UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Pavlo Pyvovar

Short-term Traffic Forecasting Using Graph Neural Networks on Taxi Data

Master's Thesis (30 ECTS)

Supervisor(s): Amnir Hadachi, PhD
Joonas Puura, MSc

Tartu 2024

Short-term Traffic Forecasting Using Graph Neural Networks on Taxi Data

Abstract:
Traffic forecasting (TF) is the task of predicting a traffic state in a city for a certain time horizon into the future. Accurate traffic forecasts are crucial for preventing congestion, which causes degradation of life quality for people, and improving mobility services that help citizens commute more efficiently. Machine learning methods have been applied extensively to TF problems and Graph Neural Networks (GNN) have recently become state-of-the-art methods. In this study, we aim to apply a GNN to a particular taxi probe dataset which was provided to us by an Estonian mobility company Bolt Technology OÜ. We show that our implementation of GNN is capable of learning seasonality from time series data and yet does not outperform traditional machine learning methods. So further improvements are needed to make a GNN that excels at TF.

Keywords:
Traffic forecasting, Machine learning, Deep learning, Graph neural networks

CERCS: P176 - Artificial intelligence; P170 Computer science, numerical analysis, systems, control.

Tüübituletus neljandat järku loogikavalemitele

Lühikokkuvõte: Liikluse prognoosimise ülesanne seisneb linna liiklusolukorra ennustamises teatud tuleviku aegadeks. Täpsed liiklusennustused on olulised ummikute ennetamiseks, mis halvendavad märkimisväärselt elukvaliteeti linnas, ja efektiivsemate liikuvsteenuste pakkumiseks, mis võimaldavad linnaelanikel paremini liigelda. Masinõppel põhinevaid meetodeid on liikluse prognoosmise ülesandeks varem laialdaselt rakendatud. Hiljuti on hakatud rohkem rakendama graaf-närvivõrkudel põhinevaid lahendusi. Käesolevas töös rakendatakse graaf-närvivõrke liikuvusettevõtte Bolt poolt jagatud andmetele. Töös näidatakse, et meie poolt arendatud graaf-närvivõrkudel põhinev implementatsioon suudab õppida hooajalisust, kuid võrreldes traditsioonilisemate masinõppe meetoditega, ei anna paremaid tulemusi. Seega, et rakendada graaf-närvivõrkudel põhinevaid lahendusi praktikas, on vaja teha täiendavat tööd.

Võtmesõnad:

CERCS: P176 - Tehisintellekt; P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine.

Contents

1 Introduction

One of the most important factors for the success of a modern ride-hailing service is the accuracy of the estimated time of arrival (ETA) for taxi drivers. ETAs are used for proposing the best matching driver to a client. Therefore, more accurate ETAs lead to better driver allocation, which allows them to complete more orders and earn more income, and shorter pickup times for customers. It also means that more orders can be completed with the same amount of drivers, hence greater supply without the increase in the number of partners. As a consequence riders and drivers are more satisfied, and the business is making more profits.

Practical experience proves that ride-hailing services still often predict suboptimal ETAs. Traffic depends on many phenomena such as weather, chaotic system of participants, accidents, public events, etc, all of which make the problem highly complex. Apart from that, ride-hailing companies usually have access to only taxis, which are a small fraction of overall traffic participants (e.g., public transportation, private cars, etc.). Moreover, the scale of the data is huge, which requires efficient processing methods. Last but not least, predicting accurate ETA requires understanding how the traffic state changes over time (temporal dependency) and between different locations of a city (spatial dependency), which is a highly complex non-linear problem. Therefore, there is room for improvement, and this topic is still a relevant area of research.

This Master's thesis was conducted in collaboration with the Estonian mobility company Bolt. The company provided anonymised taxi data, infrastructure, and mentorship intending to explore novel methods for traffic forecasting (TF), comparing them with the company's production systems and potentially improving existing solutions.

In this work, we aim to address the problem of short-term TF with the state-of-the-art approach, Graph Neural Network (GNN), due to its ability to capture both temporal and spatial aspects of traffic data. First, we prepare the taxi dataset provided by Bolt as a road graph where roads are nodes and edges are connections (intersections) between them. Second, we create a simple GNN architecture based on a Graph Attention Network [VCC+18]. We compare the GNN model with other approaches, such as historical averages (HA), linear regression, and gradient-boosted decision trees (GBDT).

The remainder of this thesis is organised as follows. Chapter 2 discusses the related work that has been done in the field of TF. It includes classical probabilistic and statistical approaches, traditional machine learning (ML), and deep learning (DL) methods with a particular focus on GNNs. Chapter 3 details the methodology of the work, including data description, data preprocessing, the implementation description of the HA approach, linear regression, GBDT and a GNN. Chapter 4 presents the results of the experiments, evaluation and comparison of models. In chapter 5 we suggest how our work can be improved in subsequent research. Lastly, chapter 6 summarises the outcomes of the project.

2 Related work

In this chapter, we first define a TF task, then give an overview of the classical models that solve it, and finally introduce the state-of-the-art methods with a particular focus on GNNs.

2.1 Traffic forecasting

TF is a spatio-temporal time series prediction problem, where the input contains a traffic variable (e.g., flow or speed) represented in one or more time series, and the output is a forecast of the future conditions [SML+22].

In this work, we focus on a particular type of TF problem, traffic speed prediction. Traffic speed is the average speed of vehicles passing through a spatial unit (e.g., road intersection) in a given time. The speed measurement on a city road can reflect the road traffic density and an accurate traffic speed prediction can be used for estimation-of-arrival applications, which in turn are vital for ride-hailing businesses [JL22]. In particular, if we calculate average speeds for every road using historical speed data, we can combine that information with road lengths extracted from a map to obtain average traversal times. Average traversal times then can be used as weights for graph edges for routing with tools like Open Source Routing Machine (OSRM) [LV11], which finds the shortest paths in road networks.

TF can be categorised by the length of the prediction horizon into short-term and long-term forecasting. In short-term TF, the prediction horizon usually ranges from seconds to multiple hours into the future and uses both historical and current traffic information [SML+22]. However, there is no agreed-upon definition for long-term forecasting; researchers distinguish it from short-term forecasting differently. Nonetheless, for the clarity of this study, we will define long-term forecasting as predicting a certain variable (speed in our case) for as far as one week ahead.

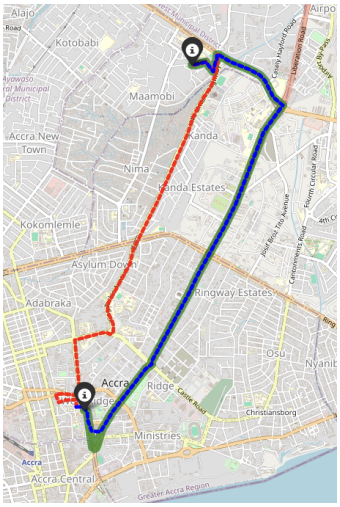

Figure 1. OSRM [LV11] routing visualised. Given average road traversal time estimates OSRM estimates potential routes which can be used for a final ETA

2.2 Statistical models

Statistical models, due to their transparency and low computational requirements, are often used for TF [YZT+20]. Non-learning approaches such as k-nearest-neighbors (KNN) [ZLY+13], HA, and the conventional vector autoregressive (VAR) models are good baselines to start with. Apart from that, the autoregressive integrated moving average (ARIMA) [WH03] and its variants have been applied to TF many times

[WH03, SW07]. Despite being useful on small data sets with short time series sequences, statistical models cannot capture spatio-temporal dependency and can only consider the temporal information [SML+22].

2.3 Traditional machine learning approaches

ML approaches can model more complex non-linear dependencies and generalise better than classical statistical models [YZT+20]. One of the methods that is commonly used in practice is gradient-boosted decision trees (GBDT). Gradient boosting is an ensembling approach, which means that it is a composition of many relatively weak models that, when taken together, achieve a good performance. Gradient boosting is also an iterative approach and with each iteration, the ensemble is made "stronger" by incorporating a new "weak" learner, which usually is a decision tree. On each iteration, a strong model predicts the target to be learned, and then the error of the prediction is used as a label for the weak learner. Next, the prediction of the weak learner is subtracted from the predictions of the strong model, and the weak model is added to the strong one. The process continues until a certain criterion is met, for example, a maximum number of iterations. Gradient boosting has proven to be effective for tabular data, and a team from Bolt [LI22] showed its efficiency on a TF task. The authors reduced the dimensionality of the input data representing traffic counters with the help of the classic Principle Component Analysis method and fed it as input to a GBDT. As a result, Bolt's team achieved second place in the Traffic4cast 2022 competition [NEM+23].

2.4 Deep learning methods

Over the last decade, DL methods have become more popular than traditional ML approaches due to their generalisation capabilities and ability to extract highly complex non-linear relationships.

One of the most basic DL methods is a multi-layer perceptron (MLP) which is a neural network that consists of more than three fully connected layers. The information flows from input to output layers in one direction, and a backpropagation algorithm is used to learn how to match input data to labels. However, MLP has limitations in reflecting local relationships in traffic data and is prone to noise. Another foundational method used to impute or compress data is Autoencoder (AE) methods [GAW17, WBY+18]. AEs can learn an efficient representation of data without using labels. Also, Restricted Boltzmann Machines (RBMs) are stochastic neural networks that learn in an unsupervised fashion, and Ma et al. [MYWW15] applied a stacked RBM to extract temporal and spatial correlations from traffic flow data. The next big leap forward in the field was made with convolutional neural networks (CNNs) and recurrent neural networks (RNNs).

CNNs proved to be one of the most effective DL approaches for learning spatial and temporal dependency. They usually consist of four types of layers: convolutional, non-linear activation (Rectified Linear Unit (ReLU) is most commonly used), pooling, and fully connected. More layers in a CNN model lead to higher capacity, albeit with a potential drawback of overfitting and long training time. Multiple works have been done to prove the utility of CNNs for extracting spatial relationships from traffic time series data [YTRJ18] motivated by the idea that road traffic time series data in the same neighbourhood are likely to be related. The CNN model was designed to perform convolution on Euclidean-structured input data. Modelling the road network in Euclidean space requires a grid-like representation, however, this limits the effectiveness of the models due to distortion of meaningful connections between nodes. For example, two adjacent roads may be located very closely in Euclidean space but be unconnected and possess vastly different characteristics, which limits the effectiveness of traditional image-based convolutional approaches for traffic models. As a result, research has been conducted to develop a graph convolution operation for TF [SML+22], which we describe in the following sections.

Recurrent neural networks (RNN) were designed to learn from sequences such as text or time series. RNN models that incorporate a memory cell perform well on sequences and time series. RNN uses a memory component to consider dependency between adjacent time steps. The first proposed RNN had two fully connected layers with a feedback loop in the hidden layer as the main contribution. However, this simple design is prone to gradient vanishing or exploding when training with backpropagation large networks, which is a significant hindrance when working with lengthy multivariate traffic time series. Therefore, other methods were developed to address this problem. The most well-known alternatives are Long short-term memory (LSTM) [HS97] and Gated Recurrent Unit (GRU) [CGCB14] models.

The ability of LSTM to capture long-range dependencies selectively makes it good for capturing both long- and short-term dependencies in time series [SML+22]. Nonetheless, LSTM still suffers from the vanishing gradient problem. Therefore, to overcome this limitation, GRU [CGCB14] was proposed. Compared to LSTM, GRU does not have a separate memory block and hence has a smaller number of learnable parameters, which makes training more efficient. Even though GRU has less complexity than LSTM, it has been proven to achieve similar performance on TF tasks [SML+22].

In recent studies, LSTM and GRU are often used in combination with other methods [DHX+20]. One particularly notable solution combined GRU with a Graph Convolutional Network (GCN) and was called the T-GCN [ZSZ+20]. Its architecture consists of cells that combine graph convolution to extract spatial features, followed by a GRU-like series of gates (e.g., reset and update gates) for capturing temporal information. Experimental results showed that T-GCN outperforms standalone GCN and GRU, pinpointing the importance of considering spatio-temporal trends in traffic data [SML+22].

2.5 Graph neural networks

GNNs are well suited for TF problems due to their ability to extract spatial information, which is represented by non-Euclidean graph structures. For instance, a road network can be represented as a graph, with road intersections as the nodes and road connections as the edges or vice versa. Several GNN-based models have demonstrated superior performance to previous approaches to TF. Maxime Labonne has written a comprehensive book [Lab23] about GNNs which we rely upon in our description of the theory behind the GNNs. In this section, we first describe how a basic GNN is built, then dive into Graph Convolutional Network (GCN) [KW17] and finally introduce an alternative attention-based approach.

2.5.1 Basic graph neural networks

In a basic neural network a linear transformation is defined in the following way:

$$h_a = x_a W^T, \tag{1}$$

where x_a is the feature vector of the node a, and W is the weight matrix. However, in such a setting node features are separate from each other and hence the information does not flow between them. We can modify the definition to take the graph topology into account in the following way:

$$h_a = \sum_{i \in N_a} x_i W^T, \tag{2}$$

where N_a is the neighbourhood of the node a.

The matrix version of the equation 1 is the following:

$$H = XW^T, \tag{3}$$

where X is the node feature matrix. To embed the information about node connections we need to add the adjacency matrix A to the equation. To take into consideration the central node we also should add self-loops to the matrix A such that $\tilde{A} = A + I$. Consequently, the graph linear layer can be written as follows:

$$H = \tilde{A}^T X W^T \tag{4}$$

Similar to MLP, graph linear layers can be stacked to form a deep structure with non-linear functions applied.

2.5.2 Graph convolutional network

The GNN layer defined in the previous section has a serious limitation, it does not take into account the fact that in an arbitrary graph, a node can have any number of

neighbours. Therefore, to fix this issue node embedding should be normalised. The following modifies equation 2 by adding a division with a node degree:

$$h_i = \frac{1}{deg(i)} \sum_{j \in N_i} x_j W^T, \tag{5}$$

where $deg(i)$ is the degree of node i. The next step is again converting it into a matrix form. The simplest solution would be to simply multiply equation 4 with a degree matrix with self-loops $\tilde{D} = D + I$, so that:

$$H = \tilde{D}\tilde{A}^T X W^T, \tag{6}$$

however, as Kipf and Welling pointed out in their seminal work [KW17] there is a more efficient solution. The authors noticed that features from nodes with plenty of neighbours disseminate easily, unlike features from more isolated nodes. Therefore, they proposed to use the following formula to counterbalance this effect:

$$H = \tilde{D}^{-\frac{1}{2}} \tilde{A}^T \tilde{D}^{-\frac{1}{2}} X W^T \tag{7}$$

Equation 7 defines a graph convolution layer which is a building block of a GCN.

GCN [KW17] is an evolution of the CNN model designed to perform convolution on graph-structured data. The main contribution of the approach was an invention of the approximation of the spectral graph convolution originally proposed by Bruna et al. [BZSL14], which reduced the number of parameters and improved learning speed. However, the original GCN operates on undirected graphs, which cannot represent the complex traffic diffusion in transportation networks (e.g., upstream and downstream traffic dependencies, as well as directed roadways). Moreover, the approximations leveraged in GCN to reduce computational complexity implicitly assume equal locality, which means that there is an equal relationship between all edges connected to a node. In practice, this assumption does not hold because roadways are designed with different classifications, capacities, and goals despite being interconnected, limiting the performance of the original GCN for TF. In addition to that, GCN fails to consider the temporal features of structured time series data [SML$^+$22]. Many attempts followed to improve GCN architecture further.

The next leap forward was the spatio-temporal graph convolutional network (STGCN) [YYZ18] with the spatio-temporal convolutional block (ST-Conv block), which consists of two gated and fully convolutional temporal layers [GAG$^+$17] that capture temporal relationships and a GCN layer in between that captures spatial features. STGCN is parallelized over the input and is faster to train than convolutional RNNs. Besides that, ST-Conv blocks can be stacked sequentially to increase the depth of the model for more complex transportation networks. Figure 2 shows the architecture of STGCN.

Also, an alternative approach, the diffusion convolutional recurrent network (DCRNN) [LYSL18], was developed to address the limitations of spectral graph convolution for

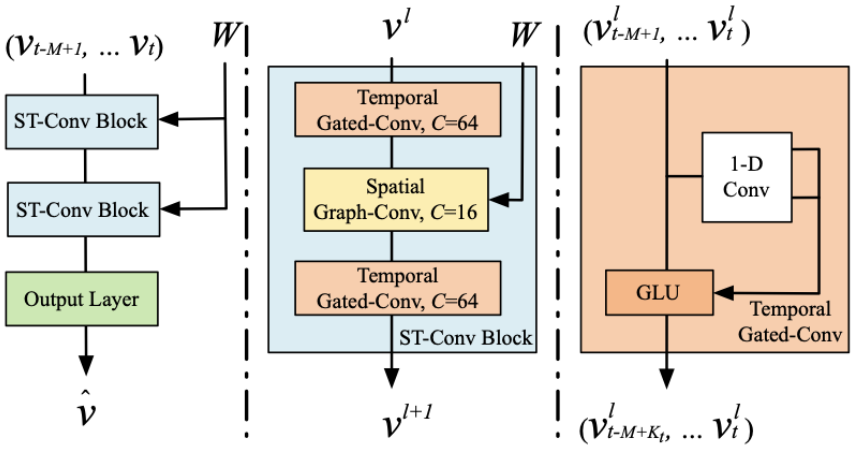

Figure 2. Architecture of spatio-temporal graph convolutional networks. The framework STGCN consists of two spatio-temporal convolutional blocks (ST-Conv blocks) and a fully connected output layer in the end. Each ST-Conv block contains two temporal gated convolution layers and one spatial graph convolution layer in the middle. The residual connection and bottleneck strategy are applied inside each block. The input $v_{t-M+1}, ..., v_t$ is uniformly processed by ST-Conv blocks to explore spatial and temporal dependencies coherently. Comprehensive features are integrated by an output layer to generate the final prediction $\hat{v}$ [YYZ18].

spatio-temporal modelling. The key contribution of this work is the proposed diffusion convolution operation, which relates the traffic flow stream to a diffusion process and can effectively capture stochastic and noisy data trends [SML+22]. Figure 3 shows the architecture of DCRNN.

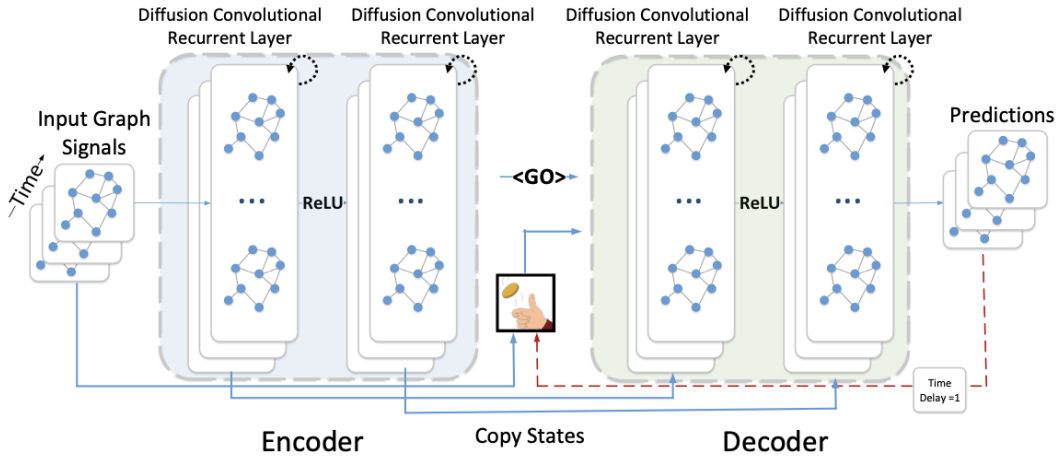

Figure 3. System architecture for the Diffusion Convolutional Recurrent Neural Network designed for spatio-temporal traffic forecasting. The historical time series are fed into an encoder whose final states are used to initialize the decoder. The decoder makes predictions based on either previous ground truth or the model output [LYSL18].

Spatial-temporal synchronous Graph Convolutional Networks (STSGCN) [SLGW20] was proposed to further improve graph-based approaches for TF by directly considering an often overlooked spatio-temporal relationship between nodes: the impact of neighbouring nodes on the future state of their neighbours. The authors define three high-level relationships the models should consider: (1) the influence of a node on its neighbours at the current time-step (spatial dependency); (2) the impact of a node on itself in the future time-step (temporal correlation); and (3) the impact of neighbouring nodes on the future state of a given node (spatio-temporal correlation). Unlike previous works, STSGCN considers all three relationships for each node directly [SML+22]. Figure 4 shows the architecture of STSGCN.

2.5.3 Attention-based methods

GCN-based architectures belong to the spectral approaches to generalising convolution operation to the graph domain. On the other hand, there is a rich avenue of research on non-spectral approaches, which define convolutions directly on the graph, operating on

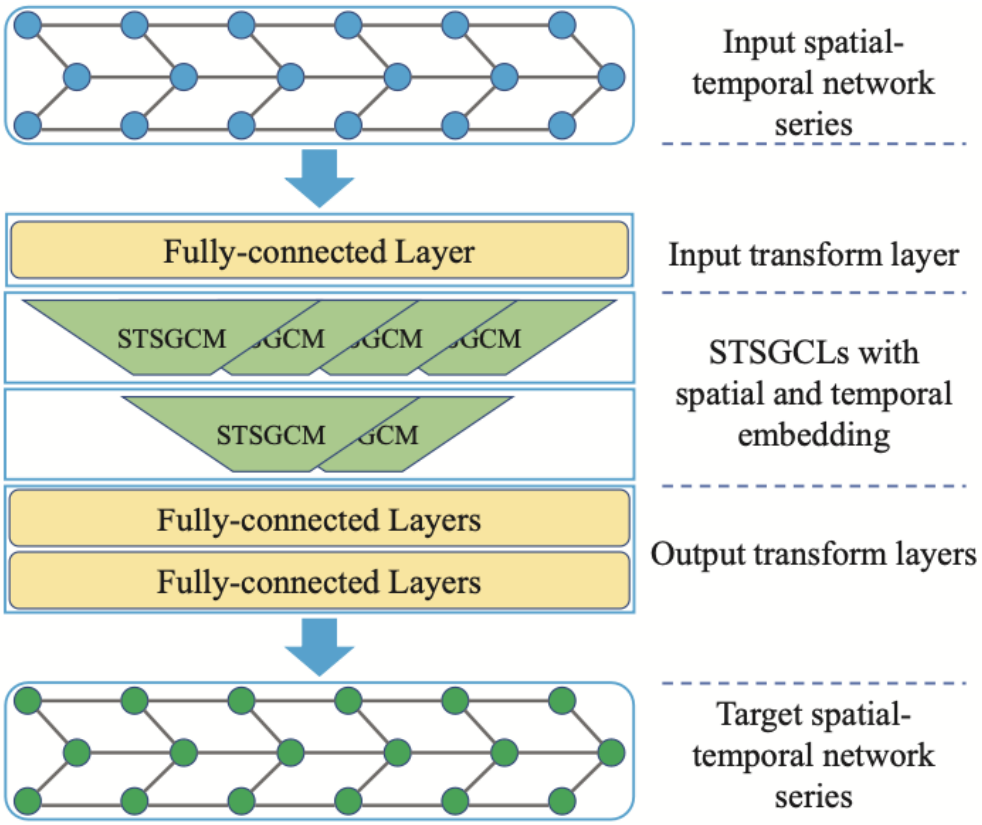

Figure 4. STSGCN architecture. STSGCN consists of multiple Spatial-Temporal Synchronous Graph Convolutional Layers (STSGCLs) with an input and an output layer. It uses an input layer to transform the input features into a higher dimensional space. Then, stacked multiple STSGCLs capture the localized spatial-temporal correlations and heterogeneities in spatial-temporal network series. Finally, it uses a multi-module output layer to map the final representations into the output space. [SLGW20].

groups of spatially close neighbours. As we saw in equation 7 in a graph convolutional layer normalisation coefficients are static. A better approach is to calculate those weight factors using a process called self-attention.

Attention mechanisms have become almost a standard in many sequence-based tasks [BCB16, GAGD17]. One of the benefits of attention mechanisms is that they allow for dealing with variable-sized inputs, focusing on the most relevant parts of the input to make decisions. When an attention mechanism is used to compute a representation of a single sequence, it is commonly referred to as self-attention. Vaswani et al. in 2017 [VSP+23] showed that self-attention is sufficient for constructing a powerful model obtaining state-of-the-art performance on the machine translation task, and the architecture was coined the Transformer.

Inspired by the success of the Transformer, Veličković et al. proposed a GAT architecture [VCC+18]. The idea is to compute the hidden representations of each node in the graph by attending to its neighbours and following a self-attention strategy. The attention architecture has several interesting properties: (1) the operation is efficient since it is parallelisable across node-neighbour pairs; (2) it can be applied to graph nodes having different degrees by specifying arbitrary weights to the neighbours; and (3) the model is directly applicable to inductive learning problems, including tasks where the model has to generalize to completely unseen graphs.

The basic idea behind graph attention is that some nodes are more important than others. The goal of the graph attention layer is to compute weight coefficients that consider the importance of node features. To define a graph attention operator let us call node weighting factors attention scores and note them α_{ij}, the attention score between nodes i and j. The graph attention operator then can be formulated as follows:

$$h_i = \sum_{j \in N_i} \alpha_{ij} W x_j \tag{8}$$

Attention scores represent the importance between a central node i and a neighbour j. There are three steps to compute attention scores α_{ij}. The first is a linear transformation:

$$a_{ij} = W_{att}^T [W x_i \| W x_j], \tag{9}$$

where a_{ij} are attention coefficients, W is a weight matrix from hidden dimension, W_{att} is weight matrix for the final linear transformation. In equation 9 $W x_i$ and $W x_j$ represent hidden vectors of nodes i and j respectively, which are then concatenated (denoted with $\|$ operator), and then another linear transformation is applied with learnable weight matrix W_{att}.

In the next step, a Leaky ReLU activation function is applied to the output of equation 9. The operation is defined in the following equation:

$$e_{ij} = LeakyReLU(a_{ij}) \tag{10}$$

Finally, softmax normalisation is applied to previous outputs e_{ij} to bring those values to the same scale and obtain attention scores:

$$\alpha_{ij} = softmax_j(e_{ij}) = \frac{exp(e_{ij})}{\sum_{k \in N_i} exp(e_{ik})} \tag{11}$$

However, as was pointed out by Vaswani et al. the self-attention mechanism is volatile [VSP+23]. Therefore, in the GAT architecture, as in the Transformer, a multi-head attention mechanism is used, such that multiple embeddings are computed with their own attention scores. The results then can be combined in two ways:

- averaging, first embeddings for each head are summed up and then normalised by the number of attention heads:

$$h_i = \frac{1}{n} \sum_{k=1}^{n} h_i^k = \frac{1}{n} \sum_{k=1}^{n} \sum_{j \in N_i} \alpha_{ij}^k W^k x_j \tag{12}$$

- concatenating, so that a larger matrix is produced:

$$h_i = \|_{k=1}^{n} h_i^k = \|_{k=1}^{n} \sum_{j \in N_i} \alpha_{ij}^k W^k x_j \tag{13}$$

In 2021, Brody et al. proposed GATv2 [BAY22] as an improvement to the original architecture, simply modifying the order of operations such that the weight matrix W is applied after the concatenation and the attention matrix W_{att} after the $LeakyReLU$ function. Hence attention scores in GATv2 are computed in the following way:

$$\alpha_{ij} = \frac{exp(W_{att}^T LeakyReLU(W[x_i \| x_j])}{\sum_{k \in N_i} exp(W_{att}^T LeakyReLU(W[x_i \| x_j]))} \tag{14}$$

There were also some interesting hybrid methods that utilised the attention mechanism with other models. An attention-based convolutional model called STAGCN [GD22] employs both graph-based and standard convolution with attention mechanisms. The authors improve upon prior approaches to dynamically learning the GCN adjacency matrix, combining an attention mechanism with the GCN [SML+22]. Figure 5 shows the STAGCN architecture.

2.6 Regression objective functions

TF is a regression problem, and therefore appropriate regression objective function should be used to optimise an ML model during training. In this section, we define some of the available options of objectives for a regression task.

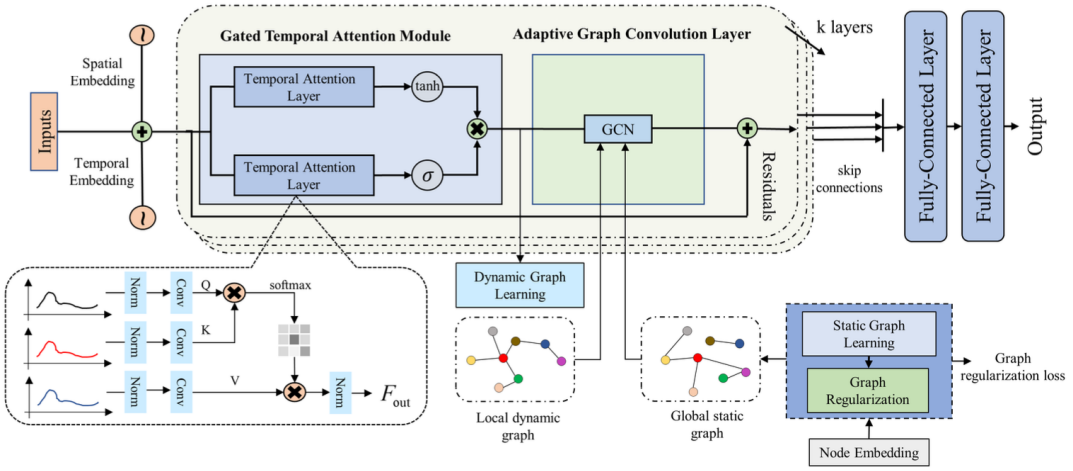

Figure 5. The framework of STAGCN. The model consists of a spatial static–dynamic graph learning layer, a gated temporal attention module (Gated TAM), and an adaptive graph convolution layer (GCN). The input and learned spatiotemporal embedding are first passed through Gated TAM, followed by the graph learning layer to obtain static and dynamic graphs. Then, feature representation and graphs are passed to GCN for spatial modelling [GD22].

Mean absolute error (MAE) is defined as the average absolute difference between predicted values and actual values. The MAE formula is given below:

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{15}$$

where y_i is the true value, x_i is the prediction, and n is the sample size.

Another objective function that is often used is mean squared error (MSE) which is the average of the squared differences between the predicted and actual values. It squares the errors and hence emphasizes larger errors more than smaller ones. This way outliers are penalised more severely. The MSE formula is given below:

$$\mathrm{MSE} = \frac{\sum_{i=1}^{n} (y_i - x_i)^2}{n} \tag{16}$$

Root mean squared error (RMSE) is defined as the square root of MSE, the mathematical formula given below:

$$\mathrm{MSE} = \sqrt{\frac{\sum_{i=1}^{n} (y_i - x_i)^2}{n}} \tag{17}$$

It is more interpretable than the MSE and also is in the same units as the input variables.

Mean absolute percentage error (MAPE) is defined according to the following formula:

$$\mathrm{MAPE} = 100\frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - x_i}{y_i}\right| \tag{18}$$

First, predicted and actual values are subtracted, and then divided by the actual value. Then the average of absolute values of such ratios is calculated and scaled by 100.

The choice of an objective function depends on the task at hand. However, if there is not a clear understanding of which function to use, it is better to conduct an experiment comparing a training procedure with different objective functions.

3 Methodology

In this chapter, we describe the data that was used for the research, how it was preprocessed, the models that were developed, and training details.

3.1 Data description and preprocessing

The data for this thesis was provided by the Estonian mobility company Bolt. In particular, data from the ride-hailing business vertical was shared. It is important to note that this data is inherently biased because its source is Bolt's platform drivers, and other traffic participants, such as private vehicles and public transport, are not taken into account. The data was collected over 7 weeks between June and August 2023.

Bolt drivers record GPS signals with mobile phones when providing service, during pickup and drop-off. These tracking points are quite noisy and do not immediately map to roads. Therefore, a map-matching algorithm is applied to match these tracking points to the likeliest route on the map. The Open Source Routing Machine (OSRM) [LV11] map-matching algorithm was used with OpenStreetMap (OSM) [Ope17]. OSRM algorithm uses a Hidden Markov Model to find the most likely road route represented by a time-stamped sequence of latitude/longitude pairs [NK09], figure 6 illustrates an example.

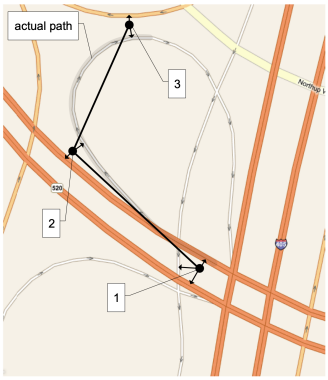

Figure 6. Map matching consists of matching measured locations (black dots) to the road network to infer the vehicle's actual path (light grey curve). Merely matching to the nearest road is prone to mistakes. [NK09]

Each tracking point has an associated timestamp with it. As a next step, we floor the timestamp of every tracking point to its closest 15-minute interval, hence constraining the measurement frequency. Also, we aggregate all tracking points that were measured within a 15-minute time interval on each edge by taking a median of them. As a result,

the core dataset contains the following attributes: OSM IDs of start and end nodes of an edge, respectively, a 15-minute time interval during which the measurement was done, and the actual speed measurement in kilometres per hour. Figure 7 visualises the aggregation preprocessing step.

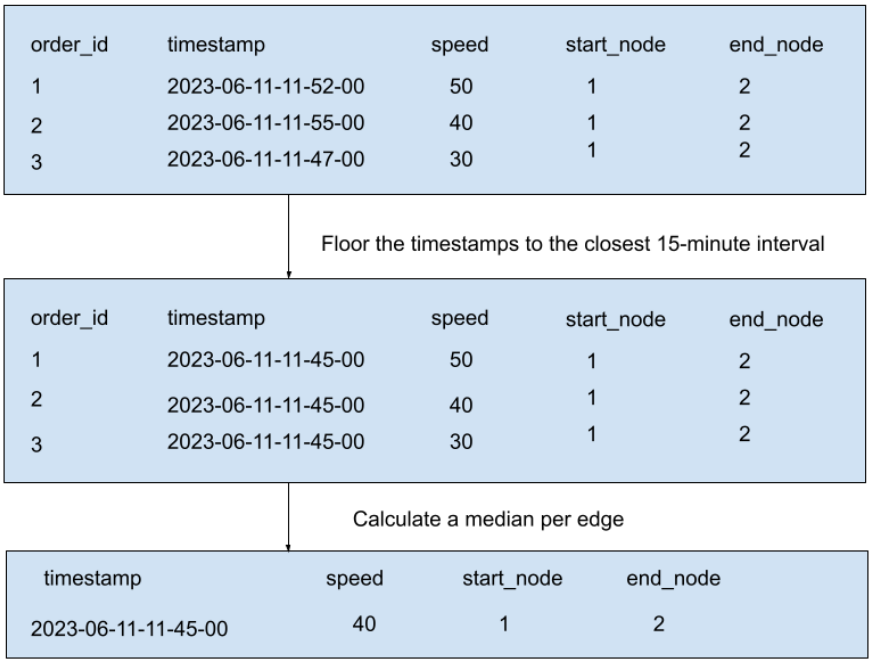

Figure 7. Aggregation of speeds by 15-minute intervals and edges

For this thesis, we focused on one city, Bucharest, Romania. We first extract from the OSM file the whole city car-driving road graph using pyrosm [Ten20], which contains 280557 nodes and 58078 edges. Figure 8 shows the full road network of Bucharest plotted on the map. Nodes are points in space, specifically road intersections, and edges are OSM ways, which are segments into which roads are split. From now on we use the terms edges and road segments interchangeably. Then, to make research iterations faster and limit resource consumption, we extracted a subgraph of the city centre of Bucharest. We have built a subgraph by extracting a k-hop neighbourhood around a popular road intersection which we chose arbitrarily. We experimented with different values of k and eventually settled with $k = 20$. However, we had to remove several edges from the resulting graph because they had insufficient measurements. As a result, we obtained a graph with 317 edges, with several connected components, one big component with the majority of roads in it, and a few smaller ones that corresponded to rarely used roads. However, the developed methods are potentially generalisable to any other city where GPS probe data is available. Figure 11 shows a subgraph of the city centre of Bucharest

that was extracted for the experiments in this thesis. After extracting a subgraph our dataset consisted of 863414 samples in 7 weeks. Figure 12 shows an example time series for two randomly selected road segments from the aforementioned subgraph. The plotted road segments are neighbours, but as we can see from figure 12, the density of samples is drastically different. This highlights the challenge of missing values we faced in this work.

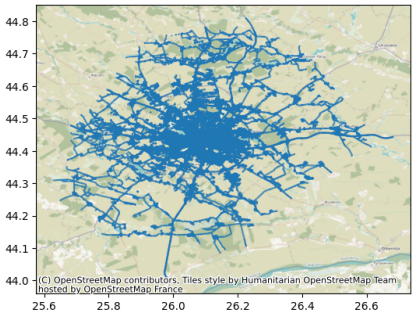

Figure 8. Bucharest road network extracted from OSM

The dataset was split into training, validation and testing sets by time, meaning that training data preceded validation data, and validation preceded testing to make sure that the order was preserved, so that the model can predict the future. We experimented with the training set size, ranging from 2 to 5 weeks, but for both validation and testing, we reserved 1 week. In 5 weeks of training data we had 619172 samples, 122491 samples in 1 week of validation, and 121751 samples in the testing set.

To calculate ETAs, we need to estimate the average traversal time on a road segment, which we can obtain using the length (available from the map) and average speed of the road segment. Since our dataset contains speed measurements, we first solved the speed estimation task. We pose this problem as a graph node regression and flip the definition of nodes and edges so that road segments become nodes, and intersections are connections between these nodes. Node regression is suitable in this case because speed is an attribute of a road segment, which we try to learn, and we also want to consider how adjacent roads impact the road at hand.

Since we aim to tackle a short-term forecasting problem, we also prepared additional dataset features. We added to every sample four speed-related features, namely speed 15, 30, 45 and 60 minutes ago, and the speed at the current timestamp is the prediction target. Later on, depending on the model used we generate additional features.

Since the dataset contains many missing values, we also had to perform data imputation. For the GBDT we used the average speed from the training set for speed-related features.

For the GNN we had to make additional preprocessing steps. We assumed that

the graph we are working with is static, in other words, its structure (node and edge composition) remains the same with time, so its edges have speed measurements for every 15-minute interval within the time bounds of our dataset. Some road segments that are popular and often used by Bolt's drivers have good data coverage and, therefore, have speed measurements for every 15-minute interval, but most do not. To give an exact measure of the sparsity of the dataset we filtered 317 edges of interest that we later used for our experiments, calculated how many 15-minute intervals per each of those edges had a measurement, and found that it was 58%. Besides that, we created a histogram of edge measurement counts shown in figure 9. The total number of 15-minute intervals in our dataset is 4704, and hence that is the maximum number of measurements an edge can have. We can see from the histogram that the largest bin corresponds to measurement counts above 4000 but a large proportion of edges have smaller measurement counts. To hold the assumption of a static graph, we had to impute the missing values. Therefore, we designed a hierarchical data imputation approach according to which if the previous condition did not suffice (a queried value does not exist) we moved to the next one. The first two rules in the list refer to the speed values on a road segment a week and two weeks ago. We assume that the speed on a road segment stays relatively stable in a 15-minute interval week to week. If the values in the past two weeks are not present, then we fall back to a neighbouring road speed average with an assumption that adjacent roads have similar traffic patterns. If neighbouring speeds are also not available we resort to speed on the given road segment 15 minutes ago, which is a less accurate missing value replacement than the previous steps because the traffic on a road can change significantly in half an hour. In the next steps, we consider progressively less granular filling values by taking speed averages of various forms. The full list of imputation steps is given below:

1. speed at the same 15-minute interval a week ago;
2. speed at the same 15-minute interval two weeks ago;
3. average neighbour speed at the same 15-minute interval a week ago;
4. average neighbour speed at the same 15-minute interval two weeks ago;
5. speed on the same road segment 15 minutes ago;
6. neighbour average speed 15 minutes ago;
7. average speed of the past hour;
8. average speed of the past 2 hours;
9. average speed of the past 3 hours;
10. average speed of the past 4 hours;

11. average speed of all preceding speed measurements on the road segment;
12. average over edge, weekday, hour, and 15-minute time interval in the training set;
13. average over the edge in the training set;
14. global average (over all edges and time slots) in the training set;

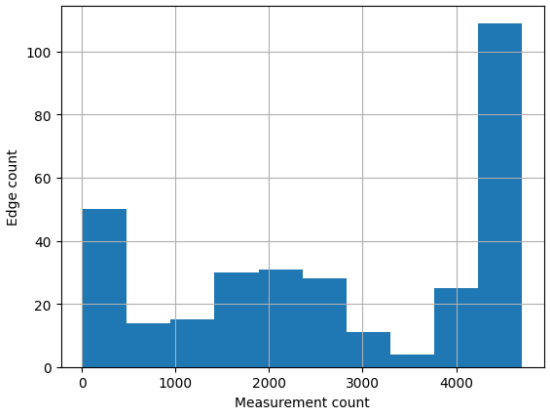

Figure 9. Histogram of edge measurement coverage

A visualisation of data imputation for a randomly selected road segment is shown in figure 10. The top part of the figure shows one of the initial approaches which we considered suboptimal due to a frequent fallback to a constant value (line pattern). The bottom part shows the final solution with more diverse changes in imputed speed.

This way, the GNN was fed with the number of snapshots corresponding to 15-minute intervals, each having a node matrix with 4 features (speed 15, 30, 45 and 60 minutes ago), and a target vector with speed at the current 15-minute interval. It is necessary to note that to maintain such a structure of the dataset we had to impute not only features but also targets. Consequently, to make sure that the model trains only from original targets and not imputed, we also had to provide masks (a binary array that contains 1 if the corresponding sample had an original target and 0 if it was imputed) that would discard samples whose targets were imputed.

Finally, to make optimisation more stable and improve convergence we also normalised floating point type features for RR and GNN using the mean and standard deviation from the training set. For the GNN we also normalised targets and denormalised them when computing final metrics. For a regression problem, the normalisation of targets is optional and usually is not needed, however, we decided to do it because it was simpler to implement.

Categorical features for RR and GBDT were transformed using one-hot encoding.

Figure 10. Data imputation methods. The top image shows a data imputation method that resulted in a fallback to a constant value. The bottom image, corresponding to the final solution used, breaks the pattern of a straight line.

3.2 Models

When developing a novel method it is important to build first reliable baselines to compare against. In this research, our primary focus is a GNN architecture, but to analyse its performance, we also developed simpler and more traditional approaches, namely historical averages (HA), ridge regression (RR), and gradient-boosted decision trees (GBDT).

3.2.1 Historical averages

We developed several alternative HA approaches with different granularity of aggregation. The first and the simplest is the global average, which is the average speed in the training set over all edges and time intervals. The second, more granular approach is the average speed per edge, which is a vector with the same number of elements as there are edges. The third, most granular, is the average speed per edge, weekday, hour and 15-minute interval. In all three cases, the averages are computed over the training set and, during inference time, applied as a look-up table (dictionary). However, this approach has a drawback, when values are averaged as mentioned above a bias is introduced. For example, when taking an average of two weeks for a particular tuple of edge, weekday, hour and 15 minutes, we leak the information from the second week into the first one. Yet, considering this limitation, we still use this HA naive baseline because it is a sufficient starting point.

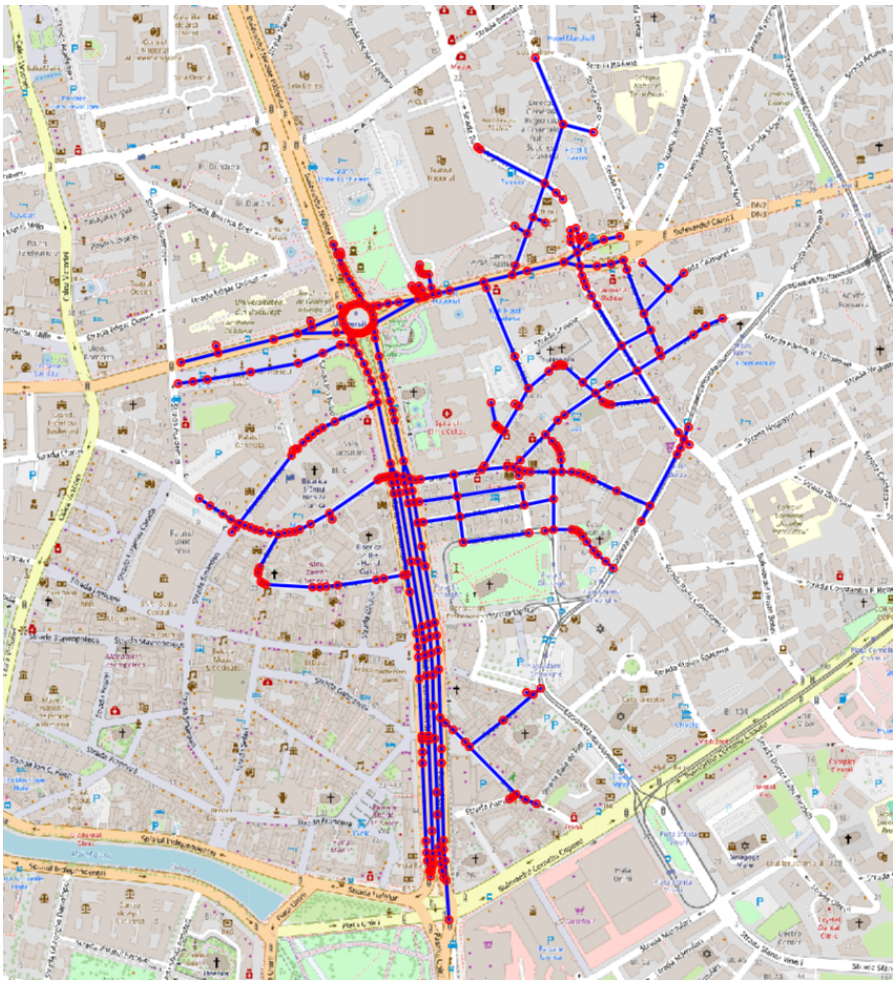

Figure 11. An example subgraph of Bucharest used for experiments

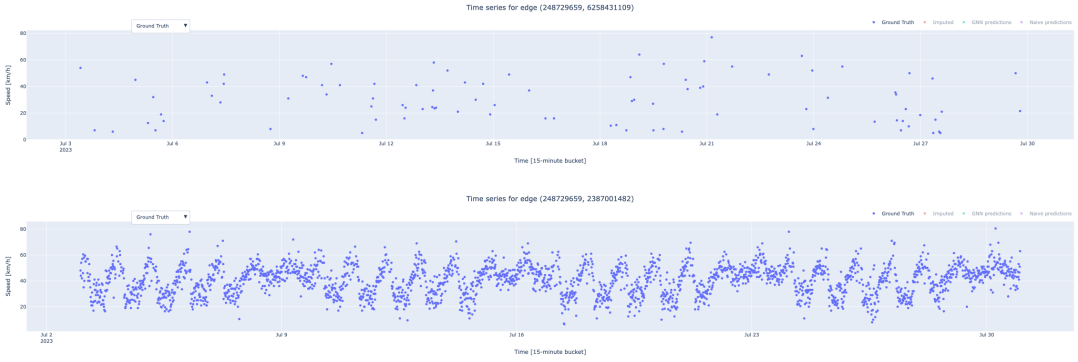

Figure 12. An example time series for two road segments

3.2.2 Ridge regression

RR is a linear regression with L2 regularisation, with the sum of the squares of all the feature weights added to the objective function. For each road segment, we apply a separate RR model. Regularisation is necessary to prevent overfitting, which we detected when we first tried to apply a regular linear regression. We used the following features for the RR model: speeds 15, 30, 45, and 60 minutes ago for the current edge and its neighbouring road segments, weekday, hour and 15-minute intervals.

3.2.3 Gradient boosted decision trees

As was pointed out in the work by Bolt's team [LI22], GBDT can be successfully applied to TF. For GBDT, we extracted additional features from OSM: road length, speed limit, highway type, and binary feature indicating whether it is a one-way road or not.

GBDT has a large number of parameters that can be tuned to enhance model performance. We performed hyperparameter optimisation using the Optuna library [ASY+19] and found the following settings to be appropriate:

- boosting_type - gbdt (default), indicates that decision trees are used as a weak learner;
- objective - regression , indicates that the regression problem is tackled;
- metric - mse, metric used to evaluate performance, MSE;
- num_iterations - 1000, the number of iterations that boosting lasts, defines the depth of the ensemble, default is 100, we selected a larger value due to the size of our dataset;
- num_leaves - 100, controls the complexity of the model, defines the maximum number of leaves that a weak learner can have;
- learning_rate - 0.1, defines how fast the model learns at each iteration;
- min_data_in_leaf - 100, used for dealing with overfitting;
- min_data_in_bin - 100, adjusts the trade-off between complexity and computational efficiency of the model.

3.2.4 Graph neural networks

We implemented several GNN architectures with GAT [VCC+18] (specifically GATv2 [BAY22]) layer as the core component following an example from [FL19]. The comparison of the performance of these models is described in section 4.

All architectures receive as input a matrix with the number of rows corresponding to the number of road segments, and 4 columns corresponding to speed 15, 30, 45, and 60 minutes ago. Following an example of the Transformer architecture [VSP+23], we use a multi-head attention mechanism to stabilise the training.

Figure 13 shows the first architecture we developed, and we called it DummyGNN. It contains three layers in total, fully connected followed by a graph attention layer with 8 attention heads, and then another fully connected layer. Fully connected layers are linear transformations followed by ReLU non-linearity. The attention heads, before being processed by a final fully connected layer, were concatenated. We also developed two variations of this architecture, DummyGNN2 and DummyGNNSlim. To the DummyGNN2 we added a second graph attention layer after the first one, with 1 attention head. In DummyGNNSlim, we instead removed the first fully connected layer so that only 2 layers were left.

In addition, we developed two more architectures different from DummyGNN, which we called GAT and GAT2. The scheme of GAT is shown in figure 14. It contains two graph attention layers, the first one having 8 attention heads, and the second 1. The outputs of 8 attention heads are averaged and then Exponential Linear Unit (ELU) [CUH16] is applied to the output. GAT2 has a linear layer after the second graph attention.

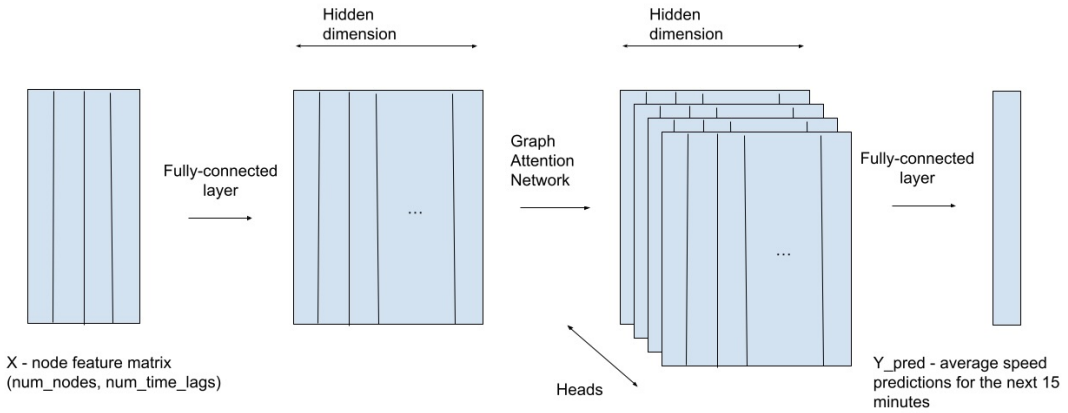

Figure 13. DummyGNN architecture.

3.2.5 Training setup, implementation and training

For all four models, we used MSE as the objective function due to its property to emphasise larger errors more than smaller ones. For GNN training we used AdamW optimiser [LH19].

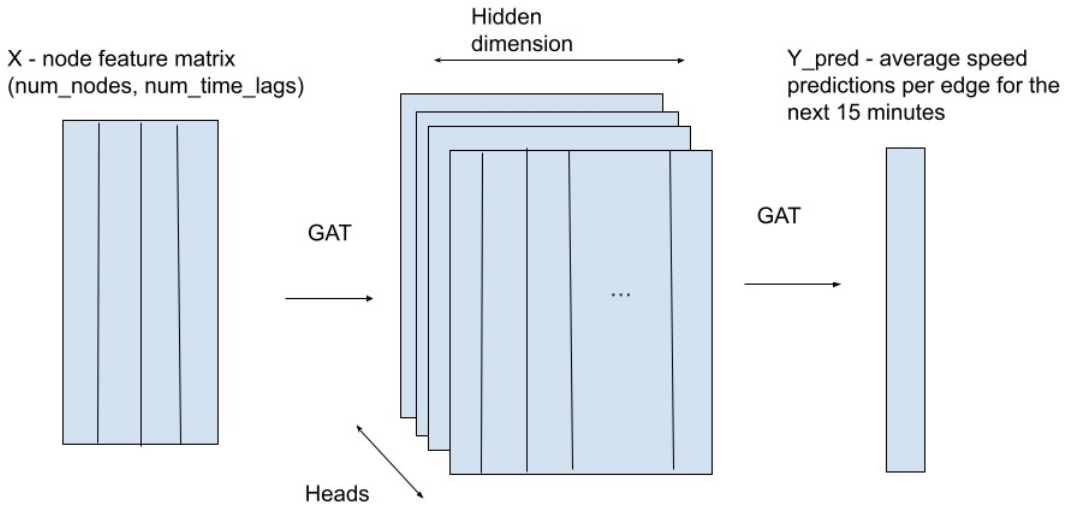

Figure 14. GAT architecture.

All models were implemented using Python 3.10.12. For data preprocessing, we mainly used Numpy and Pandas. For GBDT implementation, we used LightGBM library [KMF+17]. For RR implementation we used Scikit-learn [PVG+11]. For GNN implementation, we used PyTorch 2.2.2 [PGC+17] along with a PyTorch Geometric Temporal 0.54.0 [RSH+21], which is a temporal GNN extension library for PyTorch Geometric 2.2.0 [FL19].

The training and all experiments were conducted in the Databricks notebooks with CPUs, the amount of RAM ranged from 30 up to 120 GB depending on the experiment.

We visualised learning curves, histograms and other kinds of plots with the matplotlib [Hun07] library. Visualisations of speed time series were created using the Plotly [Inc15] library. For visualising the road graph on the city map we used the Folium [pv] library.

To correct grammatical errors and misused vocabulary, we used Grammarly [SLL09].

4 Experimental results

In this chapter, we describe the experiments that we performed. We developed an evaluation procedure that consists of several methods, including computing classical regression metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Percentage Error (MAPE) that are commonly used in the literature, and also our custom histogram of edge errors, errors aggregated by weekdays and hours of the day. MSE, RMSE, and MAE are particularly useful because they are easily interpretable measures of distance between true values and model predictions. In addition, we visualised the speed time series to demonstrate how models learn seasonality.

4.1 Ridge regression

Figure 15 shows ground truth time series, naive model, and RR predictions over 4 weeks in blue, green and red respectively. The naive model, in this case, is an average speed from the training set aggregated by edge, weekday, hour and 15-minute time intervals. Predictions by the naive model were plotted for every time stamp but for RR only for those time intervals where a GPS measurement(s) was available. As we can see from the plots the RR dots follow the seasonality of the ground truth data when there is a sufficient number of measurements. However, RR's maximum and minimum speed prediction values are not as high in magnitude as the ground truth. On the other hand, the HA closely follows the ground truth pattern when there are enough samples but falls back to a constant when there are many missing values.

4.2 Graph neural network

We experimented with different setups of a GNN architecture. To compare these different models, we used a dataset with 2 weeks of training data, 1 week of validation, and 1 of testing data. We trained each model on the dataset using 100 epochs, a learning rate of 0.001 and 32 hidden channels. Table 1 shows the performance of each architecture on the training, validation, and testing sets, as measured by MSE and MAE. As we can see DummyGNN achieved the best performance, and DummyGNN2 is the second best. The main difference between DummyGNN and other architectures is that it has several fully connected layers besides graph attention layers. This adds complexity to the model, and as we know from DL, deeper models tend to be able to model more complex relationships. Apart from that, other architectures have a smaller number of parameters as can be seen from table 2, which also indicates that those models are less complex. So, since our empirical data shows that DummyGNN is the best architecture out of the five we tried, we chose this model as the basis for further experiments.

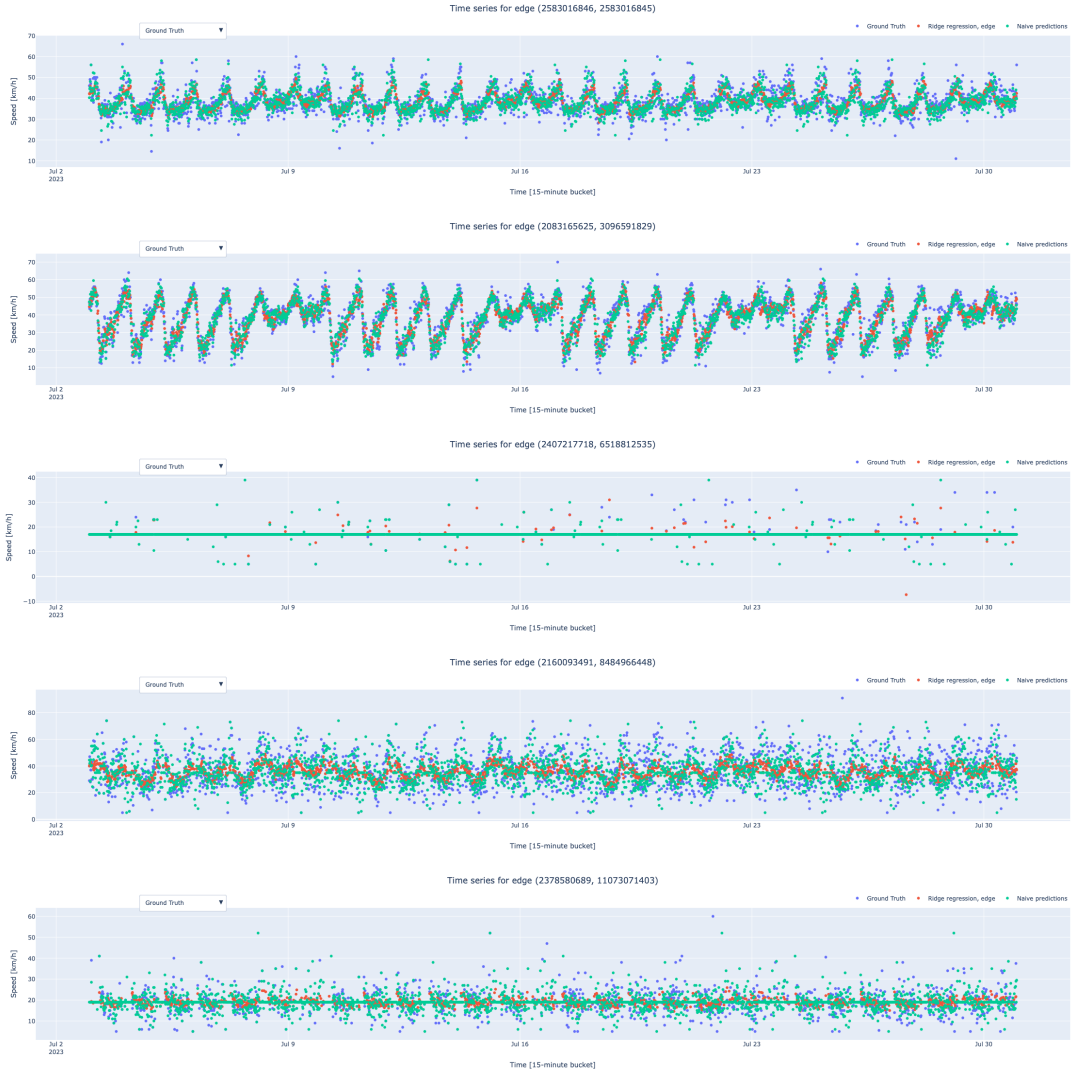

Figure 15. Time series and RR predictions for 5 randomly selected road segments.

Table 1. GNN model evaluation.

Model	Train		Validation		Test	
	MAE	MSE	MAE	MSE	MAE	MSE
DummyGNN	6.04	67.61	5.90	63.78	5.90	64.00
DummyGNN2	6.07	67.92	5.95	64.25	5.93	64.36
DummyGNNSlim	8.99	135.06	8.74	127.8	8.8	129.99
GAT	6.37	72.35	6.23	68.64	6.22	68.78
GAT2	6.42	73.06	6.28	69.34	6.14	66.65

Table 2. Number of parameters in our GNN architectures.

Model	**Number of parameters**
DummyGNN	34017
DummyGNN2	44385
DummyGNNSlim	11521
GAT	3588
GAT2	19617

We then conducted experiments to check how the performance changes with the increase in the training set size. Again, we set training epochs to 100, the learning rate to 0.001 and hidden channels to 32. The same validation and testing set weeks (one for each) were used for all four models, and progressively more past weeks were added to the training set in each consecutive experiment. Table 3 shows the change in DummyGNN performance as we scale the number of training weeks from 2 up to 5. As we can see the error metrics decrease on validation and test splits, which proves our hypothesis that more weeks of training data improve GNN performance.

Table 3. DummyGNN trained on different dataset sizes.

Number of training weeks	Train		Validation		Test	
	MAE	MSE	MAE	MSE	MAE	MSE
2	6.00	66.75	5.97	66.00	6.03	68.08
3	5.95	65.44	5.96	65.95	6.03	68.04
4	5.96	65.72	5.96	65.92	6.03	68.02
5	5.95	65.43	5.96	66.00	6.02	67.98

To evaluate DummyGNN more rigorously we trained it on 5 weeks of data and evaluated it on 2 other weeks. Then we decided to scale it by increasing the number of hidden channels from 32 to 64 and setting the number of epochs to 1000. The learning curves for the experiment are shown in figure 16. Note that the target values were normalised, therefore loss function values are also scaled. We can see from the plot that gradually the model starts to overfit, its performance on the training set improves but on the validation set degrades. The training took 33 minutes with 100 epochs and 8 hours with 1000 on a CPU.

We then compare using our evaluation procedures RR and GBDT with DummyGNN, which was chosen as our final GNN architecture. Table 4 shows the performance measured by MAE, RMSE and MAPE of each model on the train, validation and test sets respectively. As we can see RR and GBDT have the same level of accuracy, however,

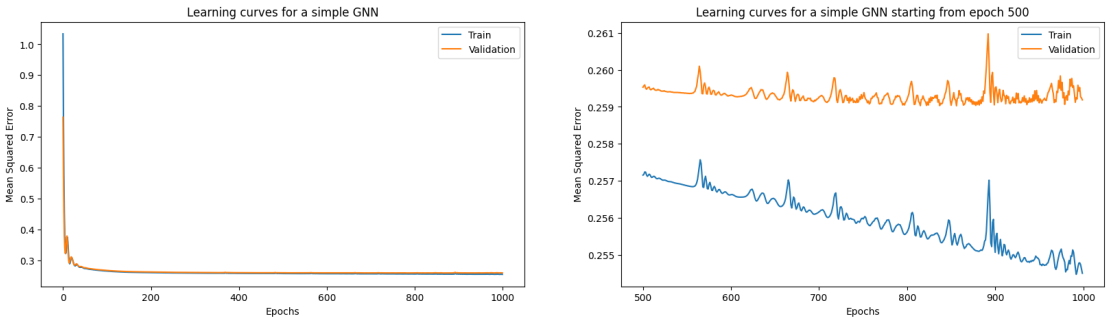

Figure 16. Learning curves for DummyGNN with 64 hidden channels. The left graph shows loss values for all 1000 epochs but the right one plots values starting from epoch 500.

DummyGNN is significantly worse.

Table 4. Model evaluation.

Model	Train			Validation			Test		
	MAE	RMSE	MAPE	MAE	RMSE	MAPE	MAE	RMSE	MAPE
RR	5.07	7.01	0.22	5.28	7.36	0.23	5.27	7.39	0.23
GBDT	4.84	6.62	0.22	5.27	7.33	0.23	5.30	7.40	0.23
GNN	5.56	7.62	0.24	5.63	7.76	0.25	5.67	7.85	0.25

Table 5 shows the performance of the HA models we developed. In all three models, the averages are calculated over the training set consisting of 5 weeks of data. The first one corresponds to a mean over all samples in the training set, the second is the mean aggregated by edges, and the third is the mean aggregated by edges and 15-minute intervals. As we can see our GNN model performs better than all three HA models we developed.

Table 5. Historical averages evaluation.

Model	Train		Validation		Test	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
Mean speed	13.02	15.89	13.01	15.85	13.01	15.87
Edge mean speed	6.90	9.37	6.85	9.32	6.83	9.29
Edge-interval mean speed	5.78	8.05	5.97	8.36	5.99	8.39

Figure 17 compares the performance of RR, GBDT and GNN models using a

histogram of edge errors, which shows how many edges had a particular magnitude of an error. This histogram is useful for understanding if there are many edges with high errors versus the case when errors per edge are relatively small. The more histogram is shifted to the right, the higher errors the model makes, hence, we can see that RR and GBDT on most edges make small errors, while DummyGNN makes higher edge errors.

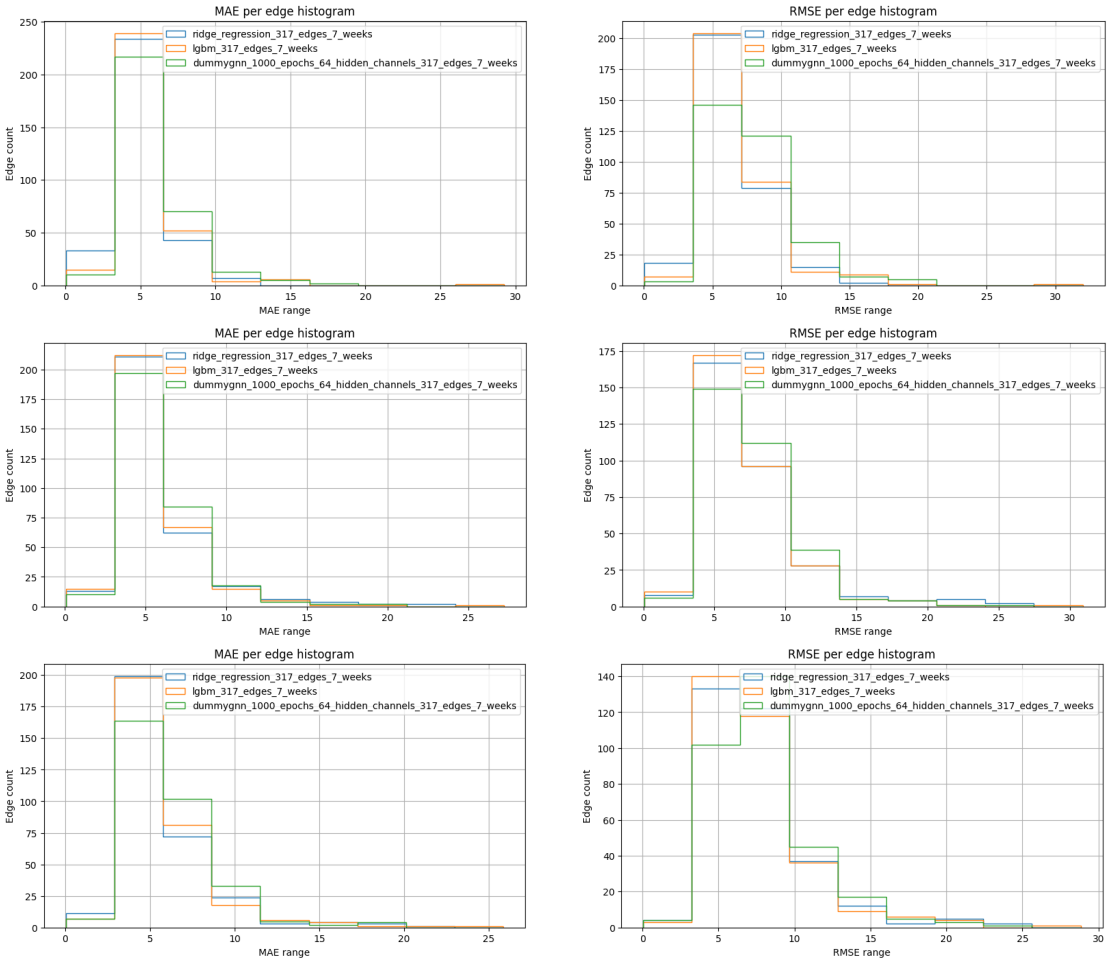

Figure 17. Error histograms of RR, GBDT and GNN models. Plots top to bottom are training, validation and test set histograms respectively. The left column is for MAE, right is for RMSE. Blue marks RR, orange GBDT, and green GNN

To understand if models make higher errors on particular days of the week, we plotted errors per day of the week in figure 18. We can see that all three models tend to make larger errors during weekdays, especially on Mondays. Moreover, errors decrease as weekends approach closer. Then, on Sunday the errors start to ramp up again. These plots also show that our GNN does not outperform GBDT and RR.

In addition to that, we plotted errors per hour of the day in figure 19. We see that for all three models, there are two major peaks in the graph: one corresponds to early morning, which is also the biggest, and the second corresponds to evening. These are rush hours when people commute to work and traffic is particularly hard to predict, hence it makes sense that models make large errors during those periods. In this plot, we also see that the GNN is worse than other models.

Figure 20 shows a road segment and its neighbours' time series, the ground truth and predictions from the three models we developed. For this case data coverage was good, and all the models captured seasonality well. We can see that the GNN predictions marked with purple follow the ground truth more precisely than others, its points reach higher up and down than RR and GBDT, following peak values in the ground truth. However, that could be a sign of overfitting. Figure 21 shows a similar plot for another neighbourhood but with less data coverage. We can see that the models still manage to make valid predictions with that amount of data. Notably, in both plots, we can see that there are many blue points at the bottom and at the top of the time series that are not covered by any model, this suggests that models struggle to predict those extreme values.

Figure 22 visualises a time series for a sequence of consecutive edges at the centre of Bucharest. We can see that the predicted time series for the first two edges and the second two are different. As we can see from the map in figure 23, we specifically selected a sequence of edges around a turn. Hence, there is a change in speed pattern between two pairs of edges.

4.3 Gradient boosted decision trees

To understand what features are most useful for GBDT we measured feature importance after training. The first one shown in figure 26, is a standard method from the LightGBM Python library. By default, this method calculates feature importance as the total gains of splits that use the feature, and that is the option we used. Otherwise, feature importance could be measured by the number of splits it is involved in (the number of times the feature is used).

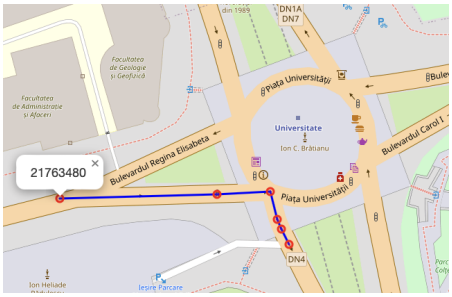

Figure 23. Edge sequence visualised of a map.

Additionally, we calculated feature importance using the SHAP method [LL17], as shown in figure 27. As we can see from both plots the length of the road segment and past-hour speeds are the most impactful features.

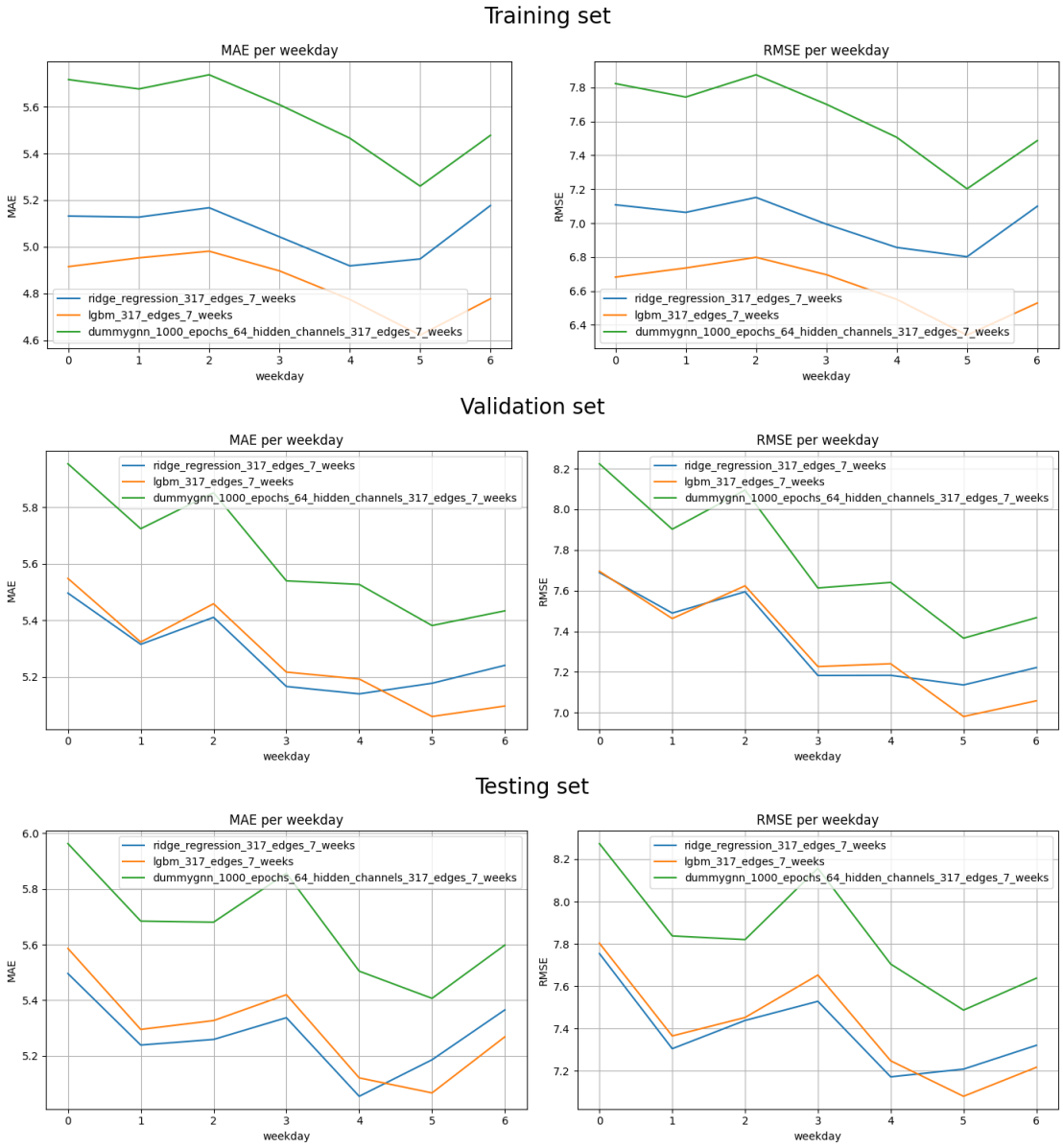

Figure 18. Errors per weekday for RR (blue), GBDT (orange) and GNN (green) models. Plots top to bottom are training, validation and test set metrics. The left column is for MAE, right is for RMSE.

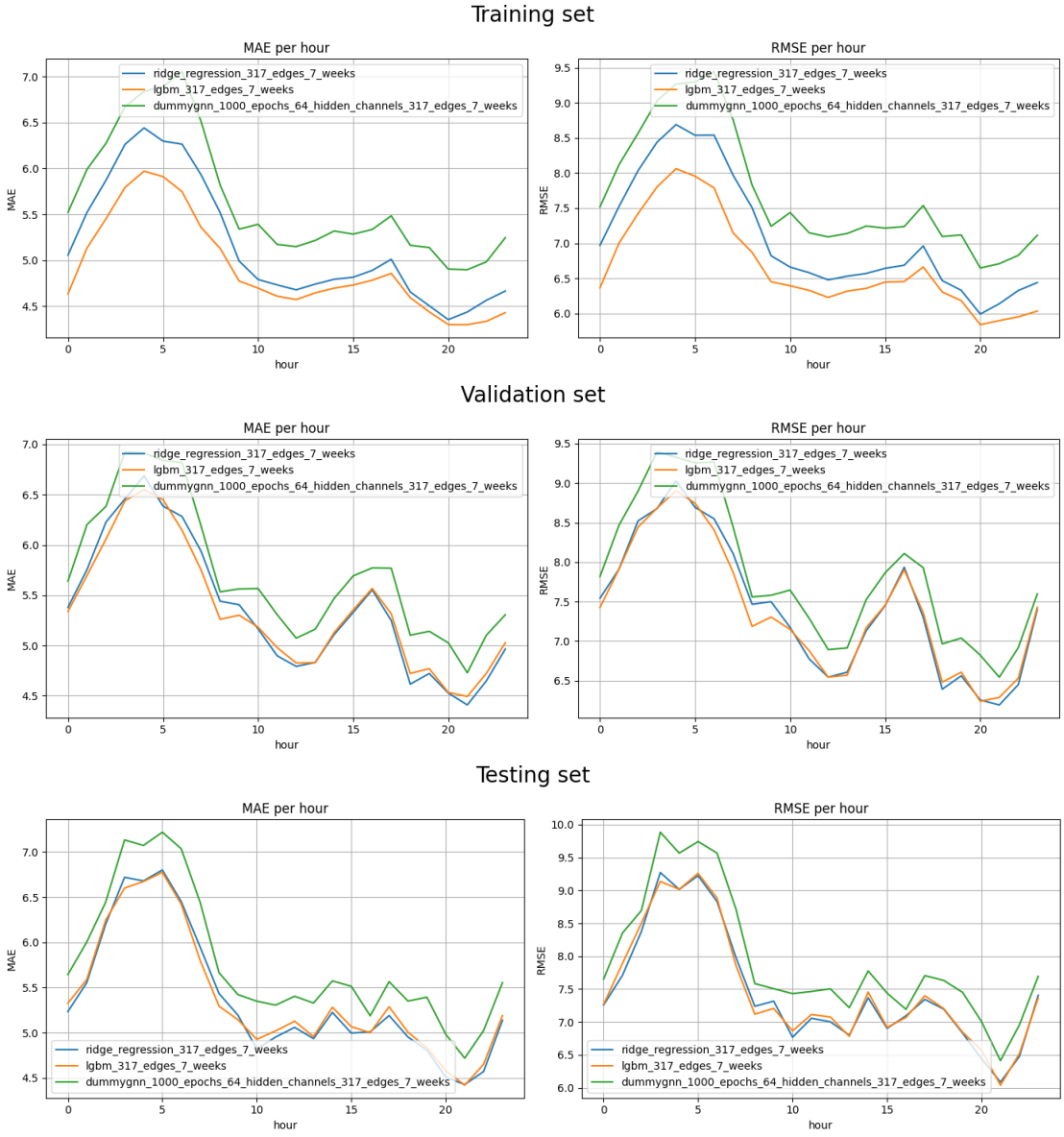

Figure 19. Errors per hour of a day for RR (blue), GBDT (orange) and GNN (green) models. Plots top to bottom are training, validation and test set metrics. The left column is for MAE, right is for RMSE.

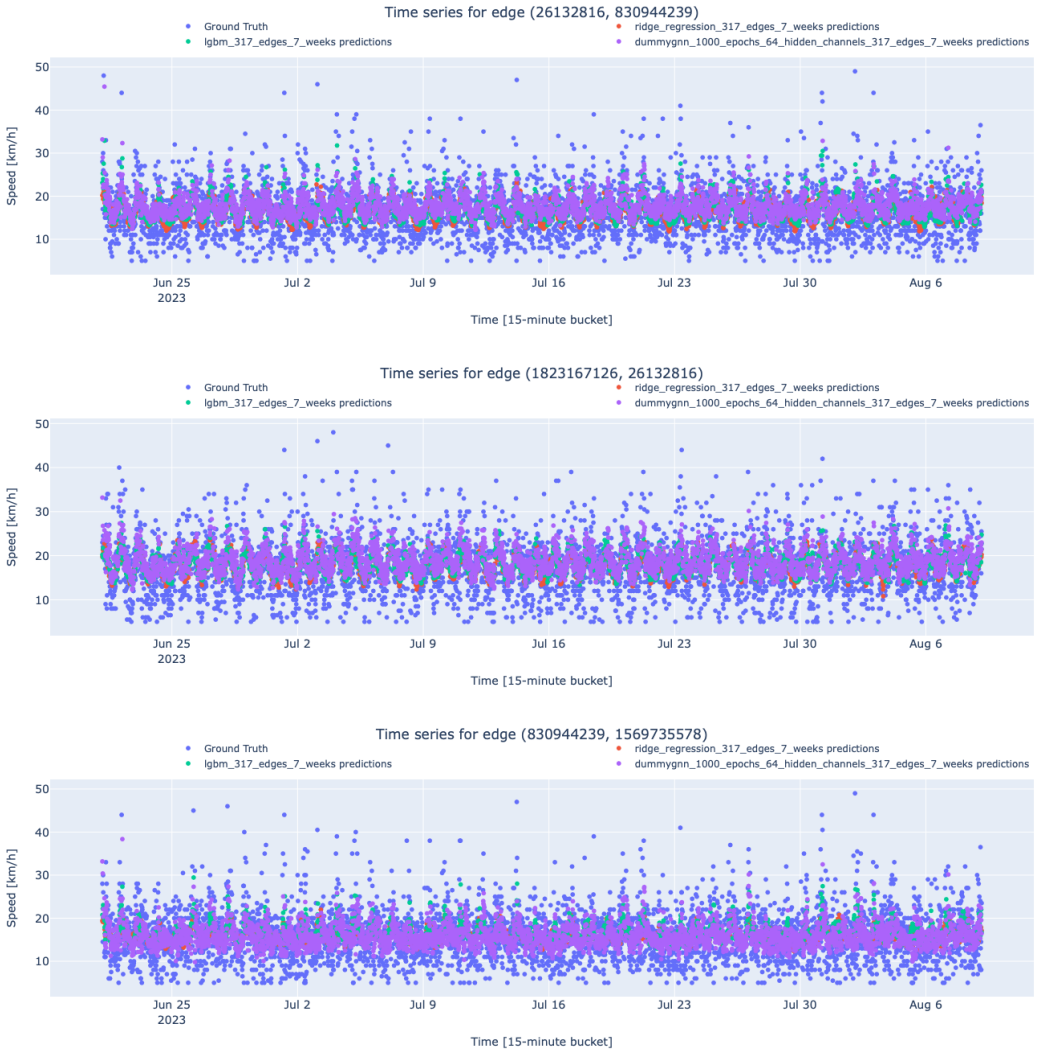

Figure 20. Time series of a randomly selected edge and its neighbours with high measurement coverage. The top plot corresponds to the central edge, and those below are its neighbours. Purple corresponds to DummyGNN, red to RR, green to GBDT and blue is ground truth.

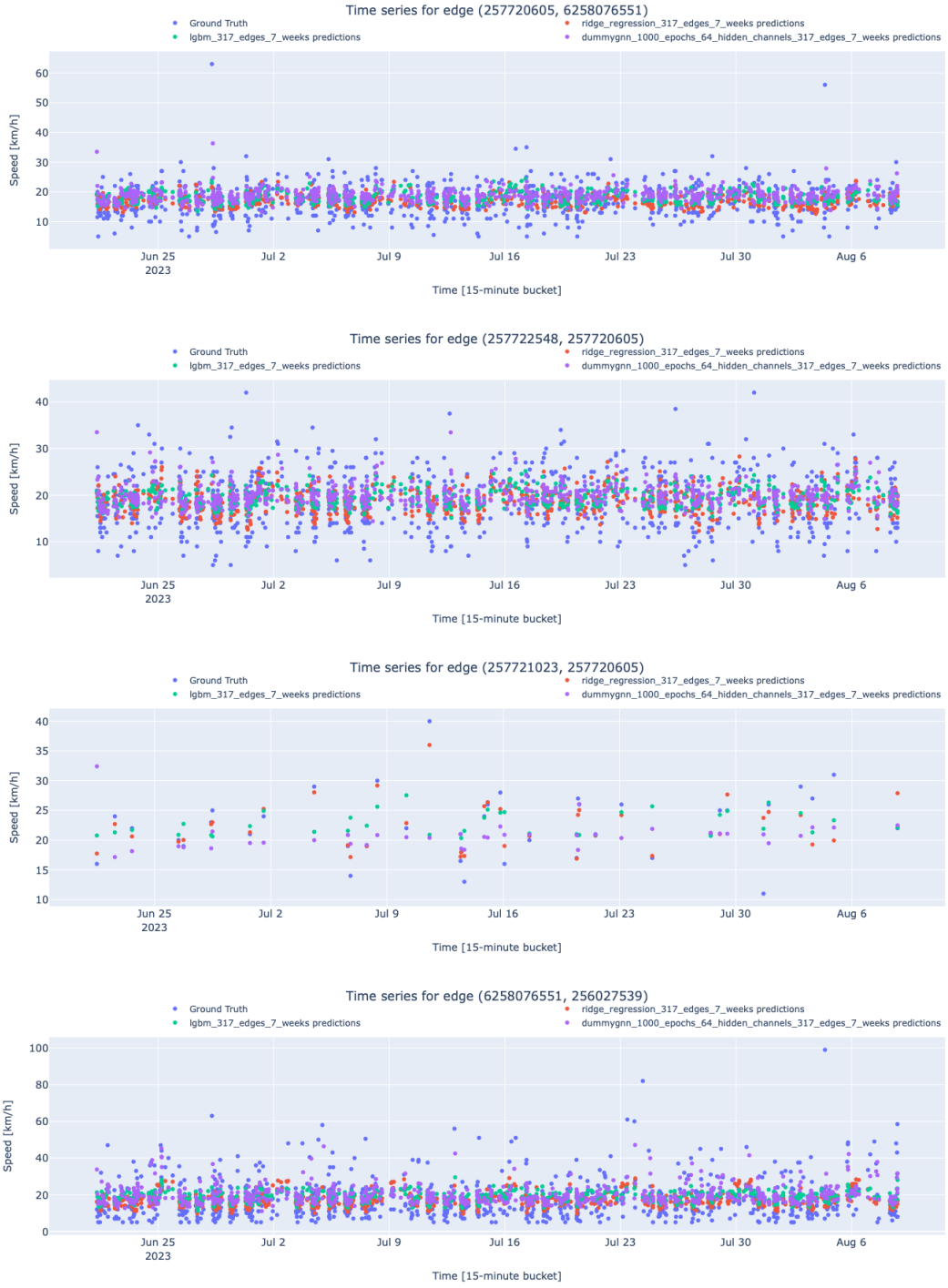

Figure 21. Time series visualisation for a neighbourhood of road segments with relatively sparse data coverage.

Figure 22. Time series visualisation for a sequence of consecutive edges.

4.4 Anecdotal data on spatial dependency learning

We did not manage to show explicitly that our GNN managed to learn spatial dependency. However, we saw some evidence of it that could potentially be further explored. Figure 24 shows a t-shaped neighbourhood of edges, such that one road transitions into the other. Figure 25 shows the corresponding time series for each road segment from figure 24. We can see that the pattern is clear for the second and fourth road segments because there are many samples. Those two segments correspond to a road with a lot of traffic (yellow in figure 24). On the other hand, the first and third plots have much fewer data points. Nonetheless, the GNN predictions marked in green follow roughly the same pattern in all four plots. That indicates that the information about the traffic state was passed within this neighbourhood, and hence spatial relationship was utilised. Yet, more investigation is required to make a confident conclusion.

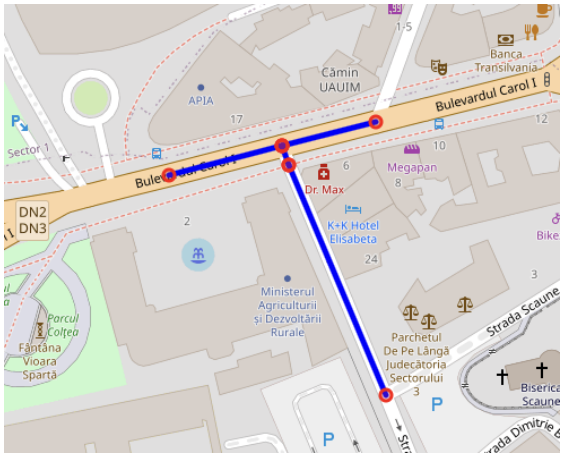

Figure 24. T-shaped road segments.

Figure 25. Time series visualisation for a case with apparent spatial dependency between road segments.

5 Discussion

Our experimental results indicate that attention-based GNN architecture is capable of capturing temporal seasonality. We saw some evidence of learning also a spatial dependency, but more investigation is necessary. We experimented with a state-of-the-art GAT [VCC+18, BAY22] architecture and created variations of it to tackle the TF problem.

Like all DL models, GNN benefits from bigger datasets as we saw in experiments with a varying number of training weeks. We also saw that increasing the number of layers and the number of hidden units (channels) positively impacts model performance. We also demonstrate in this work that a CPU is enough to train a GNN, albeit it is so time-consuming that it would not be possible to apply it in real-world applications. Other models trained significantly faster than the GNN. Consequently, optimisation of processes is required. Also, repeating the same experiments with a GPU could bolster the experimentation pace.

We have shown in our work that RR and GBDT are robust baselines. Although we did not manage to develop a GNN model that would outperform these models we see several ways that could potentially change results. First, data imputation can be improved with more advanced machine learning methods, such as auto-encoders, to learn the representation of the data and then infer missing values. Secondly, the benefit of using a graph structure in the architecture could become more apparent with a larger graph, hence scaling the road network can be promising, albeit optimisation of the procedures is required first. Third, a more rigorous hyperparameter search could yield a more optimal performance. Another potential improvement to the method is eliminating the assumption of a static graph which enforces a rigid dataset structure and requires extra data imputation. Last but not least, developing methods for model interpretability, such as feature importance for a GNN, could help understand how the model works.

In this thesis, we selected MSE as an objective function as it seemed to be suitable for our needs. However, other objective functions can potentially be better. Therefore, it would be useful to conduct an experiment analysing how different objective functions influence GNN training for TF.

We also see that the evaluation procedure needs improvement. Particularly, analyses of the spatial dependency learning by the models is necessary.

Another aspect that we did not cover is GNN generalisation capabilities. Our architecture is constrained to a particular graph we prepared. However, for business needs a generalisable architecture is needed. One of the steps towards such a goal would be a GNN architecture that can work with dynamically changing graphs such that their nodes and edges can appear and disappear from one point to another.

All in all, attention-based GNN is a subject for further research and development which can lead to a new state-of-the-art in TF.

6 Conclusion

In this thesis, we developed a data preprocessing pipeline including an imputation method for filling the missing values in a sparse road network time series data and road network extraction from OSM. Apart from that, we developed regression models including HA, RR, GBDT and a GNN. We created an evaluation procedure to compare the models and found that the GNN that we developed is capable of capturing the seasonality in data and outperforms HA models we implemented, however, its accuracy as measured by MSE is worse than that of the RR and GBDT. We assume that training on a larger graph, with GPUs and additional code optimisation our implementation of a GNN can potentially outperform traditional ML approaches we explored.

7 Acknowledgements

The work was completed during the participation in the Industrial Masters Program with Bolt Technology OÜ. The author acknowledges help and support from the company. The author also expresses a high gratitude to his supervisors Amnir Hadachi and Joonas Puura for their positive attitude, help and guidance.

References

[ASY+19] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[BAY22] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks?, 2022.

[BCB16] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[BZSL14] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs, 2014.

[CGCB14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[CUH16] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.

[DHX+20] Fei Dai, Penggui Huang, Xiaolong Xu, Lianyong Qi, and Mohamadreza Khosravi. Spatio-temporal deep learning framework for traffic speed forecasting in iot. *IEEE Internet of Things Journal*, / Magazine 3:66–69, 12 2020.

[FL19] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[GAG+17] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning, 2017.

[GAGD17] Jonas Gehring, Michael Auli, David Grangier, and Yann N. Dauphin. A convolutional encoder model for neural machine translation, 2017.

[GAW17] Soham Ghosh, Muhammad Tayyab Asif, and Laura Wynter. Denoising autoencoders for fast real-time traffic estimation on urban road networks. 12 2017.

[GD22] Yafeng Gu and Li Deng. Stagcn: Spatial–temporal attention graph convolution network for traffic forecasting. *Mathematics*, 10(9), 2022.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[Inc15] Plotly Technologies Inc. Collaborative data science. Software available from: `https://plot.ly`, 2015.

[JL22] Weiwei Jiang and Jiayun Luo. Graph neural network for traffic forecasting: A survey. *Expert Systems with Applications*, 207:117921, November 2022.

[KMF+17] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: a highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 3149–3157, Red Hook, NY, USA, 2017. Curran Associates Inc.

[KW17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[Lab23] Maxime Labonne. *Hands-On Graph Neural Networks Using Python: Practical techniques and architectures for building powerful graph and deep learning apps with PyTorch*. Packt Publishing Ltd, 2023.

[LH19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[LI22] Martin Lumiste and Andrei Ilie. Large scale traffic forecasting with gradient boosting, traffic4cast 2022 challenge, 2022.

[LL17] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions, 2017.

[LV11] Dennis Luxen and Christian Vetter. Real-time routing with openstreetmap data. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 513–516, New York, NY, USA, 2011. ACM.

[LYSL18] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting, 2018.

[MYWW15] Xiaolei Ma, Haiyang Yu, Yunpeng Wang, and Yinhai Wang. Large-scale transportation network congestion evolution prediction using deep learning theory. *PLOS ONE*, 10(3):1–17, 03 2015.

[NEM+23] Moritz Neun, Christian Eichenberger, Henry Martin, Markus Spanring, Rahul Siripurapu, Daniel Springer, Leyan Deng, Chenwang Wu, Defu Lian, Min Zhou, Martin Lumiste, Andrei Ilie, Xinhua Wu, Cheng Lyu, Qing-Long Lu, Vishal Mahajan, Yichao Lu, Jiezhang Li, Junjun Li, Yue-Jiao Gong, Florian Grötschla, Joël Mathys, Ye Wei, He Haitao, Hui Fang, Kevin Malm, Fei Tang, Michael Kopp, David Kreil, and Sepp Hochreiter. Traffic4cast at neurips 2022 – predict dynamics along graph edges from sparse node data: Whole city traffic and eta from stationary vehicle detectors, 2023.

[NK09] Paul Newson and John Krumm. Hidden markov map matching through noise and sparseness. In *17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2009), November 4-6, Seattle, WA*, pages 336–343, November 2009.

[Ope17] OpenStreetMap contributors. Planet dump retrieved from https://planet.osm.org . `https://www.openstreetmap.org`, 2017.

[PGC+17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[pv] python visualization. Folium.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RSH+21] Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzman Lopez, Nicolas Collignon, and Rik Sarkar. PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management*, page 4564–4573, 2021.

[SLGW20] Chao Song, Youfang Lin, Shengnan Guo, and Huaiyu Wan. Spatial-temporal synchronous graph convolutional networks: A new framework for spatial-temporal network data forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:914–921, 04 2020.

[SLL09] Alex Shevchenko, Max Lytvyn, and Dmytro Lider. Grammar and spelling check with grammarly. Software available from `https://www.grammarly.com/`, 2009.

[SML+22] Maryam Shaygan, Collin Meese, Wanxin Li, Xiaoliang (George) Zhao, and Mark Nejad. Traffic prediction using artificial intelligence: Review of recent advances and emerging opportunities. *Transportation Research Part C: Emerging Technologies*, 145:103921, December 2022.

[SW07] Shashank Shekhar and Billy M. Williams. Adaptive seasonal time series models for forecasting short-term traffic flow. *Transportation Research Record*, 2024(1):116–125, 2007.

[Ten20] Henrikki Tenkanen. Htenkanen/pyrosm: v0.6.0, November 2020.

[VCC+18] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[VSP+23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[WBY+18] Weitao Wang, Yuebin Bai, Chao Yu, Yuhao Gu, Peng Feng, Xiaojing Wang, and Rui Wang. A network traffic flow prediction with deep learning approach for large-scale metropolitan area network. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, 2018.

[WH03] Billy Williams and Lester Hoel. Modeling and forecasting vehicular traffic flow as a seasonal arima process: Theoretical basis and empirical results. *Journal of Transportation Engineering*, 129:664–672, 11 2003.

[YTRJ18] Wu Yuankai, Huachun Tan, Bin Ran, and Zhuxi Jiang. A hybrid deep learning based traffic flow prediction method and its understanding. *Transportation Research Part C: Emerging Technologies*, 90, 05 2018.

[YYZ18] Bing Yu, Haoteng Yin, and Zhanxing Zhu. Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, IJCAI-2018. International Joint Conferences on Artificial Intelligence Organization, July 2018.

[YZT+20] Xiaoxue Yang, Yajie Zou, Jinjun Tang, Jian Liang, and Muhammad Ijaz. Evaluation of short-term freeway speed prediction based on periodic analysis using statistical models and machine learning models. *Journal of Advanced Transportation*, 2020:1–16, 01 2020.

[ZLY+13] Lun Zhang, Qiuchen Liu, Wenchen Yang, Wei Nai, and Decun Dong. An improved k-nearest neighbor model for short-term traffic flow prediction. *Procedia - Social and Behavioral Sciences*, 96:653–662, 11 2013.

[ZSZ+20] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE Transactions on Intelligent Transportation Systems*, 21(9):3848–3858, 2020.

Appendix

I. Visualisation

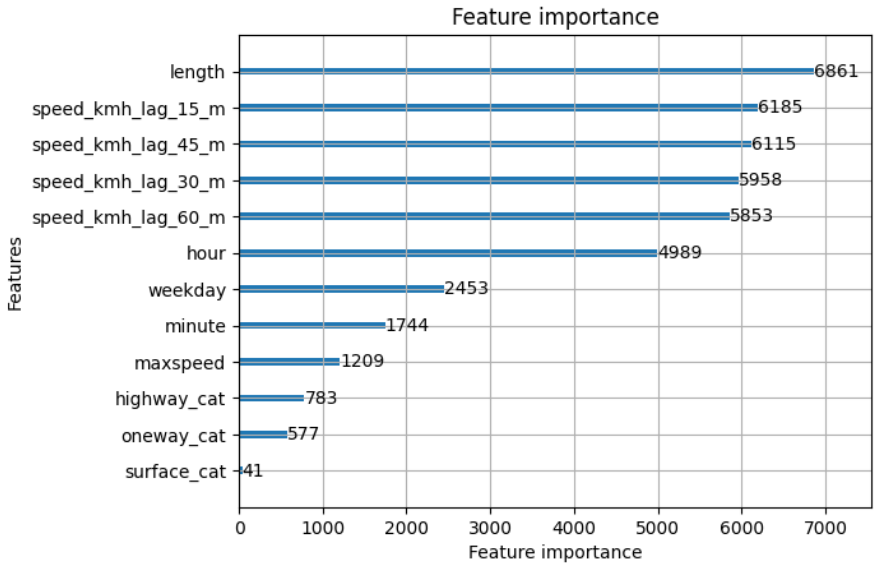

Figure 26. GBDT feature importance by Scikit-learn [PVG+11] native function.

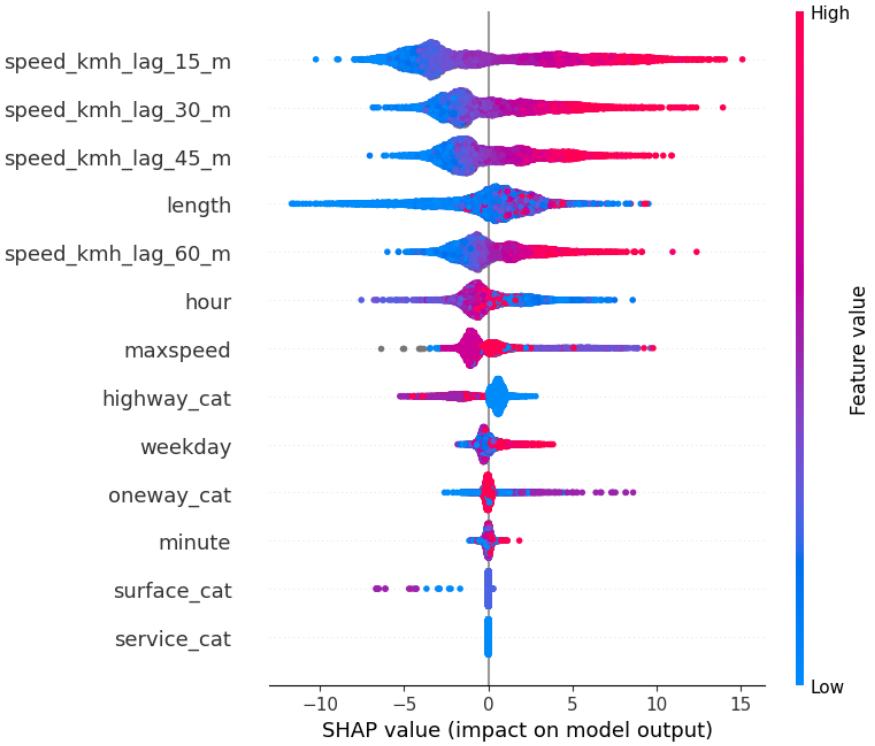

Figure 27. GBDT feature importance using the SHAP values [LL17].

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Pavlo Pyvovar**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Short-term Traffic Forecasting Using Graph Neural Networks on Taxi Data, supervised by Amnir Hadachi and Joonas Puura.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Pavlo Pyvovar
06/05/2024