject. And it is important to remember, that our progress is built upon the monumental work of those who came before us—we are indeed standing on the shoulders of giants.

Bibliography

[1] Chai, D., Wang, L., & Yang, Q. (2018, November). Bike flow prediction with multi-graph convolutional networks. In Proceedings of the 26th ACM SIGSPATIAL international conference on advances in geographic information systems (pp. 397-400).

[2] Chen, B., Pinelli, F., Sinn, M., Botea, A., & Calabrese, F. (2013, October). Uncertainty in urban mobility: Predicting waiting times for shared bicycles and parking lots. In 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013) (pp. 53-58). IEEE.

[3] Davis, S., & Boundy, R. Transportation Energy Data Book: Edition 40, Oak Ridge National Laboratory, 2022.

[4] Ding, C., Wang, D., Ma, X., & Li, H. (2016). Predicting short-term subway ridership and prioritizing its influential factors using gradient boosting decision trees. Sustainability, 8(11), 1100.

[5] Dong, X., Lei, T., Jin, S., & Hou, Z. (2018, May). Short-term traffic flow prediction based on XGBoost. In 2018 IEEE 7th Data Driven Control and Learning Systems Conference (DDCLS) (pp. 854-859). IEEE.

[6] Frasconi, P., Gori, M., & Sperduti, A. (1998). A general framework for adaptive processing of data structures. IEEE transactions on Neural Networks, 9(5), 768-786.

[7] Glorot, X., & Bengio, Y. (2010, March). Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics (pp. 249-256). JMLR Workshop and Conference Proceedings.

[8] Guo, K., Hu, Y., Qian, Z., Liu, H., Zhang, K., Sun, Y., ... & Yin, B. (2020). Optimized graph convolution recurrent neural network for traffic prediction. IEEE Transactions on Intelligent Transportation Systems, 22(2), 1138-1149.

[9] Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive representation learning on large graphs. Advances in neural information processing systems, 30.

[10] Han, H., Zhang, M., Hou, M., Zhang, F., Wang, Z., Chen, E., ... & Liu, Q. (2020, November). STGCN: a spatial-temporal aware graph learning method for POI recommendation. In 2020 IEEE International Conference on Data Mining (ICDM) (pp. 1052-1057). IEEE.

[11] Huang, Y., Xu, H., Duan, Z., Ren, A., Feng, J., & Wang, X. (2020). Modeling complex spatial patterns with temporal features via heterogenous graph embedding networks. arXiv preprint arXiv:2008.08617.

[12] Jiang, X., Zhang, L., & Chen, X. M. (2014). Short-term forecasting of high-speed rail demand: A hybrid approach combining ensemble empirical mode decomposition and gray support vector machine with real-world applications in China. Transportation Research Part C: Emerging Technologies, 44, 110-127.

[13] Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

[14] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

[15] Li, X., Pan, G., Wu, Z., Qi, G., Li, S., Zhang, D., ... & Wang, Z. (2012). Prediction of urban human mobility using large-scale taxi traces and its applications. Frontiers of Computer Science, 6, 111-121.

[16] Li, Y., Yu, R., Shahabi, C., & Liu, Y. (2017). Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. arXiv preprint arXiv:1707.01926.

[17] Liu, T., Wu, W., Zhu, Y., & Tong, W. (2020). Predicting taxi demands via an attention-based convolutional recurrent neural network. Knowledge-Based Systems, 206, 106294.

[18] Ma, X., Dai, Z., He, Z., Ma, J., Wang, Y., & Wang, Y. (2017). Learning traffic as images: A deep convolutional neural network for large-scale transportation network speed prediction. Sensors, 17(4), 818.

[19] Masiol, M., Agostinelli, C., Formenton, G., Tarabotti, E., & Pavoni, B. (2014). Thirteen years of air pollution hourly monitoring in a large city: potential sources, trends, cycles and effects of car-free days. Science of the Total Environment, 494, 84-96.

[20] Mehta, P. (2020). Graphs Neural Networks in NLP. https://medium.com/neuralspace/graphs-neural-networks-in-nlp-dc475eb089de

[21] Micheli, A. (2009). Neural network for graphs: A contextual constructive approach. IEEE Transactions on Neural Networks, 20(3), 498-511.

[22] Moreira-Matias, L., Gama, J., Ferreira, M., & Damas, L. (2012, September). A predictive model for the passenger demand on a taxi network. In 2012 15th International IEEE Conference on Intelligent Transportation Systems (pp. 1014-1019). IEEE.

[23] Pearson, K. (1895). VII. Note on regression and inheritance in the case of two parents. proceedings of the royal society of London, 58(347-352), 240-242.

[24] Prado-Rujas, I. I., Serrano, E., García-Dopico, A., Córdoba, M. L., & Pérez, M. S. (2023). Combining heterogeneous data sources for spatio-temporal mobility demand forecasting. Information Fusion, 91, 1-12.

[25] Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., & Bronstein, M. (2020). Temporal graph networks for deep learning on dynamic graphs. arXiv preprint arXiv:2006.10637.

[26] Sak, H., Senior, A., & Beaufays, F. (2014). Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. arXiv preprint arXiv:1402.1128.

[27] Scarselli, F., Tsoi, A. C., Gori, M., & Hagenbuchner, M. (2004). Graphical-based learning environments for pattern recognition. In Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops, SSPR 2004 and SPR 2004, Lisbon, Portugal, August 18-20, 2004. Proceedings (pp. 42-56). Springer Berlin Heidelberg.

[28] Scarselli, F., Yong, S. L., Gori, M., Hagenbuchner, M., Tsoi, A. C., & Maggini, M. (2005, September). Graph neural networks for ranking web pages. In The 2005 IEEE/WIC/ACM International Conference on Web Intelligence (WI'05) (pp. 666-672). IEEE.

[29] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., & Monfardini, G. (2008). The graph neural network model. IEEE transactions on neural networks, 20(1), 61-80.

[30] Sperduti, A., & Starita, A. (1997). Supervised neural networks for the classification of structures. IEEE Transactions on Neural Networks, 8(3), 714-735.

[31] Tian, Y., & Pan, L. (2015, December). Predicting short-term traffic flow by long short-term memory recurrent neural network. In 2015 IEEE international conference on smart city/SocialCom/SustainCom (SmartCity) (pp. 153-158). IEEE.

[32] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., & Bengio, Y. (2017). Graph attention networks. arXiv preprint arXiv:1710.10903.

[33] Veloso, M., Phithakkitnukoon, S., & Bento, C. (2011, September). Urban mobility study using taxi traces. In Proceedings of the 2011 international workshop on Trajectory data mining and analysis (pp. 23-30).

[34] Wu, L., Cui, P., Pei, J., Zhao, L., & Guo, X. (2022, August). Graph neural networks: foundation, frontiers and applications. In Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (pp. 4840-4841).

[35] Wu, Z., Pan, S., Long, G., Jiang, J., Chang, X., & Zhang, C. (2020, August). Connecting the dots: Multivariate time series forecasting with graph neural networks. In Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining (pp. 753-763).

[36] Yang, S., Wu, J., Du, Y., He, Y., & Chen, X. (2017). Ensemble learning for short-term traffic prediction based on gradient boosting machine. Journal of Sensors, 2017.

[37] Yu, B., Yin, H., & Zhu, Z. (2017). Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. arXiv preprint arXiv:1709.04875.

[38] Zhang, C., James, J. Q., & Liu, Y. (2019). Spatial-temporal graph attention networks: A deep learning approach for traffic forecasting. IEEE Access, 7, 166246-166256.

[39] Zhang, L., Liu, Q., Yang, W., Wei, N., & Dong, D. (2013). An improved k-nearest neighbor model for short-term traffic flow prediction. Procedia-Social and Behavioral Sciences, 96, 653-662.

[40] Zhang, N., Zhang, Y., & Lu, H. (2011). Seasonal autoregressive integrated moving average and support vector machine models: prediction of short-term traffic flow on freeways. Transportation Research Record, 2215(1), 85-92.

[41] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., ... & Sun, M. (2020). Graph neural networks: A review of methods and applications. AI open, 1, 57-81.

Appendix

.1 List of Test Runs

Data	Model	Problem	Dropout	Adj Matrix	Batch size	Optimizer	LR	Epochs	Training RMSE	Validation RMSE	Test MSE	UPE	Top5 RMSE
Bike	ST Gat, 32 heads, LSTM hidden size: 256, 512	4,4,4	0.1	Similarity	64	Adam with LR decay of 5e-5	0.0003	5	0.58	0.73	0.68		
Bike	ST Gat, 8 heads, LSTM hidden size: 64, 128	4,4,4	0.1	Similarity	64	Adam with LR decay of 5e-5	0.0003	5	0.59	0.73	0.68		
Bike	ST Gat, 8 heads, LSTM hidden size: 64, 128	4,4,4	0.3	Similarity	64	Adam with LR decay of 5e-5	0.0003	5	0.59	0.73	0.68		
Bike	ST Gat, 8 heads, LSTM hidden size: 64, 128	4,4,4	0.1	Similarity	64	Adam with LR decay of 5e-5	0.00003	5	0.66	0.81	0.74		
Bike	ST Gat, 8 heads, LSTM hidden size: 64, 128	4,4,4	0.1	Similarity	64	Adam with LR decay of 5e-7	0.00008	5	0.61	0.74	0.68		
Bike	ST Gat, 8 heads, LSTM hidden size: 64, 128	4,4,4	0.1	Similarity	32	Adam with LR decay of 5e-7	0.00008	5	0.6	0.74	0.66		
Bike	ST Gat, 8 heads, LSTM hidden size: 64, 128	4,4,4	0.1	Similarity	128	Adam with LR decay of 5e-7	0.00008	5	0.62	0.76	0.7		
Bike	ST Gat, 8 heads, LSTM hidden size: 64, 128	4,4,4	0.1	Similarity	128	Adam with LR decay of 5e-5	0.0003	5	0.6	0.74	0.69		
Bike	ST Gat, 8 heads, LSTM hidden size: 64, 128	4,4,4	0.1	Similarity	128	Adam with LR decay of 5e-5	0.003	5	0.6	0.74	0.68		
Bike	ST Gat, 8 heads, LSTM hidden size: 64, 128	4,4,4	0.1	Similarity	256	Adam with LR decay of 5e-5	0.003	5	0.59	0.72	0.68		
Bike	ST Gat, 32 heads, LSTM hidden size: 64, 256	4,4,4	0.1	Similarity	32	Adam with LR decay of 5e-5	0.0003	10	0.58	0.72	0.66		
Bike	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	4096	Adam with LR decay of 3e-5	0.0001	60	0.65	0.7598	0.7087		3.38
Bike	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	4096	RMSprop with LR decay of 3e-5, momentum = 0	0.0001	10	0.7675	0.8999	0.8296		4.5
Bike	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	4096	RMSprop with LR decay of 3e-5, momentum = 0.9	0.0001	10	0.6577	0.7623	0.7099		3.39
Bike	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	4096	RMSprop with LR decay of 3e-5, momentum = 0.9	0.0001	30	0.6227	0.7397	0.6988		3.31
Bike	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	4096	NAdam with LR decay of 3e-5	0.0001	10	0.7748	0.9120	0.8359		4.55
Bike	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	4096	Adadelta with LR decay of 3e-5	0.0001	5	0.9209	1.0454	0.9567		5.33
Bike	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	4096	AdamW with LR decay of 3e-5	0.0001	10	0.7921	0.9287	0.8530		4.71
Bike	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	4096	Adamax with LR decay of 3e-5	0.0001	10	0.8364	0.9698	0.8988		4.98
Bike	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	4096	ASGD with LR decay of 3e-5	0.0001	10	1.0125	1.1422	1.0369		5.99
Bike	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	4096	Rprop with LR decay of 3e-5	0.0001	10	0.6650	0.7792	0.7226		3.48
Bike	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	4096	RMSprop with LR decay of 3e-5, momentum = 0.7	0.0001	10	0.6801	0.7970	0.7397		3.62
Bike	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	4096	RMSprop with LR decay of 5e-5, momentum = 0.9	0.001	20	0.6524	0.7544	0.7097		3.36

Table 1: 1/3: An overview of all hyper-parameters used for the test runs

Data	Model	Problem	Dropout	Adj Matrix	Batch size	Optimizer	LR	Epochs	Training RMSE	Validation RMSE	Test MSE	UPE	Top5 RMSE
Bike	ST Gat, 16 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	4096	RMSprop with LR decay of 5e-5, momentum = 0.9	0.0001	10	0.6510	0.7626	0.7101		3.40
Bike	ST Gat, 16 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	2048	RMSprop with LR decay of 5e-5, momentum = 0.9	0.0001	10	0.6401	0.7476	0.7050		3.37
Bike	ST Gat, 16 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	1024	RMSprop with LR decay of 5e-5, momentum = 0.9	0.0001	10	0.6330	0.7367	0.6956		3.29
Bike	ST Gat, 16 heads, LSTM hidden size: 128 , 256; initialized with orthogonal initialization	4,4,4	0.3	Similarity	2048	RMSprop with LR decay of 5e-5, momentum = 0.9	0.0001	10	0.6488	0.7576	0.7051		3.29
Bike	ST Gat, 16 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	2048	RMSprop with LR decay of 5e-5, momentum = 0.9	0.0001	10	0.6330	0.7367	0.6956		3.29
Taxi	ST Gat, 16 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	2048	RMSprop with LR decay of 5e-5, momentum = 0.9	0.0001	10	3.1154	3.0906	2.9602		
Taxi	ST Gat, 16 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	2048	RMSprop with LR decay of 5e-5, momentum = 0.9	0.001	10	3.3678	3.2287	2.9073		
Taxi	ST Gat, 16 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	2048	RMSprop with LR decay of 5e-5, momentum = 0.9	0.0001	10	3.1098	3.0783	2.8310		
Taxi	ST Gat, 16 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	2048	RMSprop with LR decay of 5e-5, momentum = 0.9	0.0001	10	3.1049	3.0726	2.9205		
Taxi	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	2048	RMSprop with LR decay of 5e-5, momentum = 0.9	0.0001	10	3.1502	3.1687	3.0331		
Taxi	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	2048	Adam with LR decay of 5e-5	0.0001	10	4.0052	4.0868	3.7606		
Taxi	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	512	Adam with LR decay of 5e-5	0.0001	10	3.4150	3.4556	3.4107		
Taxi	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	256	Adam with LR decay of 5e-5	0.0001	10	3.1725	3.2263	3.0806		
Taxi	ST Gat, 32 heads, LSTM hidden size: 256 * 2, 512 * 2	4,4,4	0.3	Similarity	50	Adam with LR decay of 5e-5	0.0001	10	2.9469	2.9523	2.7738		
Taxi	ST Gat, 16 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	50	Adam with LR decay of 5e-5	0.0001	10	2.9819	2.9851	2.8596		
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	1024	Adam with LR decay of 5e-5	0.0001	20	3.3225	3.3533	3.1676		
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	50	Adam with LR decay of 5e-5	0.0001	10	2.8905	2.8520	2.6381		
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	50	Adam with LR decay of 5e-5	0.001	10	2.8284	2.7589	2.5722		
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	512	Adam with LR decay of 5e-5	0.001	15	2.7854	2.7162	2.6058		
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 128 , 256	4,4,4	0.3	Similarity	512	Adam with LR decay of 5e-5	0.001	15	2.7362	2.6332	2.4311		
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	4,4,4	0.3	Similarity	512	Adam with LR decay of 1e-5	0.0005	10	2.7509	2.7144	2.5433	1.0129	

Table 2: 2/3: An overview of all hyper-parameters used for the test runs

Data	Model	Problem	Dropout	Adj Matrix	Batch size	Optimizer	LR	Epochs	Training RMSE	Validation RMSE	Test MSE	UPE	Top5 RMSE
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	4,4,4	0.1	Similarity	512	Adam with LR decay of 1e-5	0.0005	10	2.6574	2.5765	2.3627	0.9826	
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	4,4,4	0.1	Similarity	2048	Adam with LR decay of 1e-5	0.0005	30	2.6337	2.5461	2.3489	0.9707	
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	4,4,4	0.1	Similarity	128	Adam with LR decay of 1e-5	0.0005	30	2.3885	2.3656	2.2620	0.9278	
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	4,4,4	0.1	Similarity	1024	Adam with LR decay of 1e-5	0.0005	60	2.4275	2.4051	2.3165	0.9349	
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	4,4,8	0.1	Similarity	128	Adam with LR decay of 1e-5	0.0005	20	2.4267	2.4697	2.3529	1.2464	
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	8,4,8	0.1	Similarity	128	Adam with LR decay of 1e-5	0.0005	20	2.3593	2.5190	2.4476	1.2330	
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	8,8,8	0.1	Similarity	128	Adam with LR decay of 1e-5	0.0005	20	2.4291	2.5942	2.5384	1.3215	
Taxi, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	8,4,4	0.1	Similarity	128	Adam with LR decay of 1e-5	0.0005	20	2.2232	2.3490	2.1950	1.2550	
Bike, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	4,4,4	0.1	Similarity	256	Adam with LR decay of 1e-5	0.0005	20	0.5802	0.7222	0.6859	0.5208	
Bike, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	8,4,4	0.1	Similarity	256	Adam with LR decay of 1e-5	0.0005	20	0.5813	0.7388	0.6995	0.5431	
Bike, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	4,4,8	0.1	Similarity	256	Adam with LR decay of 1e-5	0.0005	20	0.5975	0.7595	0.7197	0.5619	
Bike, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	8,8,8	0.1	Similarity	256	Adam with LR decay of 1e-5	0.0005	20	0.6583	0.7994	0.7633	0.6227	
Bike, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	4,4,4	0.1	Similarity	256	RMSprop with LR decay of 1e-5 and momentum = 0.9	0.0005	20	0.5989	0.7176	0.6687	0.5325	
Bike, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	4,4,4	0.1	Similarity	2048	RMSprop with LR decay of 1e-5 and momentum = 0.9	0.0005	20	0.5938	0.7267	0.6883	0.5200	
Bike, W, T	ST Gat, 32 heads, LSTM hidden size: 256 , 512	4,4,4	0.3	Similarity	4096	RMSprop with LR decay of 1e-5 and momentum = 0.9	0.0001	60	0.5853	0.7259	0.6919	0.5201	

Table 3: 3/3: An overview of all hyper-parameters used for the test runs

.2 Additional Baseline Comparison Evaluations

#	Model	n_x, s, n_y	Training time	RMSE All Taxi Centroids t1	t2	t3	t4	t5	t6	t7	t8
1	ST-GAT*	4, 4, 4	10:05	2.066	2.137	2.194	2.250				
2	LSTM		7:22	2.249	2.320	2.397	2.464				
3	BiLSTM		7:51	2.325	2.390	2.472	2.581				
4	Persistence		-	3.284	3.673	3.956	4.222				
5	Naive		-	3.496	3.824	4.131	4.416				
6	ST-GAT*	4, 4, 8	13:37	2.139	2.217	2.295	2.339	2.389	2.427	2.475	2.512
7	LSTM		6:56	2.245	2.302	2.343	2.373	2.424	2.471	2.525	2.586
8	BiLSTM		7:12	2.259	2.332	2.389	2.432	2.493	2.567	2.645	2.732
9	Persistence		-	3.22	3.558	3.862	4.119	4.414	4.678	4.911	5.108
10	Naive		-	3.405	3.718	4.012	4.286	4.54	4.774	4.997	5.212
11	ST-GAT*	8, 4, 4	13:50	2.104	2.169	2.223	2.28				
12	LSTM		9:02	2.19	2.24	2.291	2.349				
13	BiLSTM		9:26	2.239	2.302	2.368	2.439				
14	Persistence		-	3.266	3.61	3.919	4.179				
15	Naive		-	3.888	4.167	4.431	4.68				
16	ST-GAT*	8, 4, 8	14:02	2.217	2.315	2.401	2.436	2.495	2.548	2.565	2.576
17	LSTM		8:55	2.334	2.374	2.402	2.439	2.492	2.55	2.61	2.709
18	BiLSTM		9:14	2.25	2.306	2.344	2.377	2.433	2.488	2.549	2.624
19	Persistence		-	3.281	3.627	3.938	4.2	4.502	4.77	5.007	5.208
20	Naive		-	3.891	4.165	4.428	4.678	4.911	5.129	5.347	5.554
21	ST-GAT*	8, 8, 8	13:45	2.363	2.408	2.441	2.483	2.559	2.625	2.674	2.725
22	LSTM		8:40	2.535	2.565	2.588	2.63	2.692	2.742	2.81	2.885
23	BiLSTM		8:37	2.498	2.525	2.555	2.575	2.622	2.689	2.76	2.826
24	Persistence		-	4.513	4.776	5.013	5.213	5.44	5.651	5.851	6.031
25	Naive		-	4.856	5.063	5.274	5.481	5.682	5.874	6.073	6.273

Table 4: The training time and RMSE calculated for all n_y predictions for different models and temporal settings of the taxi rides data.

#	Model	n_x, s, n_y	Training time	RMSE All Bike Stations t1	t2	t3	t4	t5	t6	t7	t8
1	ST-GAT*	4, 4, 4	13:09	0.671	0.681	0.69	0.7				
2	LSTM		6:05	0.671	0.68	0.692	0.702				
3	BiLSTM		6:11	0.675	0.686	0.699	0.711				
4	Persistence		-	1.01	1.056	1.094	1.127				
5	Naive		-	0.942	0.982	1.015	1.043				
6	ST-GAT*	4, 4, 8	12:29	0.671	0.685	0.701	0.715	0.73	0.741	0.75	0.759
7	LSTM		6:27	0.674	0.68	0.689	0.698	0.707	0.714	0.724	0.735
8	BiLSTM		6:12	0.679	0.688	0.7	0.712	0.724	0.733	0.743	0.753
9	Persistence		-	1.027	1.076	1.115	1.149	1.177	1.201	1.22	1.234
10	Naive		-	0.96	1.002	1.037	1.066	1.089	1.108	1.124	1.137
11	ST-GAT*	8, 4, 4	16:20	0.686	0.696	0.704	0.711				
12	LSTM		6:40	0.691	0.699	0.709	0.719				
13	BiLSTM		6:31	0.693	0.703	0.713	0.721				
14	Persistence		-	1.035	1.082	1.122	1.155				
15	Naive		-	0.979	1.01	1.035	1.057				
16	ST-GAT*	8, 8, 8	15:48	0.736	0.741	0.75	0.759	0.769	0.777	0.783	0.787
17	LSTM		6:04	0.75	0.752	0.759	0.765	0.77	0.773	0.774	0.776
18	BiLSTM		5:58	0.754	0.759	0.766	0.774	0.781	0.784	0.784	0.785
19	Persistence		-	1.231	1.256	1.276	1.291	2.305	1.316	1.325	1.333
20	Naive		-	1.118	1.134	1.147	1.159	1.17	1.18	1.189	1.198

Table 5: The training time and RMSE calculated for all n_y predictions for different models and temporal settings of the bicycle data.