Junior Management Science

www.jums.academy
ISSN: 2942-1861

Waiting Time Estimation for Ride-Hailing Fleets Using Graph Neural Networks

Hashmatullah Sadid

Technical University of Munich

Abstract

Ride-hailing services are part of intermodal transport systems, allowing passengers to use various transport modes for their trip. The optimal choice for a request in the intermodal system depends on the passenger's waiting time for the ride-hailing service. Estimating this waiting time is crucial for efficient system operation. The prediction of waiting time depends on the spatial dependency of the transport network and traffic flow elements. Graph neural network (GNN) approaches have gained attention for capturing spatial dependencies in various applications, though less attention has been given to ride-hailing waiting time prediction. The aim of this master thesis is to implement a GNN-based method to predict waiting time for ride-hailing requests in the network. Simulation-based waiting time data is used for model training and validation. MATSim is chosen for generating waiting time data under different demand and supply scenarios. Graph Convolutional Network (GCN) and Gated Attention Network (GAT) are used as prediction models. Regression and MLP methods are used as baselines to compare model performance. Results show GCN outperforms regression by 15%, while GAT performs 14% better than regression.

Keywords: graph convolutional network; ride-hailing service; waiting time estimation

1. Introduction

1.1. Background and motivation

The wide implementation of smartphone-based on-demand mobility services has already started changing the transport pattern of many cities. Ride-hailing platforms such as Uber, Lyft, Cabify, or Didi provide location-based and door-to-door taxi-like services by connecting riders and drivers in a centralized automated manner. These services provide the opportunity for passengers to order a customized ride using a smartphone application (Anderson, 2014; Henao & Marshall, 2019; Xu et al., 2020; C. Yan et al., 2020; Zha et al., 2016). Ride-hailing technologies create more business opportunities even for individuals with a car capable to offer taxi-like services (de Souza Silva et al., 2018; Lee et al., 2018). On the other hand, the complex dispatching algorithms used in these systems enable efficient matching of the customers and drivers considering both their spatial and temporal distributions. This leads to a significant reduction in search frictions in on-demand mobility services, as well as boosting the ordering and payment processes. As a result, lower costs are incurred both for riders and drivers (Anderson, 2014; Lee et al., 2018; Zha et al., 2016).

The successful implementation of a ride-hailing service depends on both customers' expectations and satisfaction as well as the suppliers' advantage. From a customer perspective, the price of the service, waiting time for the vehicle, reliability of the system, trip comfort, travel time among others are the key important perceived values (Gilibert & Ribas, 2019). Whereas for the suppliers, the accurate estimation of the demand and the recognition of potential service areas where competitors do not provide high-quality services (e.g. long passengers' waiting time) are the crucial decision elements in their business models. Among other influential factors, waiting time plays a vital role in generating ride-hailing demand and thus makes it an important factor for suppliers to find their optimal service areas.

Meanwhile, ride-hailing services are part of intermodal-transport systems, allowing passengers to use various transport modes for their entire trip. The optimal choice for a request in the intermodal-transport system, therefore, depends on the waiting time of a passenger until served by the ride-hailing service. Hence, estimating this waiting time is a crucial element for the efficient operation of the system. The waiting time of a passenger is the exact time that a passen-

DOI: https://doi.org/10.5282/jums/v10i2pp462-490

ger needs to wait in the mobility service platform until picked up.

Prediction and estimation of the waiting time of a customer for a ride-hailing service depend both on the spatial dependency of the transport network and traffic flow elements of the network. The spatial information includes the road network, demand and supply of the system, request location, and more, where the traffic flow elements demonstrate the variation of traffic flow variables and their relation to the waiting time in different time intervals. Thus, to predict and estimate the waiting time considering these influential factors, it is important to find an correlation among them. This is typically done by traditional statistical approaches or model-based methods. Statistical methods require more powerful feature engineering and assumptions (often leads to inaccurate estimations), where model-based (simulation tools) approaches require detailed modelling efforts and need high computational resources. Hence, machine learning-based approaches especially deep learning techniques have been widely used to improve prediction accuracy and have been utilized in many fields including traffic forecasting (C. Chen et al., 2019; K. Chen et al., 2021; Fang et al., 2020; Jin, Yan, et al., 2021; Q. Wang et al., 2021; Zhang et al., 2021).

Deep learning methods learn multiple layers of features by extracting more complex non-linear relationships. A convolutional neural network (CNN) for instance has been proved to reflect the spatial features of the network by modelling the whole city as a grid (Jiang & Zhang, 2019). However, due to non-Euclidean structure of the road network, this method is not optimal (Bronstein et al., 2017; Jiang & Luo, 2021; Z. Wu et al., 2021). Therefore, Graph neural networks (GNNs) have attracted attention for traffic forecasting problems, as they could capture spatial dependencies of a road network as a graph (i.e., intersections as nodes of the graph and road connections as edges of the graph) (C. Chen et al., 2019; Fang et al., 2020; Jiang & Luo, 2021; Q. Wang et al., 2021). For instance, Jin et al., 2022 and Jin, Yan, et al., 2021 used spatio-temporal GNN to estimate the travel time using real-world datasets, where X. Wang et al., 2020 employed spatio-temporal GNN for traffic flow prediction. Despite many studies in traffic forecasting problems (Fang et al., 2020; Q. Wang et al., 2021; Zhang et al., 2021), less attentions have been made to estimate the ride-hailing waiting time in a service area using deep learning approaches. Thus, it is imperative to design a deep learning method to predict ride-hailing waiting time in a service area considering spatial and operational features of network and traffic flow elements.

Meanwhile, deep learning algorithms require a large amount of data for training, testing, and validation. However, ride-hailing-related data, especially the waiting time information are not publicly available or cost-deficient for academic purposes. Thus it is advantageous to use simulation-based approaches to extract waiting time data in a service area.

1.2. Research Objective

The main goal of this master thesis is to implement a GNN-based method to predict the waiting time of a ride-hailing request in the transport network. This study employs a traffic simulation tool (i.e., PTV Vissim, SUMO, MATSim, etc.) to model traffic network features and ride-hailing service scenarios aiming to extract waiting time data for GNN implementation.

The following sub-aims are also included as a licentiate part of this work:

1. A literature review on the basics of neural networks, the theory of GNNs, different types of GNNs, and their applications, and
2. Developing a simulation-based platform for extracting ride-hailing waiting time data.

1.3. Structure of the thesis

This thesis is divided into six chapters. Chapter 1 introduces the background, motivation, and objectives of the study. Chapter 2 gives an overview of topics related to this thesis. First, an overview of ride-hailing simulation methods is presented. Subsequently, the basic theory of neural networks, and GNNs together with different types of GNNs are outlined.

In chapter 3, the methodology of this master thesis which contains the waiting time extraction approach and the proposed GNN framework which is the main contribution of this master thesis are introduced.

In chapter 4, an experimental setup is designed to generate waiting time data for the Cottbus city network, and implement it in the proposed models. Chapter 5 presents the main findings of the experiment together with a sensitivity analysis.

Chapter 6 gives a summary of the main contributions of this thesis, followed by a conclusion of the study and providing outlook for future researches.

2. Literature Review

In this chapter, we review related topics to this master thesis. First, an overview of on-demand mobility simulation including taxi modelling in MATSim is presented. Second, we introduce the basics of neural networks, GNN, different types of GNN, as well as their applications.

2.1. Ride-hailing Simulation

Waiting time is an important indicator for the efficient implementation of on-demand mobility concepts both from the demand and supply perspectives. Customers prefer choosing an on-demand mobility service (e.g., a taxi) if the waiting time is reasonable. On the supply side, the service providers attempt to place the taxis and dispatch them in the area where existing suppliers do not meet the customers' expectations (long waiting times). Since waiting time data is a core advantage of a service provider, this data is not shared openly. Meanwhile, accessing such data for academic purposes is cost-deficient, thus, an efficient approach is to generate waiting time data using traffic simulation models.

There are a variety of traffic models (macro-, meso-, and microscopic) to simulate and evaluate the implementation of a mobility concept (H. U. Ahmed et al., 2021; Grau & Romeu, 2015; Jing et al., 2020). However, selecting an appropriate traffic simulation tool depends on whether it is free to use, and include multi-mode simulation. In addition, the tool should be implemented in large-scale network, and be computationally efficient. Hence, in this study, we select an agent-based simulation tool (MATSim) with functionality to model different on-demand mobility concepts (Horni et al., 2016).

2.1.1. Agent-based Modelling

In this study, we use the Multi-Agent Transport Simulation (MATSim) tool to model the on-demand mobility service and extract waiting times. MATSim is an agent-based modelling framework with an iterative, co-evolutionary learning approach, where each agent attempts to maximize their daily utility by selecting from the set of planned activities. Agents receive rewards (positive scores) for performing scheduled activities and penalties (negative rewards) for long traveling or late arriving to an activity. After each iteration, agents score their executed plans with a resulting score. By learning from the experience, agents try to either adjust their plans (e.g., choosing a new route, selecting another mode) or choose among the best plans based on their scores (Horni et al., 2016).

In MATSim, a simulation run requires three input files namely: the configuration file, the population file, and the network file. The configuration file is the core input that assigns the simulation settings under which the simulation is carried out (HÖrl, 2017). The population file describes the daily plans of each agent by providing information about the socio-demographic attributes as well as their daily trips. The network file contains the links and nodes of the study area including specific information about the road network (type, capacity, mode, etc.).

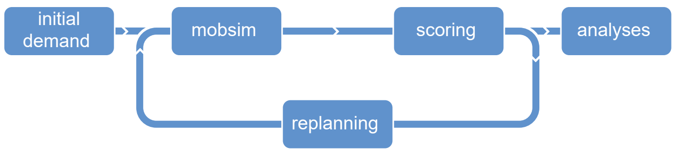


Figure 1: MATSim execution loop, source: (Horni et al., 2016)

The typical execution loop of MATSim is shown in Figure 1. The initial demand (population file) is inserted into the loop, where the simulation, scoring, and replanning of agents' plans are executed. In the first stage, all agents are moved along a physical network using Mobility Simulation (MobSim) unit. MobSim utilizes a spatial queue-based approach for traffic simulation without considering the microscopic driving behavior (car following and lane changing behaviors). In the scoring module, each agent's plan receives a reward using a utility function (e.g., performing an activity means a positive score, traveling gets negative rewards, etc.). Finally, the replanning module enables some agents to either adjust their existing plans or choose among the best plans (Horni et al., 2016). An example of the scoring scheme is shown in Figure 2.

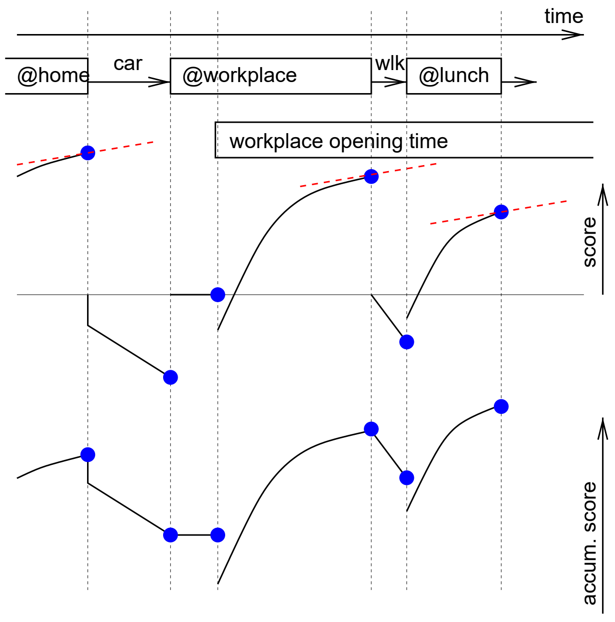


Figure 2: Scoring function representation, source: (Horni et al., 2016)

2.1.2. Taxi Modelling in MATSim

MATSim provides several extensions to model on-demand mobility services (Bischoff et al., 2017; Ruch et al., 2018, 2021). For taxi modelling, we utilize the dynamic vehicle routing problem (DVRP) extension integrated into MATSim by Maciejewski and Nagel, 2012. When a request with a coordinate and time is sent into the dispatching algorithm, the algorithm searches for a vehicle that can serve the request within a defined maximum waiting time (for more details on how the dispatching algorithm functions, readers are referred to (Maciejewski & Nagel, 2012)). To run a simple taxi simulation in MATSim, despite three main files (config, population, and network), we require the taxi location file to be added into the simulation framework. Worth mentioning that the taxi demand is defined within the population file, by modifying the mode choice of agents to taxis. The number of taxis in the network to serve a specific mode share of taxi demand is selected based on the minimum number of empty taxis during the peak hour demand.

When a request is submitted into the dispatching system, the dispatching algorithm assigns a taxi to this request. Once the taxi reaches a passenger and picks up the passenger, the taxi trip begins. The difference between the time a request is submitted and picked up is defined as the passenger waiting time. In this thesis, we try to extract the taxi waiting time for all agents under different demand scenarios.

2.2. Artificial Neural Networks (ANNs)

2.2.1. The basics of neural networks

Neural networks are a set of algorithms inspired by the human brain to mimic the relations and patterns from data (Maind & Wankar, 2014). Artificial neurons are the basic units of the neural network and are based on the Perceptron (Rosenblatt, 1957). The structure of an artificial neuron is displayed in Figure 3. Similar to biological neurons, a simple processing unit receives the input information and generates an output depending on the inputs. Overall, the information processing is conducted in five steps: First, the processing unit obtains the information as inputs $h_1, h_2, h_3, ..., h_n$. Second, each input is weighted by its corresponding weight denoted as $w_1, w_2, w_3, ..., w_n$. Third, a bias term b is added to the sum of all weighted inputs. Fourth, a non-linear activation function is applied and finally, the output y is generated. Mathematically a neuron output is expressed as:

$$y_j = \sigma\left(\sum_{i=1}^{n} h_i . w_i + b_j\right) \tag{1}$$

where y_j is the output of neuron, σ is the activation function, h_i is the input information, w_i is the corresponding learnable weight of neuron i, b_j is the bias term, and n is the number of inputs.

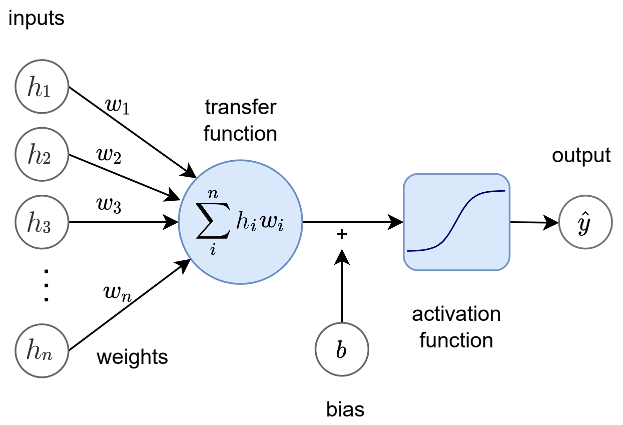


Figure 3: Illustration of an artificial neuron, perceptron. The input features are multiplied with the corresponding weights and the sum is added with a bias b, which then passes to an activation function. The outcome is the output value of the neuron.

The overall structure of neural networks consists of a number of interconnected neurons, which are arranged in at least three different layers namely: input layer, hidden layer(s), and output layer as shown in Figure 4. The input layer acts as an interface between the data and the network, and no calculation is done in this stage. The hidden layer(s) contains a predefined number of connected neurons, which transmit the information from the input layer to the neighboring hidden layers and then to the output layer. Finally, the output layer presents the results of the information process and depending on the type of prediction and the activation function may have different dimensions.

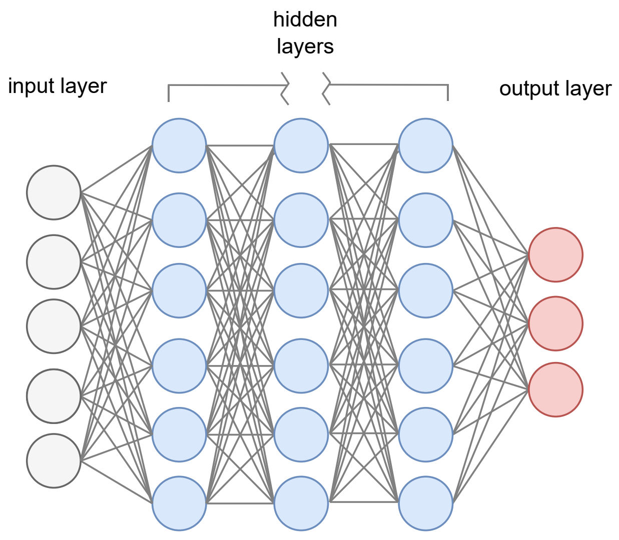


Figure 4: The structure of a fully connected 3-layer neural network. Source: (Saracoglu & Altural, 2010)

The connection between neurons in the hidden layer categorizes the neural networks into feed-forward and feedback networks. In a feed-forward neural network, information is fed in a forward direction from the input layer to hidden layers and finally to the output layer. The output of each neuron in the hidden layer is transmitted to the next neuron in the next layer. This continues until the information process reaches the output layer. On the other hand, in a feedback network, the information can be transmitted in both directions, from hidden layer l to hidden layers $l-1$ and $l+1$. In a feedback network, it is possible to create loops, where information can propagate continuously until an equilibrium condition is reached. Worth-mentioning that neural networks are widely used in many learning tasks including pattern recognition, classification, regression, signal processing, and more. In this section, we present multi-layer perceptron (MLP) in detail. MLP will be further used in this master thesis as a comparison method.

2.2.2. Multi-layer Perceptron (MLP)

MLP is a feed-forward artificial neural network that is comprised of connected layers namely: an input layer, one or more hidden layers, and an output layer (Bishop, 1995). In a fully-connected MLP, every neuron is linked to the consecutive neurons in the next layer. Depending on the prediction task, the output layer predicts and/or classifies the samples. In case, the MLP is used for classification purposes, the Softmax and its variants are utilized as the activation function in the output layer. For numerical prediction, however, Rectified Linear Unit (ReLU) is the most used activation function. The details of some activation functions are described at the end of this section. Formally, in a $(K+1)$ layers perceptron as depicted in Figure 5, where one input layer, (K) hidden layers, and one output layer are structured, the information processing of neuron (i) in layer (k) is expressed as:

$$y_i^k = \sigma\left(\sum_{j=1}^{m_{(k-1)}} y_j^{k-1}.w_{i,j}^k + b_{i,0}^k\right) \tag{2}$$

where (y_i^k) is the output of neuron i in layer k, σ is the activation function, m_l denotes the number of neurons stacked in hidden layer k, $w_{i,j}^k$ is the weight from the neuron j in layer $(k-1)$ to the neuron i in the layer (k), and $b_{i,0}^k$ is the bias for the neuron i in layer k.

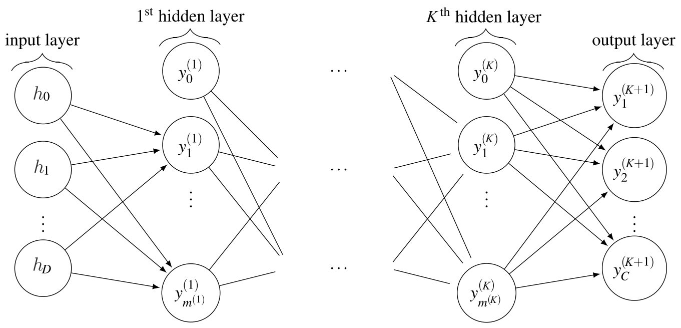


Figure 5: The structure of a fully connected MLP with K+1 layers, D inputs and C output neurons. Adopted from (Stutz, 2014)

The MLP model requires a training procedure to predict a realistic output. The training process takes advantage of the true values and/or labels of samples, and tries to minimize the difference between the predicted and true values using a loss function. Let's assume $\hat{y}$ as the MLP output and y as the true value (target), the backpropagation algorithm attempts to change the learnable weights w_{ij}, as well as the bias b in such a way to minimize the loss function L. Backpropagation algorithms are based on gradient descent and aim to optimize the model by converging the loss function to zero.

2.2.3. Activation Functions

Activation functions are the most important features of deep learning algorithms. In ANNs, the activation function of a neuron determines the output of that neuron with the given set of inputs, by simply mapping weighted inputs into an output. There are many activation functions in the literature, however, in this section, we present the widely used activation functions.

Sigmoid Function

The sigmoid function is a special form of logistic function which converts the model outputs into a probability score (between 0 and 1). As shown in Figure 6 (a), the sigmoid function generates a S-shaped curve and has a non-zero gradient throughout its domain. Hence, this function is a good candidate when applying the backpropagation algorithm.

$$f(x) = \frac{1}{1+e^{-x}} \tag{3}$$

Rectified Linear Unit (ReLU)

ReLU is a non-linear activation function that returns 0 if it receives any negative input, and returns the input value for any positive values (see Figure 6 (b)).

$$f(x) = \begin{cases} 0, & if \quad x<0 \\ x, & else \end{cases} \tag{4}$$

Hyperbolic Tangent Function

The tangent hyperbolic function denoted as tanh maps the input values between -1 and +1. Similar to the Sigmoid function, it also creates a S-shape curve as shown in Figure 6 (c).

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{5}$$

Heaviside Step Activation Function

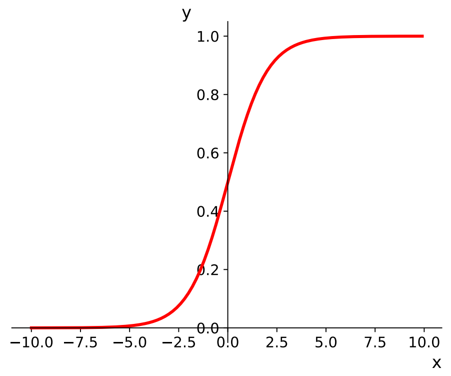
The Heaviside step activation function is a type of threshold function which generates the output 1 for positive inputs and 0 otherwise as displayed in Figure 6 (d).

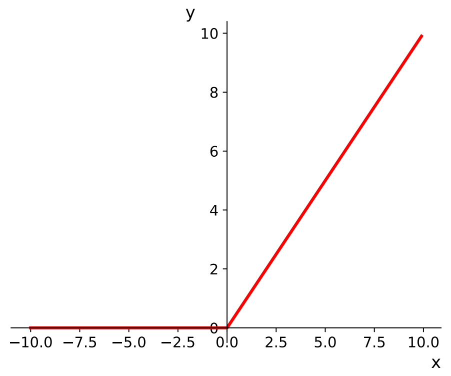
$$f(x) = \begin{cases} 0, & if \quad x<0 \\ 1, & if \quad x\geq 0 \end{cases} \tag{6}$$

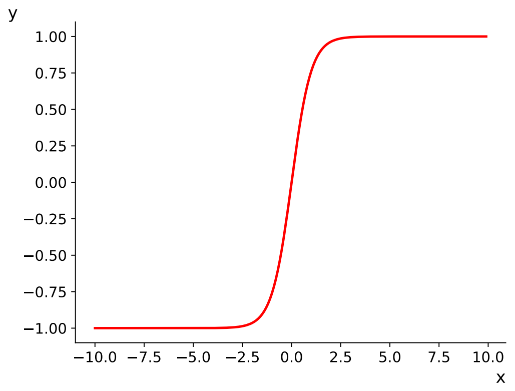
2.3. Graph Neural Networks (GNNs)

Over the past decade, deep learning paradigms such as convolutional neural networks (CNNs) (Lecun & Bengio, 1995), recurrent neural networks (RNNs) (Hochreiter & Schmidhuber, 1997), and autoencoders (Vincent et al., 2010) have become trending topics in artificial intelligence and machine learning. The superior performance of deep learning in many domains such as object detection, machine translation, speech recognition, etc., is partially related to the recent advancement in computational resources, the availability of big data, and the power of deep learning algorithms in extracting latent representation from Euclidean data. Although deep learning algorithms could efficiently replicate the hidden patterns of Euclidean data, their utilization for applications where data are generated from non-Euclidean domains in form of graphs is challenging (Bronstein et al., 2017).

Graphs such as social networks (Y. Wu et al., 2020), biological and chemical networks (Fout et al., 2017), transport networks (K. Chen et al., 2021; Jin, Yan, et al., 2021; Q. Wang et al., 2021), and other applications (Dai et al., 2018) are complex data structures that have created important challenges on existing deep learning methods (Zhou et al., 2020). These challenges include the irregularity of graph structure,


(a)


(b)


(c)

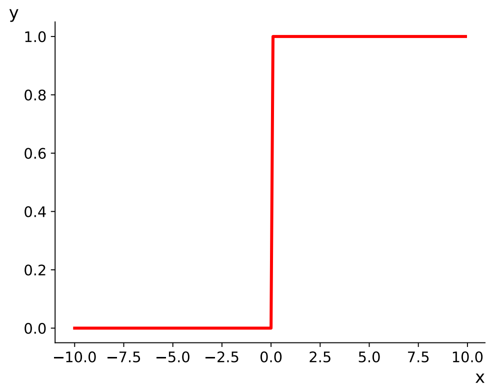
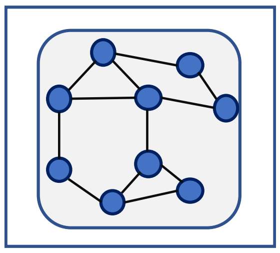

(d)

Figure 6: Plots of (a) Sigmoid, (b) ReLU, (c) tanh, and (d) Heaviside step activation functions

complex relationships, and interdependencies among the objects of a graph, as well as large-scale graphs (Jiang & Luo, 2021; Liu & Zhou, 2020; Z. Wu et al., 2021; Zhou et al., 2020). Hence, there is an increasing interest in extending the existing deep learning algorithms to mimic the graph data. Graph Neural Networks (GNNs) are widely used and the most successful in learning graphs data in various applications. In this section, we explore the basics of graph theory, GNNs, and different types of GNNs and their applications. This literature review aims to develop the methodology of this master thesis, which is described in the next chapter.

2.3.1. Graphs

Before introducing the GNNs, we briefly discuss the structure of a graph. A graph is a data structure that consists of two components namely vertices (nodes) and edges. A graph G can be defined as $G = (V, E)$, where V is the set of nodes, and E are the edges between them. The type of dependency between nodes categorizes the graphs into directed and undirected graphs. The information regarding the connection of nodes is often represented by adjacency matrix A. In a directed graph, all edges are directed from one node to another, whereas in an undirected graph, connected nodes are directed by a pair of edges with inverse directions (Z. Wu et al., 2021).


(a)

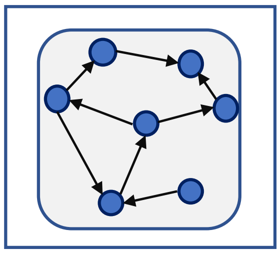

(b)

Figure 7: An illustration of (a) undirected, and (b) directed graphs

2.3.2. Graph Embeddings

Graph embedding is a method which is used to transform graph elements and their features into a low-dimensional space. The main goal of graph embedding is to map the original graph into a more computationally efficient format while keeping the original graph characteristics including the geometric relations and features of the graph (Gharaee et al., 2021; W. L. Hamilton, 2020; Hoff et al., 2002; Khoshraftar & An, 2022). Hence, the similarity in the embedding space approximates similarity in the graph or network.

The graph embedding could also be described using the encoder-decoder framework. An encoder model maps each graph element (i.e, nodes) into a low-dimensional vector or embedding. Whereas a decoder model takes the low-dimensional graph embeddings in the latent space and uses them to rebuild information about each node's neighborhood in the original graph. There are different embedding methods proposed in the literature. Khoshraftar and An, 2022 categorized graph embedding into traditional [e.g., Node2vec (Grover & Leskovec, 2016), Deepwalk (Perozzi et al., 2014), Graph factorization (A. Ahmed et al., 2013), Line (Tang et al., 2015), etc.] and GNN-based [e.g., RecGNN (Scarselli et al., 2009), GCN (Kipf & Welling, 2017), GraphSAGE (W. Hamilton et al., 2017), GCRN (Seo et al., 2016), DGCN (Zhuang & Ma, 2018), GAT (Veličković et al., 2018), and more] approaches. For more detailed information, readers are referred to (W. L. Hamilton, 2020; W. L. Hamilton et al., 2017; Khoshraftar & An, 2022).

2.3.3. The Concept of GNN

GNNs are deep learning methods operated for graph structure data. The basic idea of GNN is to iteratively update the representation of a node by aggregating its own representation and the representation of its neighboring nodes. GNNs use the representation of graph data including the node features and the connection between nodes. The output of a GNN model is the new representation of each node called embedding which contains the structural and feature information of other nodes. More specifically, each node knows about other nodes, the connection of nodes, and its context to the graph. The embeddings are further used for prediction.

The GNNs learn the representation vector of a node (h_v) by combining its own features and the neighboring node features (so-called message passing) with two important functions:

- **AGGREGATE**: The aggregate uses the states of all direct neighbors (u) of a node (v) and aggregates them in a specific method.
- **UPDATE**: The "update" operation uses the current state in time step (k) and combines it with aggregated neighbor states.

Within each message passing iteration in a GNN, a hidden embedding ($h_v^{(k)}$) related to each node $v \epsilon V$ is updated based on the information aggregated from the v node's neighbors ($N(v)$). The general framework of the GNN can be defined mathematically as follows:

$$\begin{aligned} {h_v}^{(k)} &= UPDATE({h_v}^{(k-1)}, \\ &\quad AGGREGATE\left(\left\{{h_u}^{(k-1)} : u \in N(v)\right\}\right)) \\ &= UPDATE^{(k-1)}\left({h_v}^{(k-1)}, {m_{N(v)}}^{(k)}\right) \end{aligned} \tag{7}$$

where $m_{N(v)}^{(k)}$ is the message that is aggregated from the v's neighborhood ($N(u)$).

More specifically, at each iteration k, the **AGGREGATE** function takes the information of each node (v) from its neighborhood $N(v)$ and generates a message $m(k)$. The embeddings of node (v) in iteration $k-1$ is combined with the message m(k) by the **UPDATE** function. The output of this process is the updated embeddings of node v ($h_v^{(k)}$). Worth-mentioning that at iteration $k = 0$, the initial embeddings of every node is basically the input features of all nodes ($h^{(0)} = x_u$). The final output after K iteration describes each node's embeddings in the embedding space.

Depending on the **AGGREGATE** and **UPDATE** operations, there are many variants of GNN models proposed in the literature. According to Z. Wu et al., 2021, GNNs are categorized into four groups namely: Recurrent GNN (RecGNN), Convolutional GNN (GCN), graph autoencoders (GAEs), and Spatio-temporal graph neural networks (STGNNs). In this thesis, we present the most relevant types of GNN including convolutional GNN, recurrent GNN, and graph attention networks. However, before presenting different types of GNNs, we briefly explain the basic **AGGREGATE** and **UPDATE** functions of GNN.

According to Merkwirth and Lengauer, 2005 and Scarselli et al., 2009 in the basic GNN, the **AGGREGATE** function contains trainable parameters which is defined as follows:

$${h_v}^{(k)} = \sigma(W_{self}^{(k)} {h_v}^{(k-1)} + W_{neigh}^{(k)} \sum_{v \in N(v)} {h_v}^{(k-1)} + b^{(k)}) \tag{8}$$

where $W_{self}^{(k)}, W_{neigh}^{(k)} \in \mathbb{R}^{d^{(k)} \times d^{(k-1)}}$ are parameters matrices and σ shows an element-wise non-linearity function such as tanh, ReLU, and $b^{(k)}$ is the bias term which is often eliminated for simplicity of the model.

In the basic GNN framework, the message passing is conducted similarly to a multi-layer perceptron (MLP), where linear operations are sent to a single element-wise non-linearity operation. More specifically, a linear combination of the sum of information received from the neighboring nodes and the previous embedding of a node itself is followed by a non-linear function.

Moreover, to omit the explicit update step and perform the message passing by only the aggregation function, the self-loops approach is proposed. In this approach the update function is defined through the aggregation method and the message passing is expressed as:

$${h_v}^{(k)} = AGGREGATE\left(\left\{{h_u}^{(k-1)}, \forall u \in N(v) \cup \{v\}\right\}\right) \tag{9}$$

Besides these two general methods, there are other generalized methods proposed for the AGGREGATE operator(i.e, neighborhood normalization). In the following subsections, we introduce different types of GNN mentioned in the above-section together with their aggregation methods.

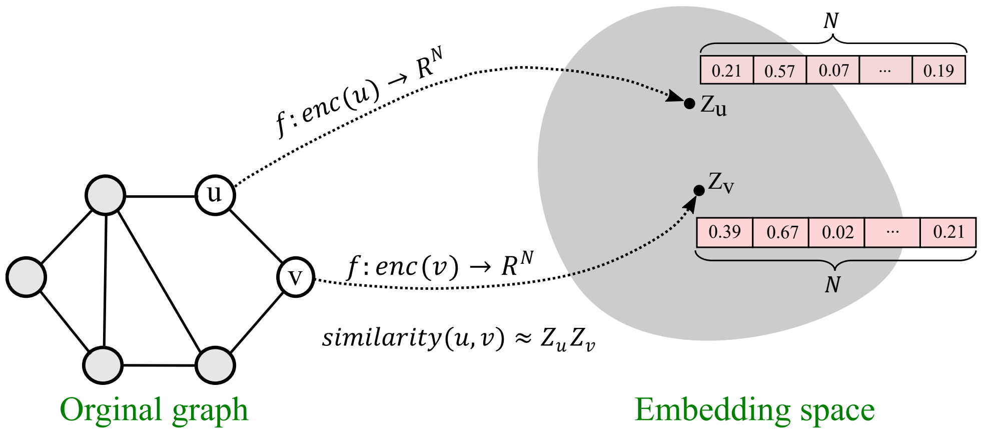


Figure 8: Illustration of the node embedding of a graph with 6 nodes, 7 edges, and each node has N features.

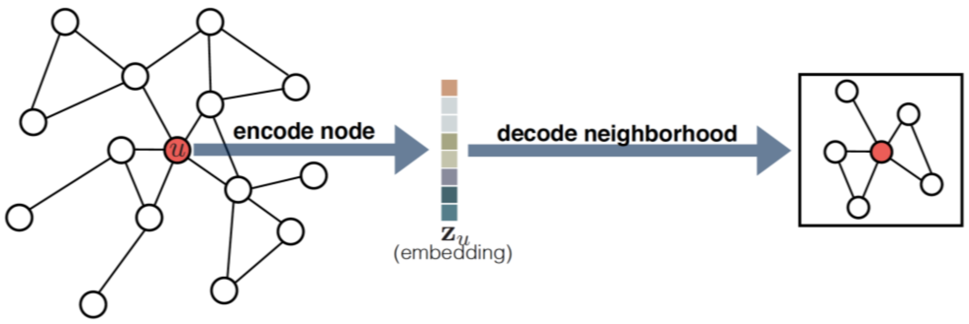


Figure 9: Overview of encoder-decoder framework. source: (W. L. Hamilton, 2020)

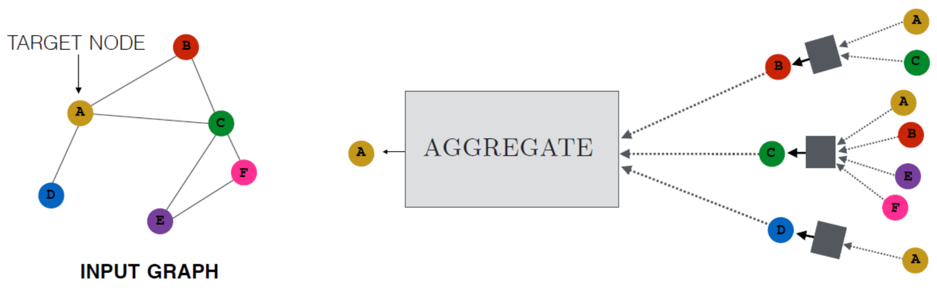


Figure 10: The information aggregation of a single node from its local neighborhood. source: (W. L. Hamilton, 2020)

Graph convolutional networks (GCNs)

GCN is the most popular GNN type which is extracted from the idea of the normal convolutional network. GCN can handle the cyclic mutual dependencies architecturally using a pre-defined number of layers with different weights in each layer. There are two types of GCN proposed in the literature namely spectral-based and spatial-based GCN. The first spectral-based method was proposed by Bruna et al., 2013. In this approach, graph convolution is defined by introducing filters from the view of graph signal processing. The information propagation in spectral GCN could be similar to signal propagation along the nodes. In spectral GCN, the convolution operation is defined in the Fourier domain by calculating the Eigen-decomposition of graph Laplacian matrix (for in-depth details, the reader is referred to (Z. Wu et al., 2021)).

On the other hand, spatial-based approaches consider the information propagation by operating on spatially close neighbors to define graph convolution. In the method proposed by Kipf and Welling, 2017, a symmetric-normalized aggregation with self-loop update operation is employed. The message-passing function of the GCN model is expressed as follows:

$$h_v{}^{(k)} = \sigma\left(W^{(k)} \sum_{u \in N(v) \cup \{v\}} \frac{h_u}{\sqrt{|N(v)|\,|N(u)|}}\right) \tag{10}$$

Worth-mentioning that the simplified spectral-method proposed by Kipf and Welling, 2017 could also be considered as spatial-based GCN. The details of this GCN method is presented in the methodology section of this master thesis.

Recurrent Graph Neural Networks (RGNN)

RGNN is a particular class of recurrent neural networks (RNNs) that is applied to sequential data. RGNN uses the same set of parameters as in GNN recurrently over nodes in a graph to extract high-level node representations. The conventional prediction methods of RNN encounter computational challenges, especially for long-term information propagation. Thus, to address this issue and reduce the exploding and gradient vanishing problems, Gated recurrent unit (GRU) and Long-short term memory (LSTM) are introduced. GRNN employed with a GRU or LSTM is called gated GNN - GGNN.

GGNN reduces the recurrence to a fixed number of steps and also it does not limit the parameters for convergence. The general framework of GNN in case it is employed with a GRU unit is defined as:

$$h_v{}^{(t)} = GRU\left(h_v{}^{(t-1)}, \sum_{u\in N(v)} Wh_u^{(t-1)}\right) \tag{11}$$

Since there is a strong interdependency between the elements of traffic in the transport network, a gated method is effective to be used for capturing the sequential relationship of the data.

Graph Attention Neural Networks (GAT)

In the aggregation process, the basic GNN framework puts equal weight on all neighbor nodes. However, not all neighbors are equally important. The aim of GAT is to apply attention to neighbor nodes to indicate the importance of each node during the aggregation step. For instance, in a road network, several links connecting to one intersection might have different traffic load, and thus should be captured with different scores. Veličković et al., 2018 proposed attention weights which define a weighted sum of the neighbors into the propagation steps. In the approach, the aggregated message passing is expressed as:

$$a_{ij} = \frac{exp(\sigma(a^\top[Wh_i]||[Wh_j]))}{\sum_{k\epsilon N_i} exp(\sigma(a^\top[Wh_i]||[Wh_k]))} \tag{12}$$

where a_{ij} is the attention coefficient of node j to i, N_i indicates the neighborhoods of node i in the graph. In addition, σ is the non-linear activation function (LeakyReLU), $h\epsilon R^{N\times F}$ is the input node features (N: number of nodes, F: dimension of the features), W is the weight matrix, and a is the learnable attention vector. Thus, the final output feature of each node is predicted as follows:

$$h_i = \sigma\cdot\big(\sum_{j\epsilon N_i} \alpha_{ij} Wh_j\big) \tag{13}$$

In addition, the multi-head attention mechanism similar to (Vaswani et al., 2017) stabilizes the learning process. Thus, in each layer, the K-independent attention mechanism (each with different parameters) is applied to compute the feature-wise aggregated output (normally by average). The equation 13 is transformed, and their features are concatenated as follows:

$$h_i = \coprod_{k=1}^{K} \sigma\cdot\big(\sum_{j\epsilon N_i} \alpha_{ij}^k W^k h_j\big) \tag{14}$$

where $\coprod$ indicates concatenation, and K is the number of attention mechanism.

Since multi-head attention is performed on the final prediction, therefore the average of the resulting attentions are considered. The final representation after averaging takes the following form:

$$h_i = \sigma\cdot\big(\frac{1}{K}\sum_{k=1}^{K}\sum_{j\epsilon N_i} \alpha_{ij}^k W^k h_j\big) \tag{15}$$

An illustration of the aggregation process of multi-head attention layer is shown in Figure 11.

Spatio-temporal Graph Neural Network (STGNN)

In real-world applications, graphs could have dynamic characteristics both in terms of the graph structure and features (Z. Wu et al., 2021). For instance, in a transport network, the link features (e.g., speed, travel time, traffic flow) change during the day and thus it is required to capture both the spatial dependency, and the temporal variation of the graph. Spatio-temporal graph neural networks (STGNN) have attracted attentions in mimicking the spatial and temporal properties of graphs simultaneously. In many STGNN related studies, GCN are integrated with a temporal block (e.g., GRU, LSTM) to capture the spatial and temporal dependencies respectively.

STGNNs have been implemented in many applications including transportation (Li et al., 2018; X. Wang et al., 2020; Yu et al., 2018), driving behavior prediction (Jain et al., 2016), environment monitoring (Jin, Sha, et al., 2021; S. Wang et al., 2020), human action recognition (S. Yan et al., 2018) and more. Figure 12 shows an illustration of STGNN which is comprised of GCN for capturing spatial dependency, and a temporal block- GRU for representing the temporal features.

To summarize, GNNs are powerful techniques in capturing graph data. For mimicking only the spatial dependencies of the graph data, GCN and GAT could be applied, where for capturing the temporal variation of a graph data, STGNNs are widely utilized in literature. Since our extracted simulation-based data will only consider a request point with its associated waiting time without considering the variation of the

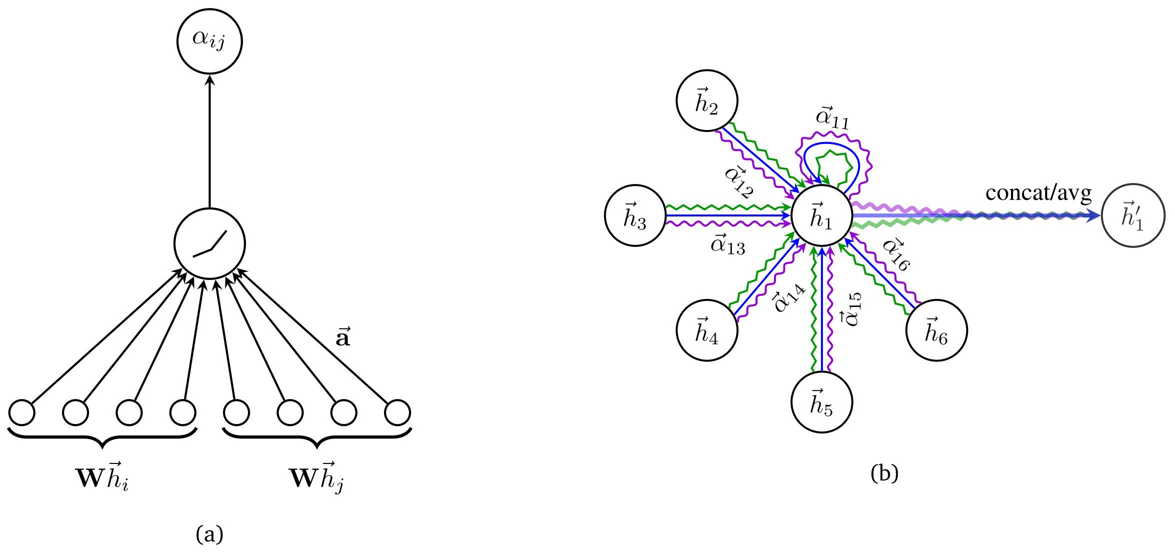


Figure 11: A display of GAT model (a) the attention mechanism of the model, and (b) the multi-head attention differentiated with different colors by node 1 and it's neighbors. Source: (Veličković et al., 2018)

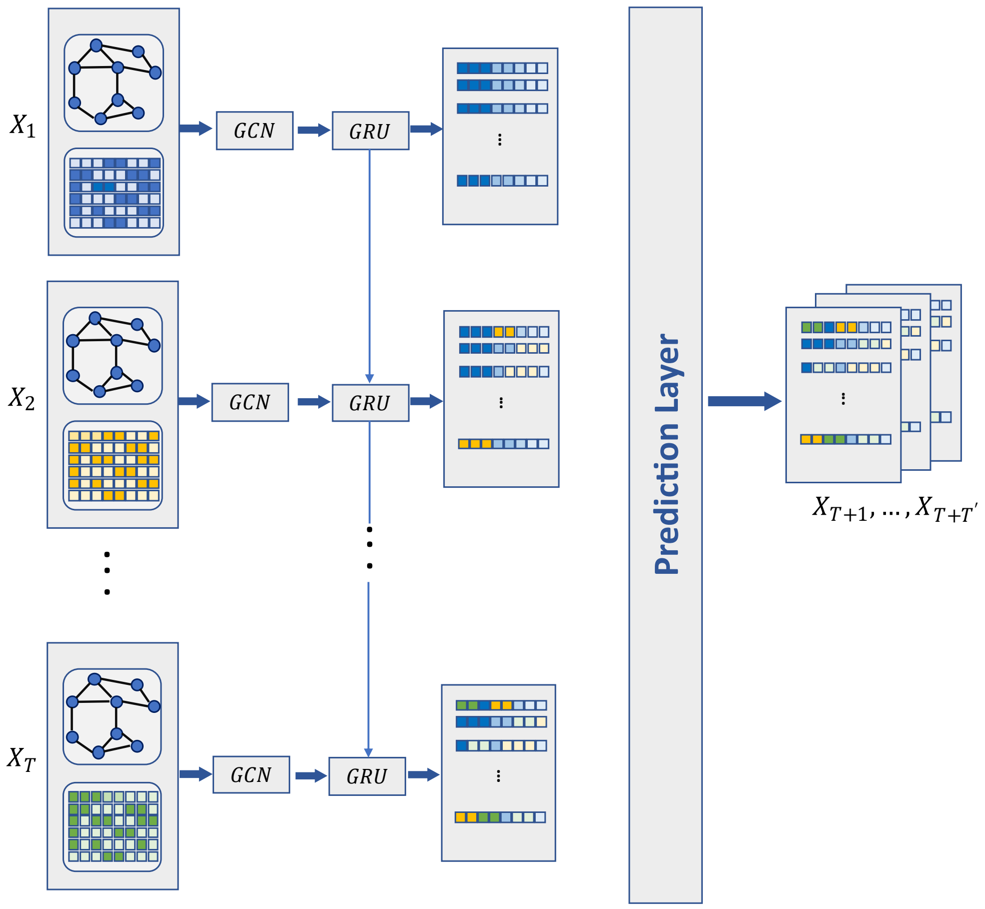


Figure 12: A sample structure of STGNN, where GCN captures the spatial dependenciy , and the temporal variation is represented by GRU, (own illustration)

waiting time in different time periods during the day, we utilize a model to predict the spatio-operational dependencies of the graph data in regards to the waiting time values. Hence, GCN and GAT model are the best candidates and are utilized in this master thesis.

3. Methodology

The methodology of this master thesis is comprised of four major sections namely: (i) the data generation, (ii) the proposed framework, (iii) the model evaluation, and (iv) the model transferability. In the data generation section, we introduce the process of ride-hailing simulation and extracting the waiting time data, followed by the map matching in QGIS and finally the transformation of the data for the final implementation in the proposed model. The proposed framework is the main contribution of this master thesis, which describes the models that are used in the experimental setup in the next chapter. Third, the model evaluation section presents the design of the loss function as well as evaluation matrices for the assessment of the proposed models' performance. Finally, in the model transferability section, the analysis of generalization capability of the trained models on a different dataset is described.

3.1. Data Generation

In this study, we utilize MATSim tool to model the ride-hailing and extract requests' waiting time. The simulation process contains running different scenarios including demand and supply variations as well as matching the outputs of the simulation runs with the road network geometry.

3.1.1. Ride-hailing simulation in MATSim

To conduct a simple simulation run in MATSim, config, population, and network files are required as describe in section 2.1.1. To include taxi modelling, we further need the definition of taxi demand within the population file and the taxi distribution data. The definition of the demand and optimal supply is a crucial step in extracting the waiting time data. Foremost, we extract the waiting time information under different penetration rates of ride-hailing services, with three different supply policies, namely: (i) Optimal supply, (ii) 20% above optimal supply, and (iii) 20% less than the optimal supply. This is done due to the fact, that we have excess and shortages of supply in reality. A schematic of the simulation scenarios including different demand and supply is shown in Figure 13.

Theoretically, we need the waiting time of all agents in the network, as for the proposed GNN, the waiting time information is needed for all nodes. However, a 100% penetration rate of ride-hailing in the network generates a huge number of empty vehicles driving in the network and thus create a huge congestion. This of course results in unrealistic waiting times. Hence, to avoid such an issue and meanwhile generate waiting time information for all agents, we propose a method which extract waiting time for all agents, however, not in one simulation setup, but under several simulation runs. To clarify, let's assume a service area where **x**% of all agents use ride-hailing for their daily trip activities. However, it is not known which agents exactly use ride-hailing and consequently any agent could be a potential candidate. Thus, first we randomly select **x**% of all agents to use ride-hailing and run the first simulation and store the resulting

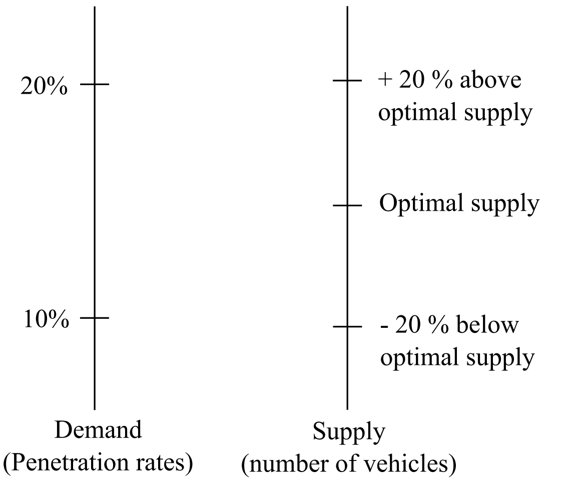


Figure 13: Different scenarios for demand scales in terms of ride-hailing penetration rates,and supply in terms of number of vehicles.

output. Second, another **x**% of all agents are chosen where they were not selected in step 1 and change their mode to ride-hailing and conduct the second simulation run and same here we store the output. Based on the value of **x**, a total of **n** different plan files are generate and sent to the MATSim simulator. Using this approach, it is intended to have ride-hailing requests for all agents but under **x**% penetration rate, and meanwhile we achieve more realistic data. The final extracted waiting times includes the waiting of request for all agents, however, under **x**% penetration rate of ride-hailing demand. Figure 14 shows the schematic overview of the agents plan creation and simulation.

On the supply side, determination of the optimal number of ride-hailing vehicles follows a trial and error approach. Depending on the network and demand size, several simulation runs have been conducted to find an optimal number of vehicles. First, we run the model with bigger fleet sizes and check the number of vehicles being idle for most of the day. Similarly, simulation runs with smaller fleet sizes are also done. Hence, simulation runs continue from bigger and smaller fleet sizes until an optimal number of vehicles is reached.

3.1.2. Map Matching

The outputs of various simulation scenarios are the information of requests points with their locations (home, work, leisure, etc.) and simulated waiting times. To allocate these requests points to their nearby links and nodes, QGIS tool is used for the purpose of map matching. The distribution of the request points could have different relations to the nearby links. First, several request points could be located nearby a single link, which requires averaging the waiting times and then allocating a single waiting time to the link. Second, some request points, are far away from the nearby links and thus, it is difficult to programmatically allocate them to those links. To address this issue and better allocate request points to corresponding links, we propose creating grid cells.

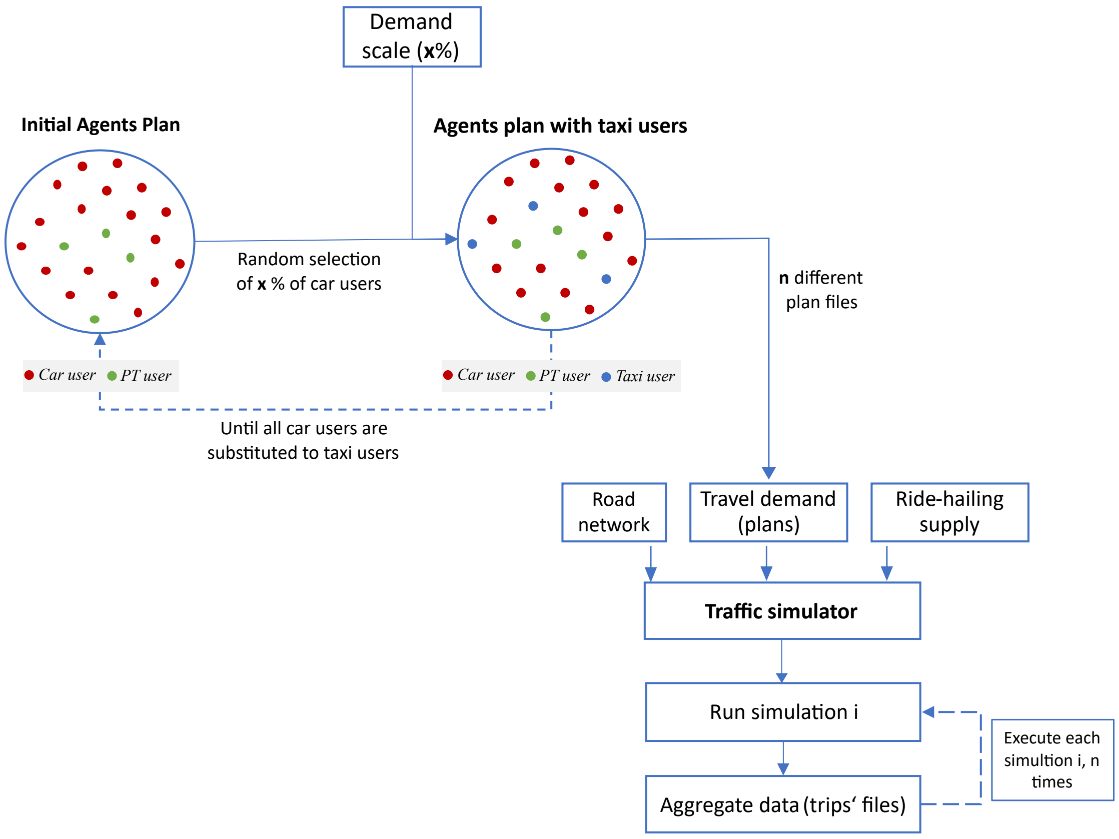

Demand scale (x%)
Initial Agents Plan
Agents plan with taxi users
Random selection of x % of car users
n different plan files
Car user
PT user
Taxi user
Until all car users are substituted to taxi users
Road network
Travel demand (plans)
Ride-hailing supply
Traffic simulator
Run simulation i
Execute each simultion i, n times
Aggregate data (trips' files)

Figure 14: Schematic overview of the ride-hailing demand creation and MATSim simulation

However, since we conduct node prediction in our final model, and this requires obtaining the waiting time information from the links emerging from a single node, it saves one step to directly acquire the waiting information for each node from the grid cells rather than the links. Hence, the node-level waiting time information will be directly extracted from the grid cells.

We created grid cells of 250*250 meters on our network, these grid cells contain both nodes, and request points as shown in Figure 15. First, the average waiting time of each cell is determined by counting the number of request points in each cell and then the attributes of request points are averaged. Of course, it is only needed to know what is the average waiting time for ride-hailing in each cell. Second, after generating the waiting time of each cell, we assign them to the containing nodes in a cell. For instance, let' assume that cell (a) contains three different nodes, thus the waiting time of those nodes could be potentially the same as the grid cell itself. Worth-mentioning that we deal with missing cell waiting time by assigning the nearby cell waiting time to it.

Meanwhile, the census data is used to extract the population density in each grid cell. This information is important, since ride-hailing request closely relate to the population density of the area. Similar to waiting time extraction for each node, we allocate the population density around a node by assigning the corresponding cell population density to it.

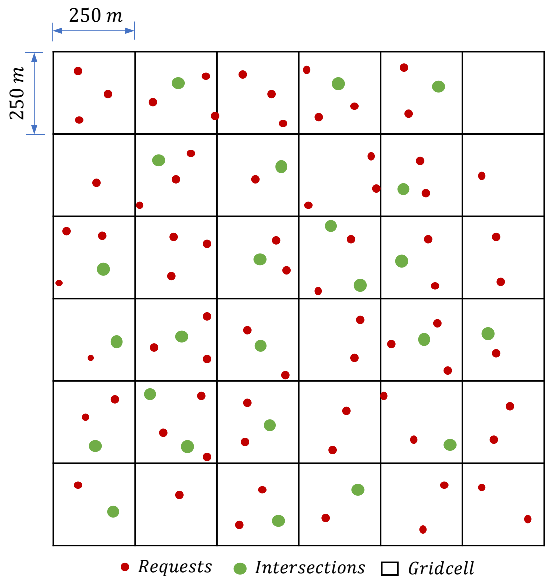


Figure 15: Illustration of a sample grid cells, request points, and road network

In addition, to include the ride-hailing vehicles distribution in our features, we map the vehicles (assigned randomly in the network during the simulation) in our network. Since, vehicles are assigned to links and their exact locations are not known, the link ID information is used to relate availability of a vehicle to a node using a dummy variable. We allocate the information from a link (0 if there is no vehicle to the link, and 1 otherwise) to the associate node and consequently create the vehicle distribution feature for the nodes.

To summarize, in this step the information for each node including population density, vehicle distribution, and the waiting time are created. The first two information will be utilized as a constant features of the nodes, where waiting time is the targets of our data for model training.

3.1.3. Feature Generation and Data Transformation

For the implementation of a GNN pipeline, we need a set of nodes with their features and the links connecting them. In a road network, most of the features are associated with links in the graph, whereas for the intersections (nodes), only a few pieces of information such as the location, type, etc are available. Therefore, for the GNN model, we should perform link prediction by using a dual graph to change the nodes to links and the links to nodes. However, the application of a dual graph changes the real typology of a road network. Thus, we propose a method so-called flow-out approach which aggregates the information of the links to the corresponding node.

The flow-out approach is straightforward. Each link is indeed emerging from a node and sinks to another node. Suppose, four links are emerging from a single node (e.g., node u in Figure 16), thus, the average of the attributes of these links could be assigned as the aggregated features of the corresponding node. As depicted in Figure 16, the aggregated attributes of the node u and v are calculated as follows:

$$\begin{aligned} h^{(u)} &= mean(h^{(e_{u,v})}, h^{(e_{u,1})}, h^{(e_{u,2})}, h^{(e_{u,3})}, h^{(e_{u,4})}) \\ h^{(v)} &= mean(h^{(e_{v,u})}, h^{(e_{v,5})}, h^{(e_{v,6})}, h^{(e_{v,7})}) \end{aligned} \tag{16}$$

where $h^{(u)}$ and $h^{(v)}$ are the aggregated features of node u and v respectively, and $h^{(e_{u,v})}$, $h^{(e_{u,1})}$,...,$h^{(e_{u,4})}$ are the features of emerging edges from node v.

Using this approach,the graph data are successfully created, which comprises nodes with their features and a set of links that shows the connection between the nodes. Worth-mentioning, that except for population density, and vehicle distribution which come from the grid cells and node degree attribute which belongs to each node, all other features are aggregated from the corresponding links. A sample presentation of the feature data as well as the edge information are depicted in Table 1.

Finally, the data is further processed and prepared for the final implementation. The detailed description of the data transformation is presented in section 4.5.

3.2. Proposed Framework

The proposed framework in this master thesis consists of two components, the embedding module, and the spatial dependency learning module. The embedding module aims to map the constant and variable properties of each node to a low-dimensional space and to further insert them into the learning module. On the other hand, the spatial dependency learning module requires the nodes embedding and edges information for node representation learning. In the following sections, each module is described in details.

3.2.1. Embedding Module

Since waiting time for ride-hailing requests is affected by many spatial factors such as the location of the request, the population density of the request area, connectivity of the nearby roads as well as the operational factors including capacity of links, speed limit, and ride-hailing vehicles distribution in the network. We initialize the node embedding by concatenating these features and mapping them into a low-dimensional latent space (one-dimensional tensor).

The embedding of nodes can be formulated as follows:

$$h_v = \sigma(W_v \cdot [L \parallel C \parallel S \parallel P \parallel V \parallel T \parallel F\ ] + b_v) \tag{17}$$

where W_v and b_v are the learnable weight and bias respectively, || is the concatenation operator, and L, C, S, P, V, T, F are the node features (please see Table 3 for description of each feature).

3.2.2. Spatial Dependency Learning Module

This module is the core contribution of this master thesis. The main function of this module is to predict the waiting time based on given spatio-operational attributes of the road network by implementing both GCN and GAT.

a) Graph Convolutional Network (GCN)

In a traffic network, nearby roads are more likely to share common attributes such as capacity, speed limit, and more. Thus, closely located nodes and links share both spatial and operational features. On the other hand, a GCN model with its $l-$layers is capable to aggregate and average the hidden representation of each node with its neighbors. We can apply the GCN approach, to predict the feature of a single node by aggregating its own features and the features of the neighboring nodes. Let's consider a graph $G = (V, E)$ with the following descriptions:

- A feature matrix $H^{(0)} = X_{in} \epsilon R^{N \times D}$, with N : number of nodes, and D : number of input features.
- An adjacency matrix A which describes the graph structure and their relations.

The output of the model $H^{(l)} \epsilon R^{N \times F}$ can be generated as follows:

$$H^{(l)} = f(H^{(l-1)}, A) \tag{18}$$

with $H^{(0)} = X_{in}$ the initial nodes' representations, $H^{(L)} = X_{out}$ is the final nodes' representations, and L is the number of convolutional layer.

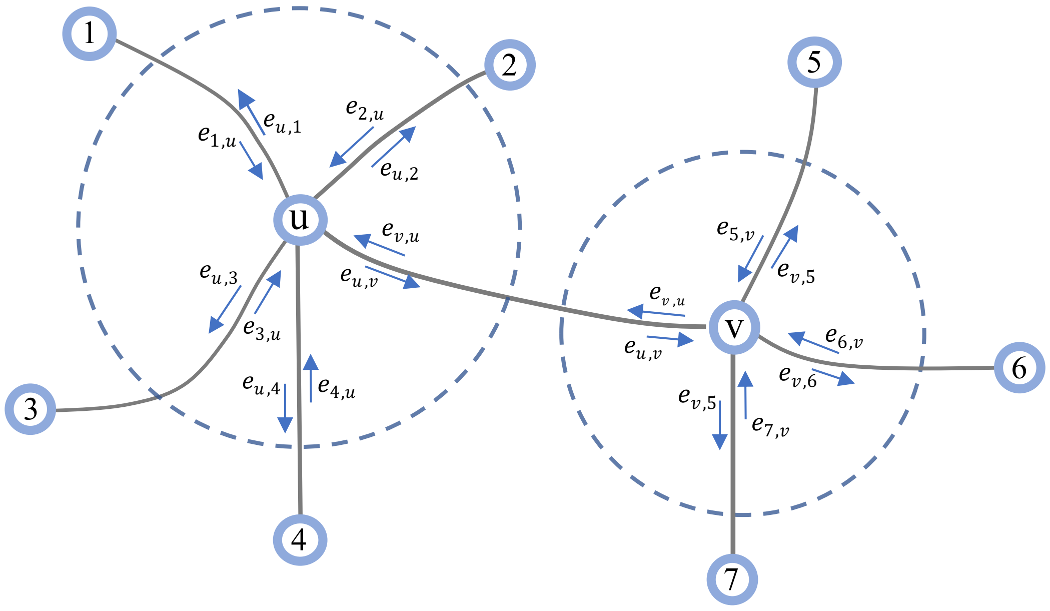


Figure 16: Illustration of the flow-out approach and allocation of links attributes to corresponding nodes

Table 1: A sample presentation of the graph data (node features and edge information)

Node	Capacity	speed limit	...	Vehicle
1	500	50		0
2	300	30		2
3	620	70		1

(a)

Link	fromNode	toNode
1	60	81
2	123	5
2	6	51

(b)

In this master thesis, we utilize the propagation rule introduced in (Kipf & Welling, 2017) as follows:

$$H^{(l+1)} = f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{19}$$

with $\hat{A} = A + I$, where I is the identity matrix, and $\hat{D}$ is diagonal node degree matrix of $\hat{A}$.

The implementation of this GCN model could be simply described in three steps namely: (i) Feature propagation, (ii) Feature transformation, and (iii) Activation layer.

Feature propagation: In each layer, we take the average of the feature vectors of the nodes' neighbors.

$$H^{(l)} = \tilde{A}H^{(l-1)}$$

where $\tilde{A} = \hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix including the self loops.

Feature Transformation: Each layer contains learnable weight matrix, which linearly transform the smoothed hidden feature representation to the next layer.

$$\tilde{H} = H^{(l)}W^{(l)}$$

where $\tilde{H}$ is the hidden layer representation.

Activation Layer: To ensure non-linearity, a non-linear component is added to the propagation and thus the final hidden feature representation is expressed as:

$$H^{(l)} = \sigma\tilde{H}$$

where σ is the non-linear activation function such as ReLU, sigmoid, tanh and more.

Based on the structure of our data, we consider a two-layer convolution operation similar to (Kipf & Welling, 2017). The schematic diagram (Figure 17) shows the two-layer GCN utilized in this master thesis.

The overall forward model takes the following forms:

$$H^{(1)} = f(H^{(0)}, A) = \sigma_1\left(\tilde{A}H^{(0)}W^{(0)}\right),$$
$$H^{(2)} = f(H^{(1)}, A) = \sigma_2\left(\tilde{A}H^{(1)}W^{(1)}\right)$$

combining the above two equations, the compact form of the representation is as follows:

$$H^{(2)} = \sigma_2(\tilde{A}\sigma_1(\tilde{A}H^{(0)}W^{(0)})W^{(1)}) \tag{20}$$

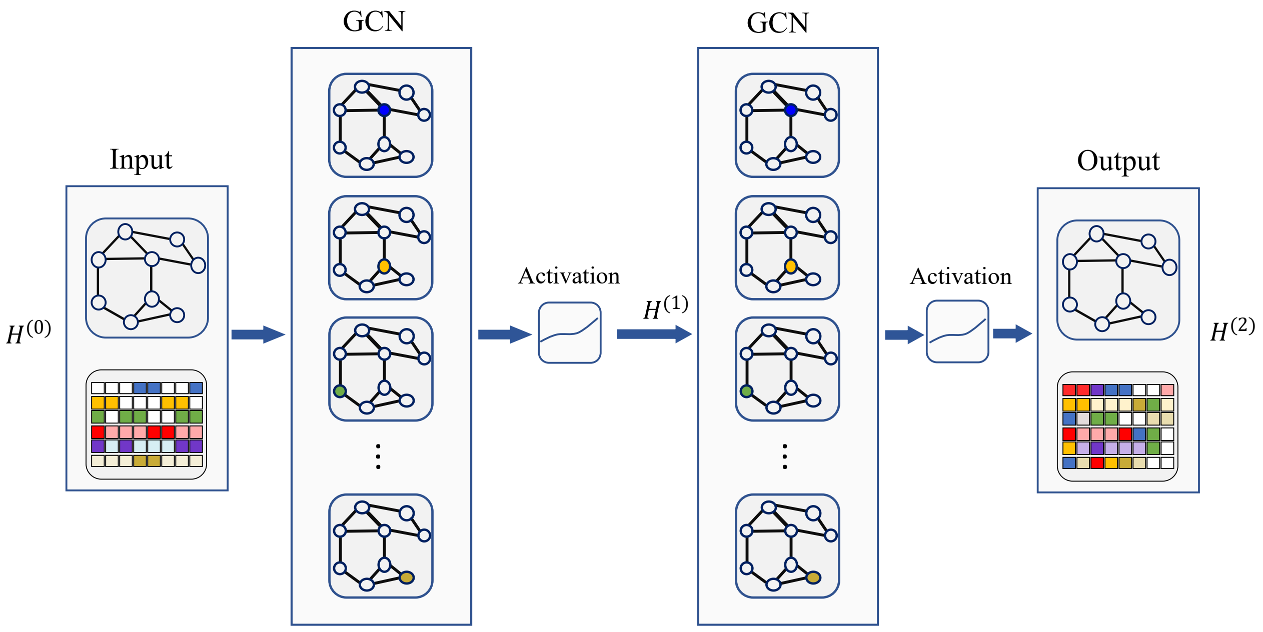


Figure 17: Overview of how a single node aggregates messages from its local

Tanh and Relu are selected as the activation functions (σ_1 and σ_2) respectively. The description of these activation functions are described in section 2.2.3.

b) Graph Attention Network (GAT)

In this model, we consider the attention mechanism in the feature propagation step. The attention mechanism proposed by Veličković et al., 2018 calculates the graph attention coefficient and adds it to the GCN operation. For details, the reader is referred to section 2.3.3. Similar to GCN model implementation, the GAT application could also be simply described in three steps, however, with adding a relation coefficient matrix.

Feature propagation: In each layer, we take the weighted average of the feature vectors of the nodes' neighbors.

$$H^{(l)} = \tilde{R}H^{(l-1)}$$

where $\tilde{R}$ is the relation matrix.

Feature transformation: The learnable weigh matrix is added, which transforms the hidden feature embeddings to the next layer.

$$\tilde{H} = H^{(l)}W^{(l)}$$

where $\tilde{H}$ is the hidden layer representation.

Activation layer: A non-linearity component is added to the propagation and thus the final hidden representation is expressed as:

$$H^{(l)} = \sigma\tilde{H}$$

where σ is the non-linear activation function.

Similar to section 3.2.2, we utilized a two-lyer GAT model, and thus GCN operation can take the following form:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{R}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}) \tag{21}$$

where σ is the activation functions, $W^{(0)}$ and $W^{(1)}$ are the two learnable transformation matrices, and $\tilde{R} = mask(R) + I_N$ is the relation matrix. The mask is used to sparsify the relation matrix as follows:

$$mask(R) = \begin{cases} R_{ij} & , if\ \tilde{A}_{ij} > 0 \\ 0 & , otherwise \end{cases} \tag{22}$$

3.3. Evaluation module

During the training process for both proposed models (GCN and GAT), we select Mean Absolute Percentage Error (MAPE) as a loss function.

MAPE measures the percentage of average absolute error in comparison to the predicted value (how large is the difference between the predicted and actual values in comparison to the predicted value, see equation 25).

In addition, in this master thesis, we select three evaluation matrices namely: (i) Mean Absolute Error (MAE), (ii) Root Mean Square Error (RMSE), and (iii) Mean Absolute Percentage Error (MAPE) to evaluate the proposed models. MAE and RMSE are used to estimate the absolute error between the actual and predicted values, where RMSE is more sensitive in capturing large errors. On the other hand, MAPE is used to measure the estimation accuracy based on the percentage error. For these three matrices, the lower values indicate the better performance of a model. The definition of each matrix is described as follows:

$$\boxed{RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \bar{Y})^2}} \tag{23}$$

$$\boxed{MAE = \frac{1}{N}\sum_{i=1}^{N}\left|Y_i - \bar{Y}\right|} \tag{24}$$

$$\boxed{MAPE = \frac{100\%}{N}\sum_{i=1}^{N}\left|\frac{Y_i - \bar{Y}_i}{Y_i}\right|} \tag{25}$$

where Y_i and $\bar{Y}_i$ are the predicted and true values of the i-th sample respectively, and N total number of predictions.

3.4. Model Transferability

We train both the proposed and baseline models (described in 4.7) with different datasets and evaluate the performance of each model. However, to check the generalization capability of the models, the best-trained model on the source data is selected and applied to a different dataset (we call it prediction dataset). The performance of each model is then assessed with the new dataset. This method is called transfer learning. Both the training datasets and the prediction dataset have the same number of features, however, the distribution of features varies among datasets.

4. Experimental Setup

In this section, we conduct an experiment to test the proposed models (see section 3.2.2) using simulated waiting time data. The following sections describe the study area, the extracted simulation-based data, the experiment settings, and methods for comparison which will be followed by results in the next chapter.

4.1. Study Area and Simulation Settings

In this master thesis, we utilize a synthetic MATSim model of the city of Cottbus to model ride-hailing demand. Cottbus is a city in the federal state of Brandenburg with around 100 000 inhabitants. The city is located roughly 110 km south of Berlin. The Cottbus transport network as shown in Figure 18, is comprised of 4470 nodes and 10729 links used for MATSim simulation. The population file contains 70000 agents and their work-related trips. To achieve more realistic results, the ride-hailing simulation runs are conducted together with the public transport lines which are operated within the city and its close surroundings.

Furthermore, this thesis uses the DRT extension of MATSim, which performs a centralized, on-the-fly assignment of vehicles to passengers as soon as the passengers' request to use the services. The ride-hailing is assumed to be door-to-door service, where the maximum waiting time of a passenger is set to 30 min. If a request is not served by any vehicles within the pre-defined constraint, the request is rejected and not considered in the simulation. In addition, vehicles are assumed to operate the whole day (24 h), without considering the consumed time for fueling or charging, operational issues or maintenance.

The data used for simulation runs as well as feature extraction in QGIS are collected from different sources. Cottbus network, agents plan file, and other MATSim-related files are gathered from Institut für Land- und Seeverkehr (ILS) - TU Berlin, and demographic information are used from the 2011 census data. In addition, OpenStreetMap is used as a base map in all maps of this master thesis.

4.2. Simulation Runs

As discussed in section 3.1.1, various simulation runs for different scenarios are conducted. On the demand side, 10% and 20% of all private trips are set to ride-hailing, whereas on the supply side, we have determined the optimum number of vehicles for both demand cases (10% and 20%) to be 1000 and 2000 vehicles respectively. Moreover, we run an additional simulation run with 15% demand and 1500 vehicles and extract another dataset (prediction dataset) to evaluate the transferability of the models.

First, for a 10% demand scenario, we select randomly 10% of agents from the population file and change their transport mode from private vehicle to taxi. To achieve the waiting time of all agents, we create 8 different sets of plans' files each with 10% taxi users, which contains 8*10% = 80% of all agents. The remaining 20% of agents use public transport for their daily activities, and therefore we do not change their mode of transport. Each of these 8 plan files together with one supply scenario (e.g., 750 taxis in the network) are simulated in MATSim. In total, 8*3 =24 different simulation scenarios have been conducted, and the sum of 24 trip files has been generated. Each trip file corresponds to 10% demand, and a certain supply in terms of the number of vehicles. A trip file contains a set of information (e.g., person ID, location, the start of trip, ...) for all generated trips during the simulation. Since we are interested in only taxi trips, we filter our data to include only the necessary information about taxis (e.g., request ID, location, waiting time,...). For each supply scenario, we concatenate 8 taxi trips data and as a result, we achieve the final taxi requests data for 80% of agents.

Similarly, for a 20% demand scenario, we select randomly 20% of agents from the population file and thus a total of 4 different sets of plans' files are created, and the same procedure is repeated. To exclude outliers from our data, we simply remove, waiting times which are less than a minute and more than 30 minutes. Finally, we create 6 different trips files (3 for 10% demand, and 3 for 20% demand), where each

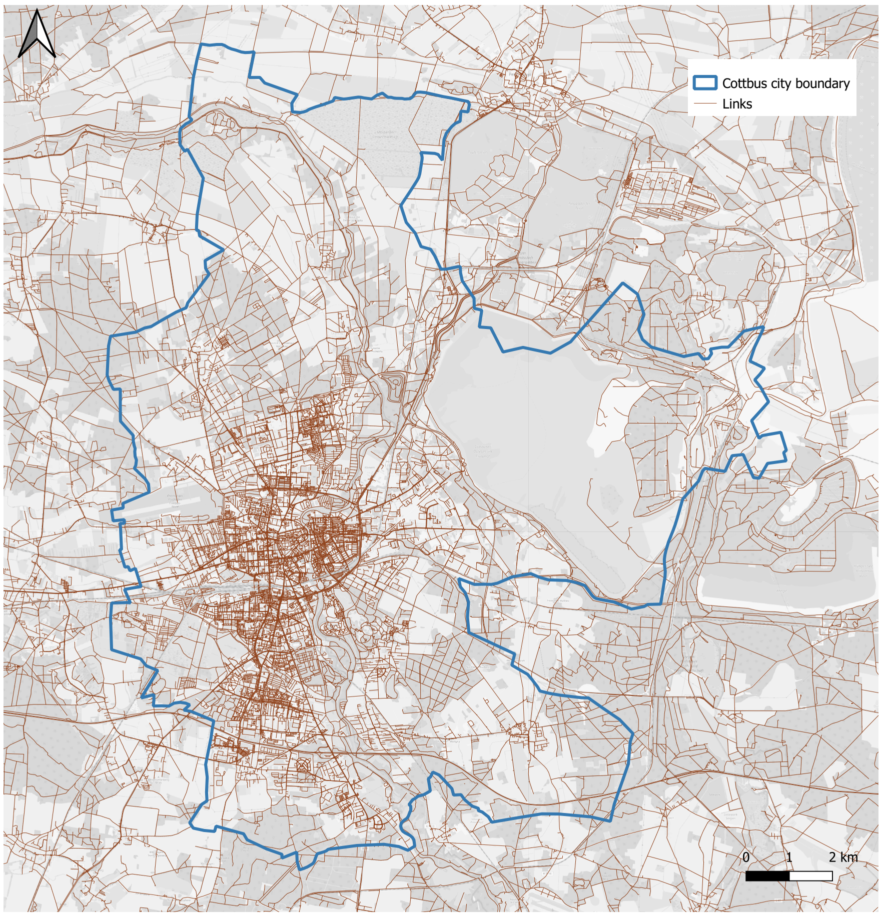


Figure 18: The Cottbus city road network.

is differentiated with the number of taxis in the network and these files are exported to QGIS for map matching.

4.3. Map Matching output

In the map matching process, we simply import 6 trip files and add them as trip points on the Cottbus network. Each point corresponds to a taxi trip request and the simulated waiting time. We create the grid cells (250*250) over the Cottbus network, and using the processing toolbox of QGIS (join attribute by location), we assign each trip point to a grid cell. The distribution of trip requests in grid cells is as shown in Figure 19. For visualization purposes, we only show the inner city map. Hence, for each trip file, a grid cell file is generated.

Similarly, we plot the nodes over the grid cells and assign each node to a grid. Consequently, the grid cell file includes information about the nodes and trip points located in each cell. Since it is possible that more than one trip point is located in a grid cell, we take the average of trip points in a cell and get the cell mean waiting time. Finally, each cell's waiting time corresponds to the node(s) waiting time located in that cell. The average waiting time distribution throughout the network using grid cells for two demand scenarios and associated optimal supply cases after map matching in QGIS is depicted in Figure 20.

On the hand, the taxi distribution file contains information about the links to which the taxis are located. Since the coordinates of taxis are not known, assigning a taxi to a grid cell is not possible. Therefore, we simply merge the taxi distribution file using the QGIS processing toolbox (join attribute by field value) with the links file, and thus the new link file will also have information on whether the link has a taxi or not. As a result, for node feature extraction, we use both the nodes' features extracted from the grid cells, as well as the information from the links file.

4.4. Features Generation

In the map-matching process, we successfully extract each node's waiting time information. As discussed in section 3.1.3, to allocate link features to a node, we assign the attributes of emerging links from a node using the QGIS processing toolbox (join attribute by field value). As a result, each node might contain features of different links including length, capacity, speed limit, travel time, average traffic flow, and vehicle availability. To have a unique features list for each node, we take the mean of features allocated to each node. Finally, the node file contains the feature (waiting time) from grid cells, and link attributes from the link file.

Furthermore, features such as population density and nodes' location-related information are independent of the

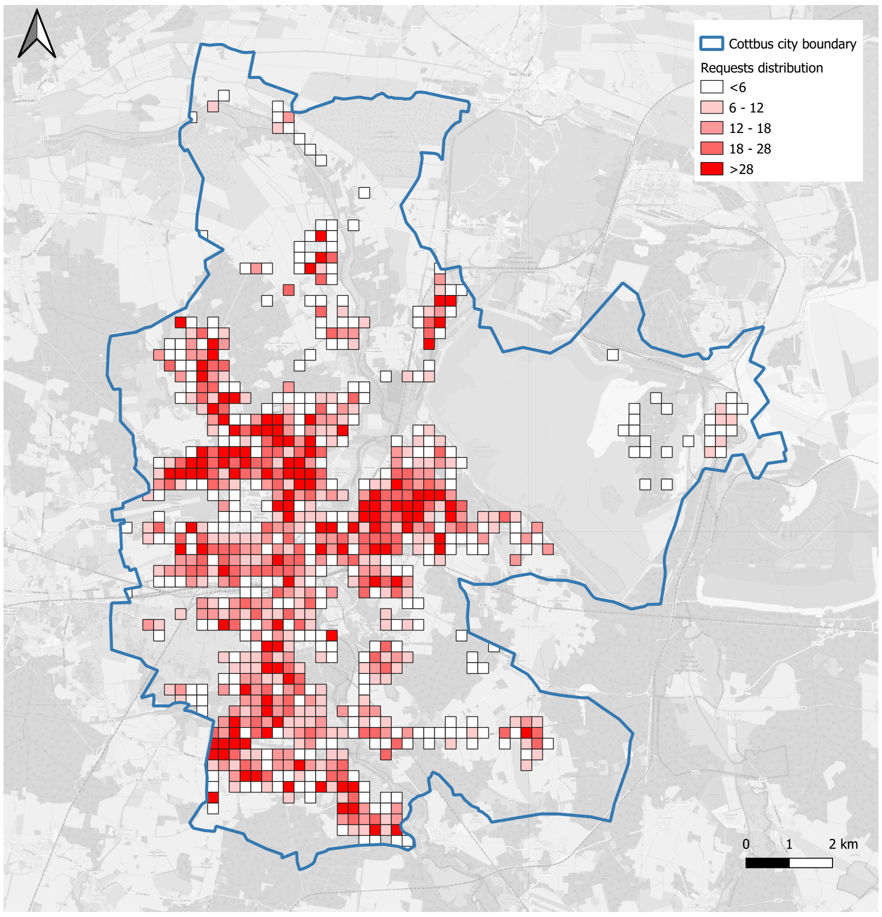
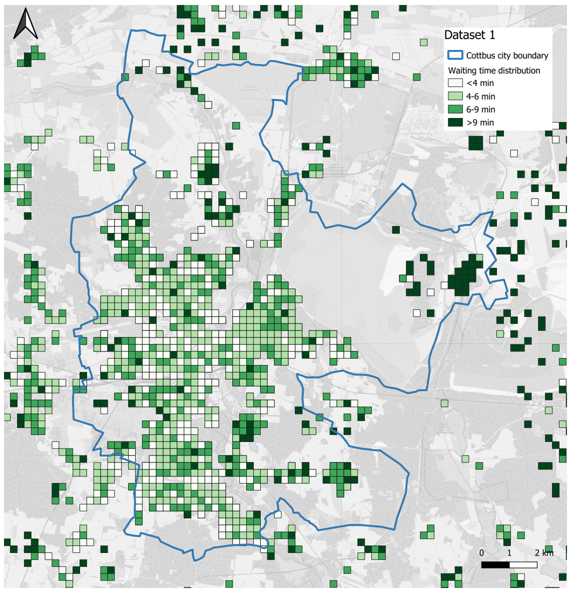
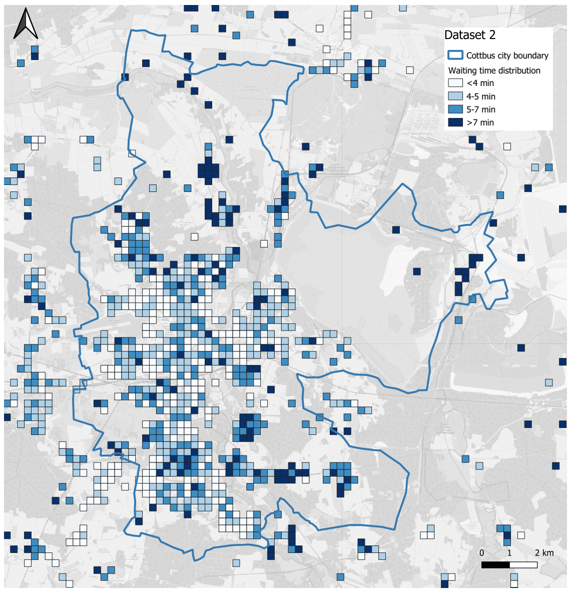


Figure 19: Illustration of the requests distribution under 20% demand and 2500 vehicles within Cottbus city.



(a)



(b)

Figure 20: The average waiting time in each grid cell under: (a) 10% demand and 1000 vehicle, and (b) 20% demand and 2000 vehicles.

simulation output. We utilize census data which contains the population density points corresponding to 100-meter grid cells for Cottbus city. To estimate the population density in our proposed grid cells (250*250), we import the census data and overlay them in our grid cells and simply sum the values of population points in each grid cell. Meanwhile, we extract the node-related information such as betweenness, closeness, degree, and more by using the processing toolbox of QGIS (network centrality) as displayed in Table 2. Worth-mentioning that only degree and closeness are considered in the final features set.

Finally, combined features of nodes could be categorized into constant and variable features as presented in Table 3. The constant features include node degree, closeness, the average length, capacity and speed limit of the links emerging from a node, and average population density around a node, where variable features depends on each simulation scenario. The variable features contain ride-hailing vehicles distribution, average travel time and traffic flow of the emerging links from a node.

We train and test the proposed models, with a total of 6 datasets each containing the ride-hailing demand and the number of ride-hailing vehicles. Table 4 shows part of data for 10% demand and 1000 ride-hailing vehicles.

Meanwhile, we investigate the correlation among the features as well as between features and the waiting time for 10% and 20% demand datasets. As shown in Figure 21, there is not a strong correlation between the features and the waiting time in both datasets. However, length and travel time as well as capacity and speed limit are strongly correlated with each other.

In addition, to have a clear understanding of the waiting time distribution in each demand and supply scenario, we plot the histogram of each in Figure 22.

4.5. Data Transformation

After successful map matching of the ride-hailing request in QGIS and extraction of the nodes' features, the data is ready for the GCN and GAT implementation. However, the data will be further processed to easily implement them in the deep learning pipeline. First, we load each dataset as our node data, and network links as our edge data in python. The node data includes node features and target values (waiting times). We convert node features and target values to Numpy arrays. Numpy is a Python library that allows easy and efficient manipulation of arrays (Harris et al., 2020). The Numpy arrays are required, since in most machine learning techniques, they are the input of model. In addition, we use row-normalization technique to normalize the features data. A sample of transformed data is depicted in Table 5.

4.6. GCN and GAT settings

For the GCN model, We train a three-layer GCN as presented in section 3.2.2, and evaluate the performance of the model with our datasets. We select randomly 40% of the dataset for training, 30% for validation, and the rest for testing. In addition, we integrate an optimization module (differential evolution) to find the best set of hyperparameters.

Differential evolution (DE) is a stochastic population-based optimization method which is used for global optimization problems. DE does not require gradient information and hence could be efficiently used for nonlinear optimization problems (Georgioudakis & Plevris, 2020). The algorithm iteratively searches the design space to improve a candidate solution with regard to pre-defined targets. The candidate solution moves around the design space to check whether an improvement for the objective function is achieved. In case, a new candidate solution outperforms its parent, it replaces the parent, otherwise, it's simply discarded. In this thesis, the value of the objective function is the loss value of the validation, where the input variables are the hyperparameters. DE tries to change the hyperparameters within their boundary conditions aiming to find the minimum loss value for validation of the model.

The outcome of the optimization module for a 100 number of epochs depicts 53 hidden units, 0.0096 learning rate, 0.5 dropout rate for all layers, and $4e-5$ for L_2 regularization factor for the first layer. However, for simplicity, we use 64 as the number of hidden units, 0.01 learning rate, and 0.5 dropout rate.

Similarly, for the GAT model, we train a two-layer GAT model. The split of the dataset for training, validation, and testing remains the same as in the GCN model. Moreover, we use the optimized hyperparameters obtained for the GCN model to the GAT model, with additional hyperparameters for GAT namely as $K = 3$ number of attention heads, and $\alpha = 0.2$ for leakyReLU activation function.

Both models take features and adjacency matrix as inputs and predict numerical values as output. These numerical values are compared with the target values (true waiting times) until the loss function is minimized. For both models, we utilize Adam optimizer for minimization of the loss function. Regarding parameters analysis, we perform a sensitivity analysis of different number of layers of GCN and number of hidden units in regards to the GCN model performance as well as the number of attention heads and number of hidden units for GAT model.

4.7. Methods for comparison

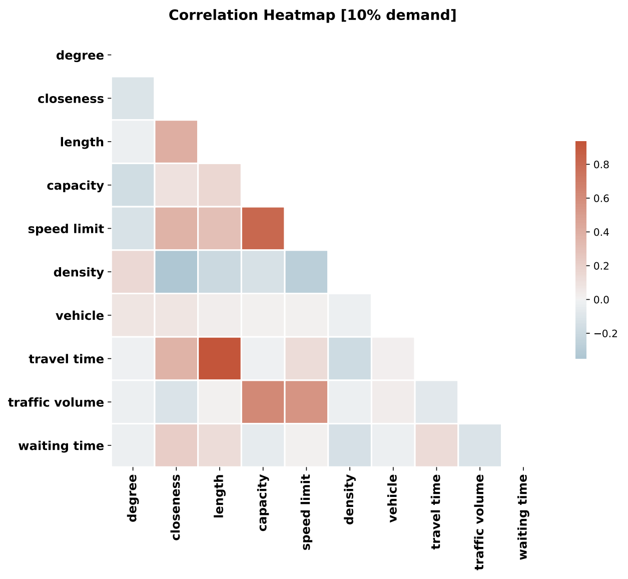
To verify the performance of the proposed models, we compare them with two baselines namely: (i) regression, and (ii) MLP models. The simple regression model takes the features as the independent variables and the waiting time as the dependent variable and creates a relation between them. On the other hand, Multiple layer perceptron (MLP) is a simple fully connected neural network model which takes the features as input and predicts the waiting time. In our experiments, we utilize three number of layers and ReLU as the activation function. The number of hidden units is same as in GCN and GAT models.

4.8. Software and Tools

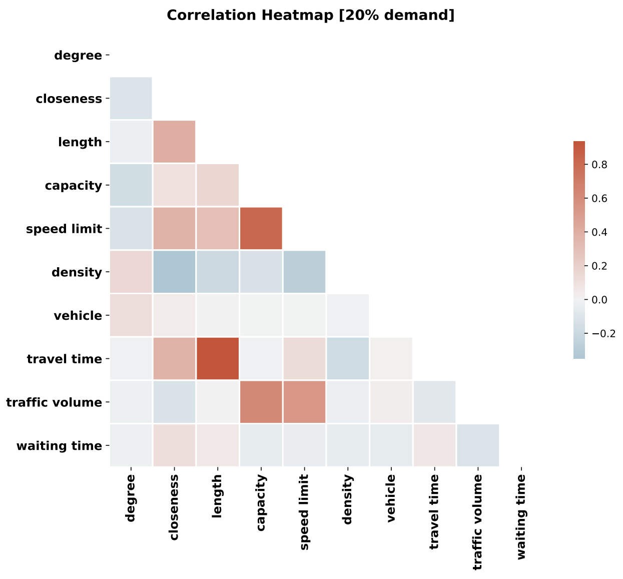
In this master thesis, we use Python programming language to process the data and implement the proposed models. Python is a powerful programming language that has an extensive collection of libraries for data processing, computation, and visualization. The machine learning frameworks

Table 2: Description of the centrality features of the network nodes.

Node	degree	closeness	betweenness	eigenvector	xcoord	ycoord
60	0.000447	22629.9376	0	0	452370.2496	5747706.58
61	0.000447	7881.35775	0	0	447532.3292	5733172.71
62	0.000893	12573.66711	-5.56192E+12	0	452999.6389	5734564.663
63	0.00134	13370.23422	-5.74345E+12	0	455877.7776	5731834.515
64	0.00067	11899.66277	-4.2045E+12	0	454049.0625	5733308.502
65	0.00134	13336.2458	-3.51704E+12	0	455864.8711	5731921.305
66	0.00134	13520.31327	-5.92705E+12	0	456027.2457	5731850.125



(a)



(b)

Figure 21: Correlation matrix among features in (a) 10% demand and 1000 vehicles, and (b) 20% demand and 2000 vehicles datasets.

Table 3: Description of the nodes' features

Constant features	Variable features
D: degree Cl: closeness L: average length C: average capacity S: average speed limit P: population density	V: vehicle distribution T: average travel time F: average traffic flow

have been widely implemented in Python among the scientific community. The list of the prominent python libraries and tools used in this master thesis is as follows:

1. Computation: Numpy
2. Data handling and manipulation: Pandas
3. Visualization and plotting: Matplotlib, Seaborn
4. Machine learning: PyTorch

The hardware used for the study is a 2020 Lenovo Flex 5 with i7 processor and 16 GB RAM as well as a Desktop PC with almost the same specifications.

5. Experiment Results

In this chapter, the main findings of this master thesis are presented. First, we investigate the learning process of the implemented models by analyzing the convergence of loss functions for each model and in each dataset. Second, we evaluate the performance of the proposed models and compare them with regression and MLP models. Third, the findings of the analysis of the parameters under different hyperparameters settings for GCN and GAT models are presented. Finally, we analyze the transferability of each model, which are trained with the 10% and 20% demands and optimal supplies.

For simplicity, in the first and third sections, we only depict the plots for two datasets namely: (i) dataset 1, which include the 10% ride-hailing demand and the optimal number of vehicles (1000 vehicles) scenario, and (ii) dataset 2, which is the data under 20% ride-hailing demand and 2000 vehicles.

Table 4: Node features and waiting time values for a 10% demand scenario and 1000 ride-hailing vehicles

Node	D	Cl	L	C	S	P	V	T	F	W*
60	0.00045	22629.94	461.88	600.00	12.50	1.00	0.00	36.95	0.00	457.19
61	0.00045	7881.36	345.20	60.00	14.00	23.00	0.00	24.66	0.00	457.19
62	0.00089	12573.67	336.13	60.00	14.00	190.00	0.00	24.01	0.00	218.50
63	0.00134	13370.23	118.88	600.00	8.33	118.00	0.00	14.33	9.37	532.05
64	0.00067	11899.66	160.14	1800.00	15.28	80.00	1.00	11.27	1011.45	106.50
65	0.00134	13336.25	174.37	600.00	8.33	32.00	0.00	21.04	10.45	568.00
66	0.00134	13520.31	61.03	600.00	8.33	166.00	0.00	7.32	2.46	453.50
67	0.00134	12053.81	226.48	600.00	9.72	10.00	0.00	25.47	56.97	207.80
68	0.00134	12457.21	152.56	600.00	8.33	453.00	0.00	18.39	3.41	150.80
69	0.00089	11985.13	64.71	1800.00	17.36	23.00	1.00	4.00	1023.85	239.90
70	0.00134	11981.92	35.75	1400.00	15.74	23.00	1.00	2.67	717.53	239.90

W^*: average waiting time

Table 5: Normalized node-features for a 10% demand scenario and 1000 ride-hailing vehicles

Node	D	Cl	L	C	S	P	V	T	F
60	1.88E-08	0.9531	0.0195	0.0253	0.00053	4.21E-05	0.00000	0.001556	0.00000
61	5.35E-08	0.9441	0.0414	0.0072	0.00168	2.76E-03	0.00000	0.002954	0.00000
62	6.77E-08	0.9527	0.0255	0.0045	0.00106	1.44E-02	0.00000	0.001819	0.00000
63	9.41E-08	0.9390	0.0083	0.0421	0.00059	8.29E-03	0.00000	0.001007	0.00066
64	4.47E-08	0.7944	0.0107	0.1202	0.00102	5.34E-03	0.00007	0.000752	0.06753
65	9.45E-08	0.9403	0.0123	0.0423	0.00059	2.26E-03	0.00000	0.001483	0.00074
66	9.33E-08	0.9412	0.0042	0.0418	0.00058	1.16E-02	0.00000	0.000510	0.00017
67	1.03E-07	0.9285	0.0174	0.0462	0.00075	7.70E-04	0.00000	0.001962	0.00439
68	9.79E-08	0.9098	0.0111	0.0438	0.00061	3.31E-02	0.00000	0.001343	0.00025
69	5.99E-08	0.8033	0.0043	0.1207	0.00116	1.54E-03	0.00007	0.000268	0.06863
70	9.45E-08	0.8451	0.0025	0.0987	0.00111	1.62E-03	0.00007	0.000188	0.05061

5.1. Convergence Analysis

The initial results of this study reveal that in each model, the loss function (MAPE) successfully converges and learns the learnable weights. However, the speed of converges and the value of the loss function after 300 epochs varies depending on the model. For the GCN model, the loss function in training phase reduces from 1 to 0.35 after 300 epochs under dataset 1, and from 1 to 0.40 in dataset 2. For both scenarios, the loss function begins converging after 70 epochs in training phase as shown in Figure 23 (a) left , where in validation phase, the loss function converges already after 60 epochs (see Figure 23 (a) right). In comparison to training phase, where the fluctuation in loss function still exists after 70 epochs, in validation phase, loss function does not fluctuate after 60 epochs. This determines how fast the model learns the parameters.

Similarly, for GAT model, the loss function decreases from 4 to 0.37 during training phase in dataset 1, and from 5 to 0.39 when implementing dataset 2. In comparison to GCN model, GAT model learns to converge later and after around 150 epochs (see Figure 23 (b) left). The same could be seen for validation, the loss function converges after 150 epochs as depicted in Figure 23 (b) right. In addition, the loss function fluctuates higher than in GCN model. Although, GCN model is simple to implement, but able to learn faster and achieve higher performance.

On the other hand, for the regression model, the loss function starts from 18 and converges to 0.44 in dataset 1, and from 24 to 0.45 in dataset 2. For visualization reason, the loss values after epoch 25 is displayed in Figure 24. The model converges after 130 epochs in the training phase, and 125 epochs in the validation phase as depicted in Figure 24 (a).

Furthermore, the loss function reduces from 1 to 0.38 in dataset 1, and from 1 to 0.40 in dataset 2 when implementing the MLP model as displayed in Figure 24 (b). The loss function converges after 100 epochs in the training and validation phases. In comparison, to the regression model, the loss function fluctuates higher in MLP model.

5.2. Model Evaluation

Comparing the performance of GCN and GAT models considering regression and MLP models as baselines, we can find that regression model has a weaker performance than deep learning models. The reason might be the due to the linearity of regression model as well as the limited capabilities of regression model in capturing the complex structure of graph data. On the other hand, GCN and GAT model show better

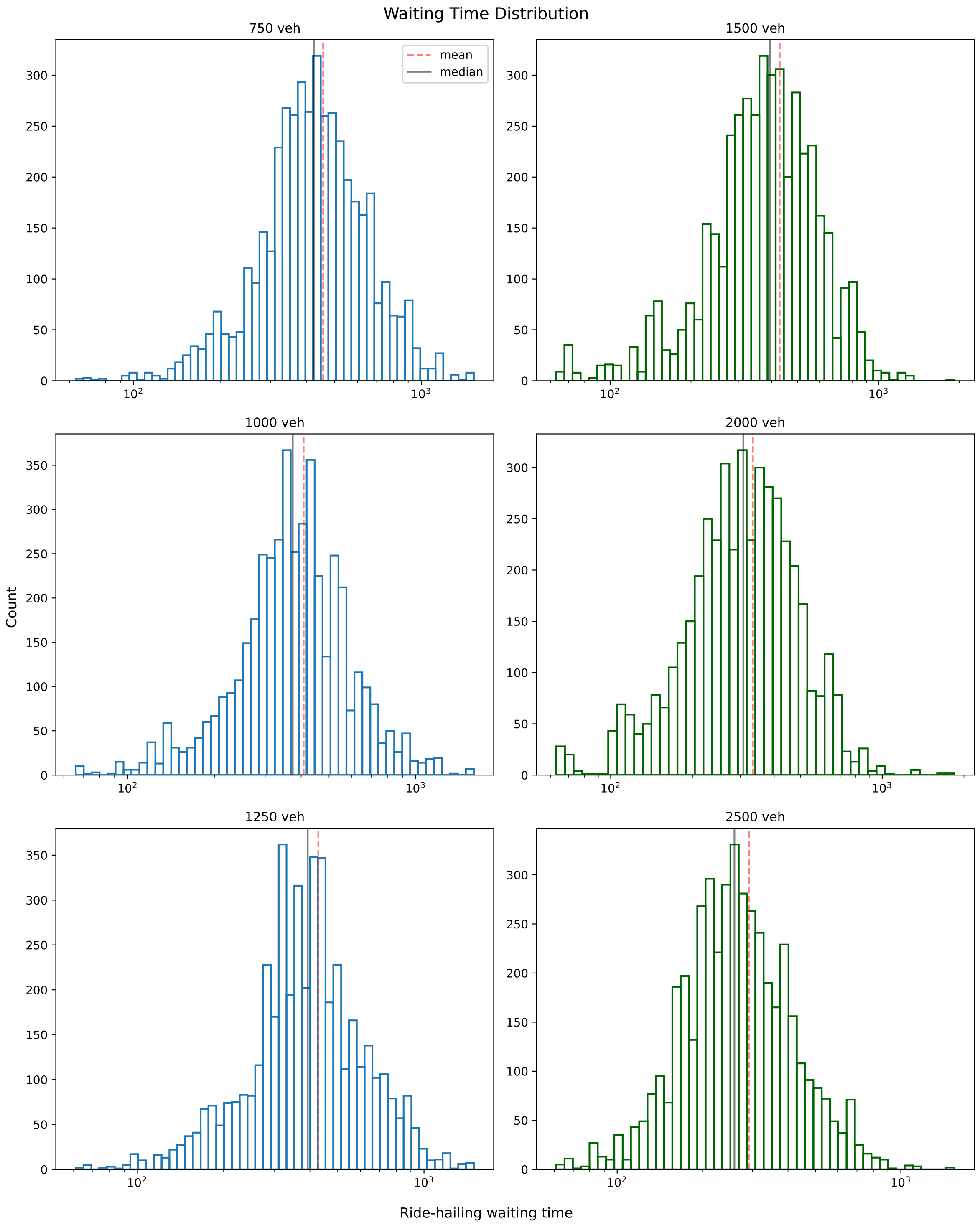
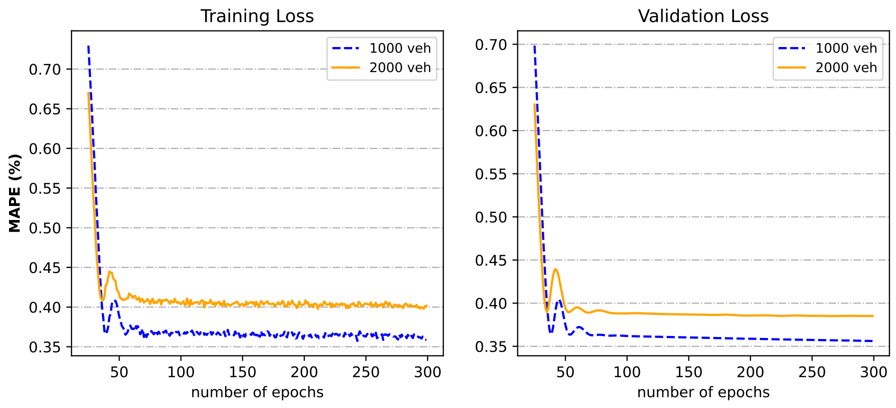


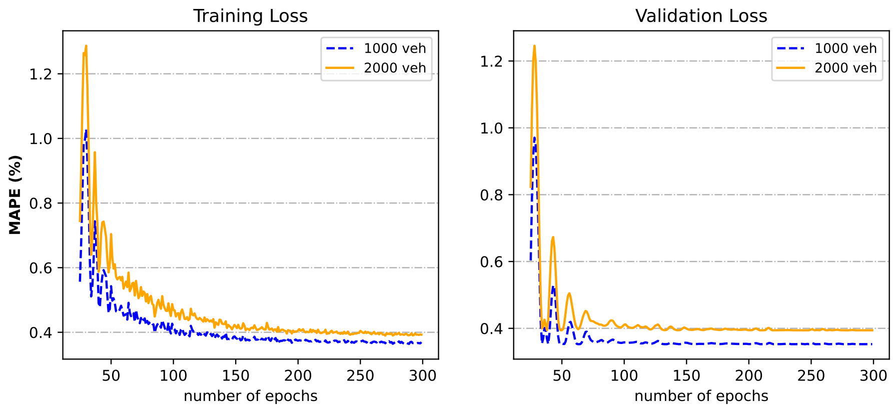
Figure 22: Waiting time data variation with 10 % demand on the left , and 20 % demand on the right under three different supply scenarios.

performance in all datasets. First, the findings of the models performance for 10% demand and three different supply scenarios depict that GCN model outperforms regression model as an average of 21.5%, 17.5% and 11.4% in MAPE, MAE, and RMSE respectively as depicted in Table 6. The comparison of GCN and MLP models reveal that GCN has better performance approximately 3% in MAE and 2% in RMSE, however, MAPE does not change.

Regarding the GAT model, it outperforms regression model about 21.5%, 17.5% and 11.5% in three evaluation matrices respectively. Similar to GCN, GAT shows around 2.7% and 2.3% better performance than MLP in MAE and



(a)



(b)

Figure 23: Training and validation losses for (a) GCN model, and (b) GAT model under 10% and 20% demand scenarios

RMSE, where no improvement has achieved for MAPE. In addition, the comparison of GCN and GAT models performance, it is found that both have almost the same performance (see Table 6).

Furthermore, the findings for the 20% demand scale and three supply scenarios reveal that the GCN model outperforms the regression model by an average of 15.4%, 16.4%, and 10.2% in MAPE, MAE, and RMSE respectively. However, when compared with MLP, GCN show around 1% in MAPE, 5.5% in MAE, and 5.3% in RMSE better performance. Meanwhile, the GAT model outperforms the regression model by an average of 13.8%, 13.4%, and 10.2% in MAPE, MAE, and RMSE respectively. However, when comparing with MLP, GAT shows 2.3% in MAE, and 2.5% in RMSE higher performance, whereas MAPE depicts a slightly higher loss value as shown in Table 7.

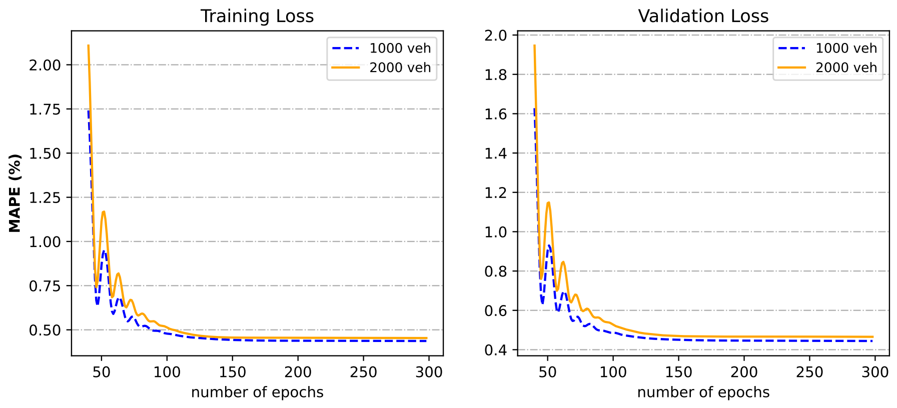
Furthermore, the analysis of the evaluation matrices reveals that by increasing the supply in terms of the number of vehicles, the overall models' performance improves. This also applies considering both demand scales. As depicted in Tables 6, and 7, all models have the best performance in the dataset with 20% demand and 2500 vehicles supply. On the hand, increasing the number of vehicles in the network results in decreasing the average waiting time of the network as well as the spread of the data in terms of standard deviation as displayed in Table 8. Hence, a less dispersion in the dataset (e.g., waiting time values) might best match with real-world data, and could be better linked with spatio-operational features of the graph.

5.3. Sensitivity Analysis

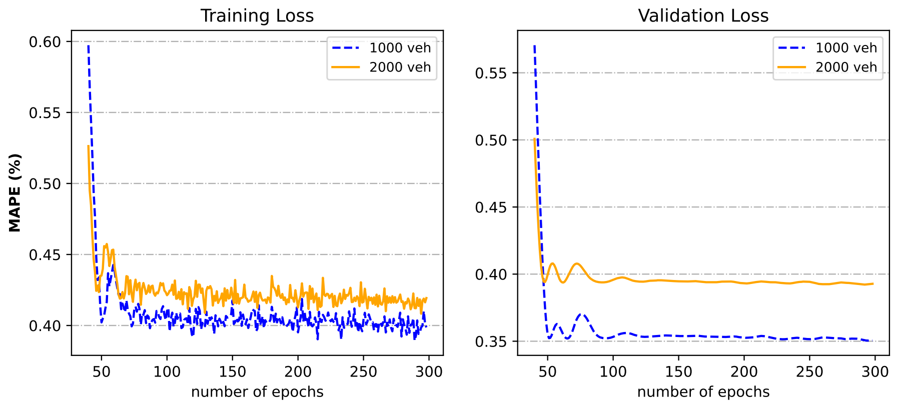
To investigate the effectiveness of different parameters in the performance of a model (e.g., GCN, GAT models), we conducted several experiments under various parameters settings. For GCN model, we choose number of GCN-layer, and the number of hidden units. On the other hand, number of attention heads and number of hidden units are selected for sensitivity analysis of GAT model performance.

First, fixing the number of hidden units ($n = 64$) , we run GCN model by changing the number of layers from 1 to 8. The findings reveal that the change in number of layers in GCN model does not have huge impact in performance of the model in all evaluation matrices. Still, with three-layers, GCN shows better performance when considering the datasets, and matrices as depicted in Figure 25.

Second, we fix the number of GCN-layer ($K = 3$), and change the number of hidden units to [4,8,16,32,64,128].



(a)



(b)

Figure 24: Training and validation losses for (a) regression model, and (b) MLP model under 10% and 20% demand scenarios

Table 6: Performance of the proposed models in comparison to baseline models for estimation of waiting time under 10% ride-hailing demand and supply scenarios.

Scenario	- 20% optimal supply			optimal supply			+ 20 optimal supply		
Method	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE
Regression	0.42	182.77	239.07	0.44	170.11	223.27	0.44	183.32	244.80
MLP	0.32	150.94	209.88	0.35	148.08	208.09	0.35	155.75	221.40
GCN	0.32	146.95	204.10	0.35	144.56	206.43	0.35	150.83	215.04
GAT	0.32	146.60	204.54	0.35	144.40	203.35	0.35	151.60	216.66
Improvement (%)									
*GCN**	**23.8**	**19.6**	**14.6**	**20.5**	**15.0**	**7.5**	**20.5**	**17.7**	**12.2**
*GCN***	**0.0**	**2.6**	**2.8**	**0.0**	**2.4**	**0.8**	**0.0**	**3.2**	**2.9**
*GAT**	**23.8**	**19.8**	**14.4**	**20.5**	**15.1**	**8.9**	**20.5**	**17.3**	**11.5**
*GAT***	**0.0**	**2.9**	**2.5**	**0.0**	**2.5**	**2.3**	**0.0**	**2.7**	**2.1**

(*, **) Comparison with regression and MLP models respectively.

As shown in Figure 26, by increasing the number of hidden units, the model performance improves in all scenarios. However, the slope of the change in GCN model performance is different with regards to the change in the number of hidden units. For instance, the change in number of hidden units from 8 to 16 has higher impact on the model performance than the change from 32 to 64 units. Meanwhile, the training time increase with higher number of hidden units. Hence, we found out that a 64 number of hidden units is a an optimal choice for GCN model evaluation.

As displayed in figures 25, and 26, the change in the number of units has more impact on the model performance in all matrices in comparison to the change in the number of GCN layers.

Table 7: Performance of the proposed models in comparison to baseline model for estimation of waiting time under 20% ride-hailing demand and supply scenarios.

Scenario	- 20% optimal supply			optimal supply			+ 20 optimal supply		
Method	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE
Regression	0.42	184.30	238.80	0.47	155.28	206.70	0.40	119.11	160.17
MLP	0.37	154.17	213.83	0.39	138.58	194.10	0.34	110.78	162.04
GCN	0.39	153.40	211.41	0.38	125.79	177.76	0.32	103.13	151.88
GAT	0.38	151.13	208.20	0.39	135.33	189.14	0.34	108.00	157.91
Improvement (%)									
*GCN**	**7.1**	**16.8**	**11.5**	**19.1**	**19.0**	**14.0**	**20.0**	**13.4**	**5.2**
*GCN***	**-5.4**	**0.5**	**1.1**	**2.6**	**9.2**	**8.4**	**5.9**	**6.9**	**6.3**
*GAT**	**9.5**	**18.0**	**12.8**	**17.0**	**12.8**	**8.4**	**15.0**	**19.3**	**1.4**
*GAT***	**-2.7**	**2.0**	**2.6**	**0.0**	**2.3**	**2.4**	**0.0**	**2.5**	**2.5**

(*, **) Comparison with regression and MLP models respectively.

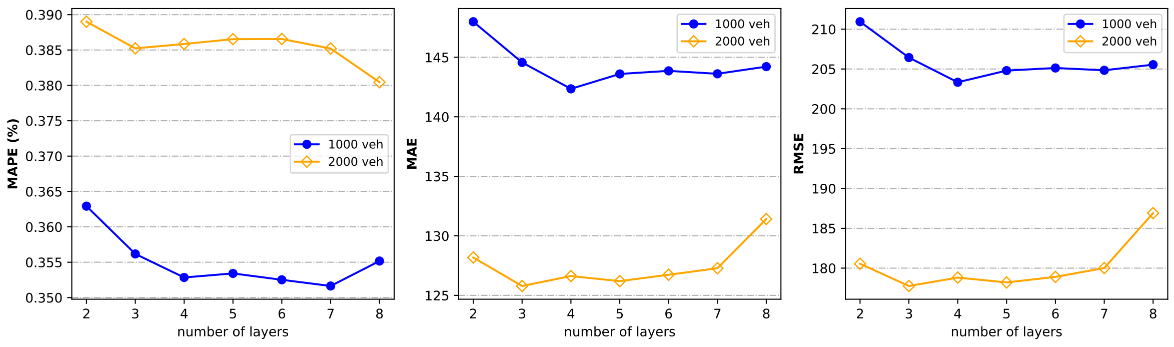


Figure 25: Variation of the GCN model performance with different number of layers under the optimal supply scenarios (1000 and 2000 vehicles) for both demand scenarios (10% and 20%) respectively.

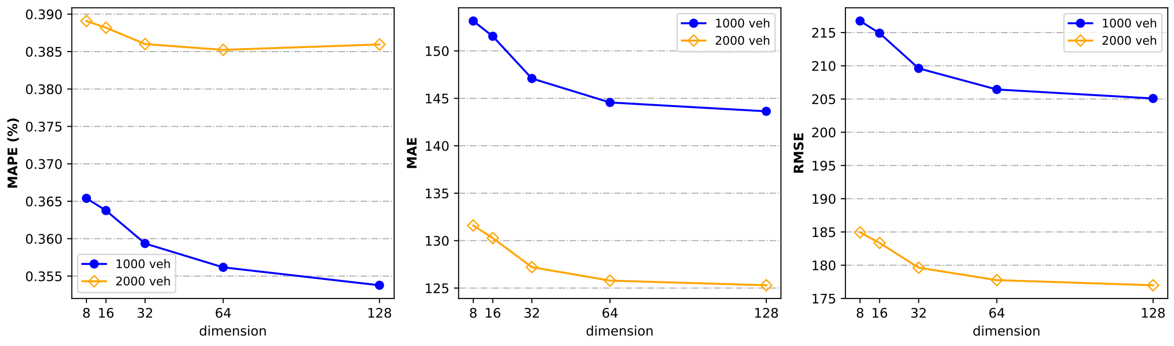


Figure 26: Variation of the GCN model performance with different number of hidden units under the optimal supply scenarios (1000 and 2000 vehicles) for both demand scenarios (10% and 20%) respectively.

Regarding the GAT model, first, we fix the number of hidden units to (n=64) and change the number of attention heads from 1 to 6. The findings show that does not change significantly, however, MAE and RMSE reduce considerably when the number of attention heads is set to 2 in all scenarios. Meanwhile, increasing the number of attention heads results in higher MAE and RMSE as depicted in Figure 27.

Meanwhile, the change in the number of hidden units [4,8,16,32,64,128] while keeping the number attention heads to (K =2), depicts that a higher number of hidden units results in poor performance in evaluation matrices and in both datasets. Similarly, a higher number of hidden units increases the training time, and thus selection of hidden units to 64 might be an optimal choice considering all evaluation matrices.

5.4. Model Transferability Analysis

The model performances on prediction data are shown in Table 9 and 10. Depending on the data used for training the models, different findings are extracted. For instance,

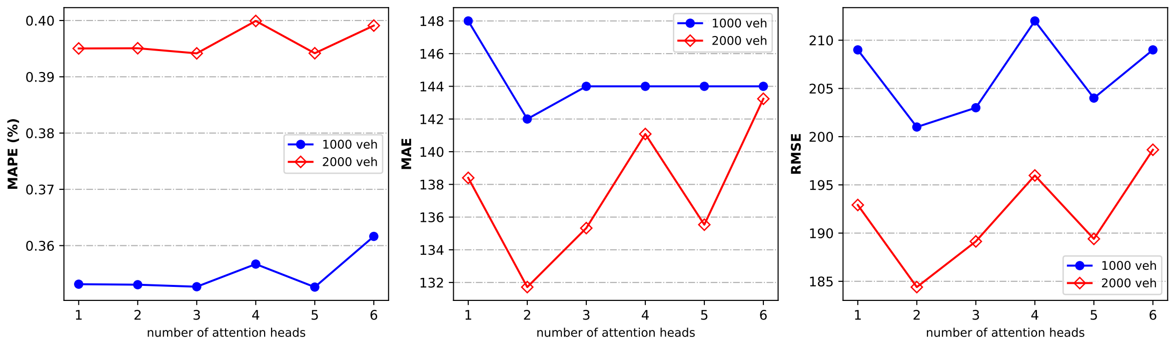


Figure 27: Variation of the GAT model performance with different number of attention heads under the optimal supply scenarios (1000 and 2000 vehicles) for both demand scenarios (10% and 20%) respectively.

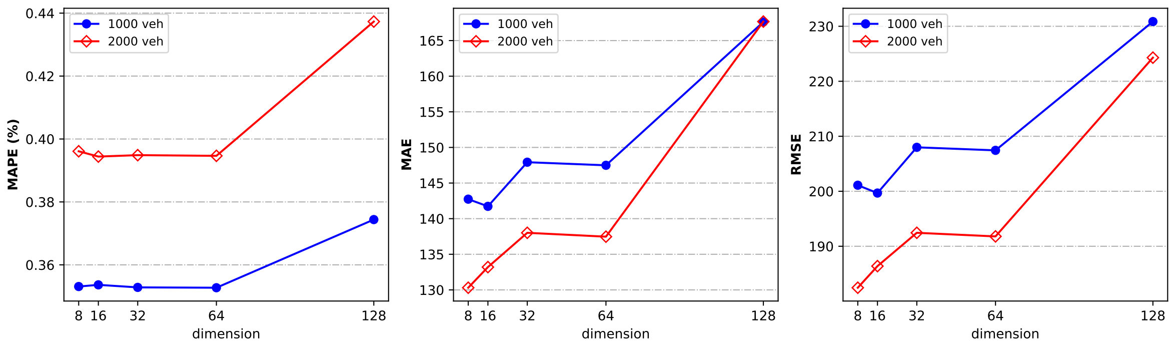


Figure 28: Variation of the GAT model performance with different number of hidden units under the optimal supply scenarios (1000 and 2000 vehicles) for both demand scenarios (10% and 20%) respectively.

Table 8: The description of mean and standard deviation of waiting time in each dataset.

Dataset	Mean	St.dev
750	456	194
1000	408	187
1250	428	198
1500	407	181
2000	334	160
2500	292	139

when the models are trained with the dataset 1, their performances on prediction data show improvements in most of evaluation matrices. As shown in Table 9, all models depict better performance in MAE and RMSE, where in MAPE, they show higher values. On the other hand, when the models are trained with the dataset 2, their performances in the prediction dataset deteriorate in all evaluation matrices as displayed in Table 10.

6. Conclusion

This chapter presents a conclusion of the contents of the whole thesis with the main findings as well as the future outlooks derived from this study.

In this thesis, we aim to estimate the ride-hailing requests' waiting time using the graph neural network (GNN) and considering the spatio-operational features of the transport network. The study begins with a comprehensive literature review, where we review the agent-based simulation models as well as the basics of neural networks and GNNs. The objectives are: (i) to choose a suitable simulation tool and settings for modeling the ride-hailing model and extract waiting time data, and (ii) to find which GNN-based approach(es) could predict the ride-hailing waiting time considering the features of the transport network data.

The methodology of this master thesis contains four main parts namely: (i) data generation, (ii) the proposed framework, (iii) the evaluation scheme, and (iv) model transferability. Multi-source data including the transport network, and agents plan data for running simulations, and OSM data, population density, and more are utilized for features' extraction in this master thesis. The final nodes data including the impacting features and the links information are prepared for the GNN implementation. In light of the application of different GNN-based approaches, GCN and GAT models are utilized to predict the waiting time in a service area. Both models take nodes' information together with each node's features as well as the relation between the nodes (links data) as inputs and predict the ride-hailing waiting time.

An experimental setup is developed to run MATSim-based

Table 9: Transferability analysis of the trained models with dataset 1 [10% demand and optimal supply]

	MAPE			MAE			RMSE		
Model	Eval	Pred	Change	Eval	Pred	Change	Eval	Pred	Change
Reg	0.44	0.52	(0.08)	170.11	146.14	23.97	223.27	199.37	23.90
MLP	0.35	0.47	(0.12)	148.08	126.05	22.03	208.09	181.75	26.34
GCN	0.35	0.50	(0.15)	144.56	125.63	18.93	206.43	175.32	31.11
GAT	0.35	0.49	(0.14)	144.40	126.89	17.51	203.35	180.27	23.08

Table 10: Transferability analysis of the trained models with dataset 2 [20% demand and optimal supply]

	MAPE			MAE			RMSE		
Model	Eval	Pred	Change	Eval	Pred	Change	Eval	Pred	Change
Reg	0.47	0.46	0.01	155.28	155.33	(0.05)	206.70	214.86	(8.16)
MLP	0.39	0.42	(0.03)	138.58	140.61	(2.03)	194.10	204.94	(10.84)
GCN	0.38	0.42	(0.04)	125.79	134.34	(8.55)	177.76	194.10	(16.34)
GAT	0.39	0.42	(0.03)	135.33	134.60	0.73	189.14	197.69	(8.55)

simulation runs under various demand and supply scenarios for the Cottbus city network. The extracted trips information together with other related features are concatenated and a total of 6 datasets are generated for the final implementation. In addition, regression and MLP models are selected as baselines for comparing the performance of the utilized models. We select MAPE, MAE, and RMSE matrices for the evaluation of each model.

The findings of the experiment test reveal that deep learning-based approaches have better performance than the regression model. For instance, GCN outperforms the regression model as an average of 15% in all datasets and evaluation matrices, whereas in comparison to MLP, GCN shows 3% better performance. Similarly, the GAT model depicts 14%, and 1.5% better performance than regression and MLP models respectively. In addition, it is found that all models have their best performance in the dataset with 20% demand scale and 2500 vehicles in terms of supply. Meanwhile, to test the effectiveness of the models' hyperparameters in regard to the performance of each model, a sensitivity analysis is carried out. The results depict that in the GCN model, the change in the number of GCN layers does not have a huge effect on the performance of the model, however changing the number of hidden units from low to high results in better performance of the model. On the other hand, the change in the number of attention heads in the GAT model does have a significant impact, however, by increasing the number of hidden units, the model performance deteriorates. Finally, the model transferability analysis shows that the models trained with the dataset 1 depict better performance in prediction dataset in comparison to models trained with the dataset 2.

The contribution of this master thesis successfully achieved the main goal of this study. We implemented several deep learning methods including regression model, MLP, GCN and GAT to estimate waiting time. Although GNN-based models are powerful in extracting graph data, their performance could be well-differentiated from other models (regression and MLP) with more complex data including several features. Meanwhile, there are some limitations in this master thesis and therefore it raises several new lines of work that could be pursued as valuable research topics in the future.

First, in this master thesis we used the waiting time data extracted from a simulation platform, however, utilization of real-world data for extraction of the graph features and further implementation of such data in the GCN and GAT models might have different outcomes. Second, in our data generation process, apart from waiting time information, and the traffic-related features, we also utilized population density as an extra feature that might have a direct impact on the ride-hailing waiting time. However, additional features such as land-use type, build-environmental characteristics (e.g., points of interest), public transport stops, and more impacting factors on waiting time could be considered. Thus, a study with more rich features might bring new insights into the waiting time prediction models. Third, our data is limited to spatio-operational features of the network. Each ride-hailing trip is requested in a specific location within a day. However, to include the impacts of traffic flow and congestion level, the temporal variation of the requests could be considered. For instance, a request waiting time in different time intervals should be added to the graph features. Using this data, which includes both the spatial and temporal variation of the request points, the STGNN model could be implemented to estimate the waiting time. Hence, a study could be conducted to extract such data from a microscopic simulation platform (e.g., Vissim, SUMO) and implement it in STGNN. Fourth, we used the features of the links to allocate node features. However, it is also possible to utilize a dual graph approach and directly conduct the prediction on links. Therefore, a study to transform a graph to its dual, and further do the prediction might have valuable outputs.

References

Ahmed, A., Shervashidze, N., Narayanamurthy, S., Josifovski, V., & Smola, A. J. (2013). Distributed large-scale natural graph factorization. *Proceedings of the 22nd International Conference on World Wide*

Web - WWW '13, 37–48. https://doi.org/10.1145/2488388.2488393

Ahmed, H. U., Huang, Y., & Lu, P. (2021). A Review of Car-Following Models and Modeling Tools for Human and Autonomous-Ready Driving Behaviors in Micro-Simulation. *Smart Cities*, *4*(1), 314–335. https://doi.org/10.3390/smartcities4010019

Anderson, D. N. (2014). "Not just a taxi"? For-profit ridesharing, driver strategies, and VMT. *Transportation*, *41*(5).

Bischoff, J., Maciejewski, M., & Nagel, K. (2017). City-wide shared taxis: A simulation study in Berlin. *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 275–280. https://doi.org/10.1109/ITSC.2017.8317926

Bishop, C. M. (1995). Neural Networks for Pattern Recognition. *Oxford University Press, USA*, 498.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, *34*(4), 18–42. https://doi.org/10.1109/MSP.2017.2693418

Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. (2013). Spectral Networks and Locally Connected Networks on Graphs.

Chen, C., Li, K., Teo, S. G., Zou, X., Wang, K., Wang, J., & Zeng, Z. (2019). Gated Residual Recurrent Graph Neural Networks for Traffic Prediction. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*, 485–492. https://doi.org/10.1609/aaai.v33i01.3301485

Chen, K., Deng, M., & Shi, Y. (2021). A Temporal Directed Graph Convolution Network for Traffic Forecasting Using Taxi Trajectory Data. *ISPRS International Journal of Geo-Information*, *10*(9), 624. https://doi.org/10.3390/ijgi10090624

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., & Song, L. (2018, February). Learning Combinatorial Optimization Algorithms over Graphs [Comment: NIPS 2017]. https://doi.org/10.48550/arXiv.1704.01665

de Souza Silva, L. A., de Andrade, M. O., & Alves Maia, M. L. (2018). How does the ride-hailing systems demand affect individual transport regulation? *Research in Transportation Economics*, *69*, 600–606. https://doi.org/10.1016/j.retrec.2018.06.010

Fang, X., Huang, J., Wang, F., Zeng, L., Liang, H., & Wang, H. (2020). ConSTGAT: Contextual Spatial-Temporal Graph Attention Network for Travel Time Estimation at Baidu Maps. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2697–2705. https://doi.org/10.1145/3394486.3403320

Fout, A., Byrd, J., Shariat, B., & Ben-Hur, A. (2017). Protein Interface Prediction using Graph Convolutional Networks. *Advances in Neural Information Processing Systems*, *30*.

Georgioudakis, M., & Plevris, V. (2020). A Comparative Study of Differential Evolution Variants in Constrained Structural Optimization. *Frontiers in Built Environment*, *6*, 102. https://doi.org/10.3389/fbuil.2020.00102

Gharaee, Z., Kowshik, S., Stromann, O., & Felsberg, M. (2021). Graph representation learning for road type classification. *Pattern Recognition*, *120*, 108174. https://doi.org/10.1016/j.patcog.2021.108174

Gilibert, M., & Ribas, I. (2019). Main design factors for shared ride-hailing services from a user perspective. *International Journal of Transport Development and Integration*, *3*(3), 195–206. https://doi.org/10.2495/TDI-V3-N3-195-206

Grau, J. M. S., & Romeu, M. A. E. (2015). Agent Based Modelling for Simulating Taxi Services. *Procedia Computer Science*, *52*, 902–907. https://doi.org/10.1016/j.procs.2015.05.162

Grover, A., & Leskovec, J. (2016, July). Node2vec: Scalable Feature Learning for Networks [Comment: In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016].

Hamilton, W., Ying, Z., & Leskovec, J. (2017). Inductive Representation Learning on Large Graphs. *Advances in Neural Information Processing Systems*, *30*.

Hamilton, W. L. (2020). *Graph Representation Learning* (tech. rep.).

Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Representation Learning on Graphs: Methods and Applications [Comment: Published in the IEEE Data Engineering Bulletin, September 2017; version with minor corrections]. https://doi.org/10.48550/arXiv.1709.05584

Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., et al. (2020). Array programming with numpy. *Nature*, *585*(7825), 357–362.

Henao, A., & Marshall, W. E. (2019). The impact of ride-hailing on vehicle miles traveled. *Transportation*, *46*(6), 2173–2194. https://doi.org/10.1007/s11116-018-9923-2

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hoff, P. D., Raftery, A. E., & Handcock, M. S. (2002). Latent Space Approaches to Social Network Analysis. *Journal of the American Statistical Association*, *97*(460), 1090–1098. https://doi.org/10.1198/016214502388618906

HÖrl, S. (2017). Agent-based simulation of autonomous taxi services with dynamic demand responses. *Procedia Computer Science*, *109*, 899–904. https://doi.org/10.1016/j.procs.2017.05.418

Horni, A., Nagel, K., & W. Axhausen, K. (Eds.). (2016). *The Multi-Agent Transport Simulation MATSim*. Ubiquity Press. https://doi.org/10.5334/baw

Jain, A., Zamir, A. R., Savarese, S., & Saxena, A. (2016, April). Structural-RNN: Deep Learning on Spatio-Temporal Graphs [Comment: CVPR 2016 (Oral)].

Jiang, W., & Luo, J. (2021). Graph Neural Network for Traffic Forecasting: A Survey. *arXiv:2101.11174 [cs]*.

Jiang, W., & Zhang, L. (2019). Geospatial data to images: A deep-learning framework for traffic forecasting. *Tsinghua Science and Technology*, *24*(1), 52–64. https://doi.org/10.26599/TST.2018.9010033

Jin, G., Sha, H., Feng, Y., Cheng, Q., & Huang, J. (2021). GSEN: An ensemble deep learning benchmark model for urban hotspots spatiotemporal prediction. *Neurocomputing*, *455*, 353–367. https://doi.org/10.1016/j.neucom.2021.05.008

Jin, G., Wang, M., Zhang, J., Sha, H., & Huang, J. (2022). STGNN-TTE: Travel time estimation via spatial–temporal graph neural network. *Future Generation Computer Systems*, *126*, 70–81. https://doi.org/10.1016/j.future.2021.07.012

Jin, G., Yan, H., Li, F., Huang, J., & Li, Y. (2021). Spatio-Temporal Dual Graph Neural Networks for Travel Time Estimation. *arXiv:2105.13591 [cs]*.

Jing, P., Hu, H., Zhan, F., Chen, Y., & Shi, Y. (2020). Agent-Based Simulation of Autonomous Vehicles: A Systematic Literature Review. *IEEE Access*, *8*, 79089–79103. https://doi.org/10.1109/ACCESS.2020.2990295

Khoshraftar, S., & An, A. (2022, June). A Survey on Graph Representation Learning Methods.

Kipf, T. N., & Welling, M. (2017, February). Semi-Supervised Classification with Graph Convolutional Networks [Comment: Published as a conference paper at ICLR 2017].

Lecun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time-series. In M. Arbib (Ed.), *The handbook of brain theory and neural networks*. MIT Press.

Lee, K., Jin, Q., Animesh, A., & Ramaprasad, J. (2018). Are Ride-Hailing Platforms Sustainable? Impact of Uber on Public Transportation and Traffic Congestion. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3244207

Li, Y., Yu, R., Shahabi, C., & Liu, Y. (2018, February). Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting [Comment: Published as a conference paper at ICLR 2018].

Liu, Z., & Zhou, J. (2020). Introduction to Graph Neural Networks. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *14*(2), 1–127. https://doi.org/10.2200/S00980ED1V01Y202001AIM045

Maciejewski, M., & Nagel, K. (2012). Towards Multi-Agent Simulation of the Dynamic Vehicle Routing Problem in MATSim. In D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, R. Wyrzykowski, J. Dongarra, K. Karczewski, & J. Waśniewski (Eds.), *Parallel Process-*

ing and Applied Mathematics (pp. 551–560, Vol. 7204). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-31500-8_57

Maind, M. S. B., & Wankar, M. P. (2014). Research Paper on Basic of Artificial Neural Network. *International Journal on Recent and Innovation Trends in Computing and Communication*, *2*(1), 96–100. https://doi.org/10.17762/ijritcc.v2i1.2920

Merkwirth, C., & Lengauer, T. (2005). Automatic Generation of Complementary Descriptors with Molecular Graph Networks. *Journal of Chemical Information and Modeling*, *45*(5), 1159–1168. https://doi.org/10.1021/ci049613b

Perozzi, B., Al-Rfou, R., & Skiena, S. (2014). DeepWalk: Online Learning of Social Representations [Comment: 10 pages, 5 figures, 4 tables]. *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 701–710. https://doi.org/10.1145/2623330.2623732

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton : (Project Para)* (tech. rep.). Buffalo, NY.

Ruch, C., Hörl, S., & Frazzoli, E. (2018). AMoDeus, a Simulation-Based Testbed for Autonomous Mobility-on-Demand Systems. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 3639–3644. https://doi.org/10.1109/ITSC.2018.8569961

Ruch, C., Lu, C., Sieber, L., & Frazzoli, E. (2021). Quantifying the Efficiency of Ride Sharing. *IEEE Transactions on Intelligent Transportation Systems*, *22*(9), 5811–5816. https://doi.org/10.1109/TITS.2020.2990202

Saracoglu, Ö. G., & Altural, H. (2010). Color Regeneration from Reflective Color Sensor Using an Artificial Intelligent Technique. *Sensors*, *10*(9), 8363–8374. https://doi.org/10.3390/s100908363

Scarselli, F., Gori, M., Ah Chung Tsoi, Hagenbuchner, M., & Monfardini, G. (2009). Computational Capabilities of Graph Neural Networks. *IEEE Transactions on Neural Networks*, *20*(1), 81–102. https://doi.org/10.1109/TNN.2008.2005141

Seo, Y., Defferrard, M., Vandergheynst, P., & Bresson, X. (2016, December). Structured Sequence Modeling with Graph Convolutional Recurrent Networks.

Stutz, D. (2014). *Understanding Convolutional Neural Networks* (tech. rep.).

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., & Mei, Q. (2015). LINE: Large-scale Information Network Embedding [Comment: WWW 2015]. *Proceedings of the 24th International Conference on World Wide Web*, 1067–1077. https://doi.org/10.1145/2736277.2741093

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, December). Attention Is All You Need [Comment: 15 pages, 5 figures].

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018, February). Graph Attention Networks [Comment: To appear at ICLR 2018. 12 pages, 2 figures].

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P.-A. (2010). Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion, 38.

Wang, Q., Xu, C., Zhang, W., & Li, J. (2021). GraphTTE: Travel Time Estimation Based on Attention-Spatiotemporal Graphs. *IEEE Signal Processing Letters*, *28*, 239–243. https://doi.org/10.1109/LSP.2020.3048849

Wang, S., Li, Y., Zhang, J., Meng, Q., Meng, L., & Gao, F. (2020). PM2.5-GNN: A Domain Knowledge Enhanced Graph Neural Network For PM2.5 Forecasting. *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*, 163–166. https://doi.org/10.1145/3397536.3422208

Wang, X., Ma, Y., Wang, Y., Jin, W., Wang, X., Tang, J., Jia, C., & Yu, J. (2020). Traffic Flow Prediction via Spatial Temporal Graph Neural Network. *Proceedings of The Web Conference 2020*, 1082–1092. https://doi.org/10.1145/3366423.3380186

Wu, Y., Lian, D., Xu, Y., Wu, L., & Chen, E. (2020). Graph Convolutional Networks with Markov Random Field Reasoning for Social Spammer Detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(01), 1054–1061. https://doi.org/10.1609/aaai.v34i01.5455

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A Comprehensive Survey on Graph Neural Networks [Comment: Minor revision (updated tables and references)]. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(1), 4–24. https://doi.org/10.1109/TNNLS.2020.2978386

Xu, Z., Yin, Y., & Ye, J. (2020). On the supply curve of ride-hailing systems. *Transportation Research Part B: Methodological*, *132*, 29–43. https://doi.org/10.1016/j.trb.2019.02.011

Yan, C., Zhu, H., Korolko, N., & Woodard, D. (2020). Dynamic pricing and matching in ride-hailing platforms. *Naval Research Logistics (NRL)*, *67*(8), 705–724. https://doi.org/10.1002/nav.21872

Yan, S., Xiong, Y., & Lin, D. (2018, January). Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition [Comment: Accepted by AAAI 2018].

Yu, B., Yin, H., & Zhu, Z. (2018). Spatio-Temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting [Comment: Proceedings of the 27th International Joint Conference on Artificial Intelligence]. *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, 3634–3640. https://doi.org/10.24963/ijcai.2018/505

Zha, L., Yin, Y., & Yang, H. (2016). Economic analysis of ride-sourcing markets. *Transportation Research Part C: Emerging Technologies*, *71*, 249–266. https://doi.org/10.1016/j.trc.2016.07.010

Zhang, K., Zhao, X., Li, X., You, X., & Zhu, Y. (2021). Network Traffic Prediction via Deep Graph-Sequence Spatiotemporal Modeling Based on Mobile Virtual Reality Technology (B. Nagaraj, Ed.). *Wireless Communications and Mobile Computing*, *2021*, 1–12. https://doi.org/10.1155/2021/2353875

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI Open*, *1*, 57–81. https://doi.org/10.1016/j.aiopen.2021.01.001

Zhuang, C., & Ma, Q. (2018). Dual Graph Convolutional Networks for Graph-Based Semi-Supervised Classification. *Proceedings of the 2018 World Wide Web Conference*, 499–508. https://doi.org/10.1145/3178876.3186116