Trajectory based Road Network Representation Learning

Master Thesis
Master of Science in Computer Science

Paul Heinemeyer
Matriculation Number: 10016184

First Examiner: Prof. Dr. tech. Wolfgang Nejdl
Second Examiner: Prof. Dr. Elena Demidova
Advisor: Stefan Schestakov

November 18th 2022

Eigenständigkeitserklärung

Hiermit versichere ich, die vorliegende Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die wörtlich oder inhaltlich aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I hereby certify that this thesis is my own work and that I have not sought or used inadmissible help of third parties to produce this work, and that I have clearly referenced all sources used. This thesis has not yet been submitted to another examination institution, neither in the same nor in a similar way.

Ort, Datum Unterschrift

> ***If I have seen further than others, it is by standing upon the shoulders of giants.***

Isaac Newton, *Letter to Robert Hooke*, 1675

Acknowledgements

Foremost, I thank profusely my advisor Stefan Schestakov for the continuous support and the nice conversations while writing this thesis. Furthermore, many thanks to Prof. Dr. tech. Wolfgang Nejdl and Prof. Dr. Elena Demidova for reading and evaluating this thesis.

I would like to thank my grandma, who supported me in life long before I could set the first foot in a university and stood by me in good times and bad. The same goes for my parents and my brothers, without whom I would never have gotten this far and who always had an open ear for my problems, beyond the university.

I would also like to give special thanks to my friends Christian Kalfar and Heinz Renziehausen, who helped me a lot to fine-tune this thesis.

Abstract

Road networks, as the core component of urban transportation, are naturally highly dynamic, considering traffic patterns and movement behavior induced by traversing vehicles. Deriving robust road network representations that capture the underlying dynamics and spatial properties of the road network can directly improve the effectiveness of downstream tasks, such as travel time or destination predictions. Although, various approaches in the field of representation learning for road networks have been recently proposed, they focus on spatial properties without capturing the dynamic characteristics.

In this work, we first propose the Graph Trajectory Convolution (GTC) model, which incorporates traffic flow information induced by real trajectory data into the aggregation process. We transfer the idea of modeling traffic flows directly in the learning process to the DeepWalk architecture and subsequently propose the Trajectory Smoothed DeepWalk (TSD) model. We further combine the mentioned models with a transformer architecture, resulting in the Graph Trajectory Network (GTN) model, which additively optimizes the representations to capture multidimensional trajectory traveling behavior and structural properties of the road network. We provide extensive experiments including comprehensive benchmarking, a parameter study, trajectory feature analysis, and an embedding analysis, showing the superior performance of our GTN model across five different intelligent transportation systems (ITS) downstream tasks and three different real-world road networks. The GTN model achieved a 57% higher generalization performance than the best performing baseline model.

Furthermore, we propose another novel model architecture, namely Temporal Graph Trajectory Convolution (T-GTC), which incorporates the temporal dependence of road networks into the road segment representations. This model learns the temporal dependence of the road network by considering dynamic properties, such as driving speed measurements. The T-GTC model achieved a 30.4% higher generalization performance over the downstream tasks and a 7.3% better mean absolute error (MAE) on travel time prediction than the best performing non-temporal baseline, showing that the incorporation of temporal dynamics significantly boosts the performance on temporal dependent tasks.

Contents

Chapter 1

Introduction

Efforts on smartening and improving the infrastructure of cities is increasingly attracting the attention of various institutions due to population growth and increased urbanization. These efforts include the field of intelligent transportation systems (ITS) [1], which deals with the analysis, prediction, and mining of data from road networks. Various ITS task, such as travel time estimation, flow prediction, traffic estimation, or destination prediction are fundamentally based on road networks. To achieve robust performance on different downstream tasks, it is crucial to learn generalizable and robust representations of such road networks. Learning representations that can capture various aspects of a road network, including spatial, temporal as well as properties induced by vehicle behavior, can boost the performance of tasks based on road networks. However, learning such representations is difficult due to a low number of features, network sparsity, and volatile homophily.

Since a road network is essentially a graph structure, which models road segments as edges and intersections as nodes, it is a natural choice to apply graph-based machine learning approaches to this problem. Current state-of-the-art (SOTA) Graph Neural Network (GNN) models, such as the Graph Convolutional Network (GCN) are designed for node classification tasks in citation or social networks, which commonly assume network homophily. That property describes a network, in which connected nodes tend to be similar in contrast to distant nodes regarding their attributes. This assumption mostly does not hold for road networks because spatially neighboring nodes might not share similar properties and traffic patterns. For example, a primary road segment that is connected to various secondary road segments commonly has different properties in contrast to the connected secondary segments. This includes traffic volume, driving speed restrictions, or the length of the road segment. Furthermore, compared to citation or social networks, road networks have a low average node degree, i.e., the edge density is low. For example, the average node degree of Porto's road network is 2.35, which is extremely sparse compared to commonly used benchmarking citation or social graphs with an average node degree of 3.9 up to 64 and a max degree between 168 and 3992 [2]. The sparsity and the additional lack of rich attributions on the road segments can lead to issues similar to oversmoothing [3], since the aggregation function of standard graph convolutional methods would learn an identity function [4]. Hence, standard GNN models are not suitable for road network representation learning.

Recent work in the field of road network representation learning explored various approaches to handle the mentioned issues regarding the unique properties of road networks. The model proposed in [5], namely SRN2Vec, learns road segment representations by simultaneously predicting geo-locality and common properties of road segments. The model proposed in [4] introduces an auxiliary traffic aware skip-gram model, which predicts context windows generated from random walks and furthermore incorporates traffic context by predicting properties of the road segments to learn road segment representations. Other approaches [6, 7] directly apply GNN models to road networks to learn road segment embeddings. However, the mentioned SOTA models suffer because of the volatile homophily and sparsity of these graphs. Furthermore, all the mentioned models do not incorporate travel semantics into the representation learning of road segments, which is crucial for ITS tasks, such as travel time estimation or destination prediction. Another issue of current approaches is the lack of inclusion of the temporal dimension. Road network properties, such as traffic volume or driving speed, are highly dependent on temporal features. For example, the traffic volume on most road segments is higher during rush hours in comparison to night hours. Recently proposed models learn representations based on road network structure without considering the underlying dynamic of road networks induced through traveling semantics and temporal dependence. We argue that it is crucial to incorporate these dynamics to learn task robust representations of road segments.

In this thesis, we propose several model variants, namely GTC, TSD, GTN and T-GTC. The GTC model is a convolutional model that directly incorporates travel semantics into the aggregation process. The incorporation is accomplished by aggregating information along trajectories while weighting the importance of each road segment by the probability of occurrence on a trajectory. This allows both oversmoothing and sparsity to be addressed while additionally modeling transition flows. We further transfer the idea of modeling transition flows to the DeepWalk model, which we extend by a transition flow matrix. This matrix weights the edges inside the road network by the probability of occurrence on trajectories and therefore models the direct transition flows. The random walks in the training process are sampled by directly using the transition matrix, which results in learning real road network behavior in addition to structural properties. To combine the proposed models into a single embedding generating model, we propose the GTN model, which applies a transformer to fine-tune the combination of the learned representations. Specifically, the transformer is initialized with the pretrained embedding and fine-tunes it, by learning to reconstruct trajectory sequences. This further induces traveling semantics into the learned representation and makes the representation more robust for different trajectory-based downstream tasks, such as destination or next location predictions. Finally, we propose the T-GTC model which additionally incorporates temporal dynamics into the road segment representations. Fundamentally, the model combines the GTC model with a Long-Short Term Memory (LSTM) model. This enables the model to incorporate temporal dynamic properties, such as traffic volumes or driving speed measurements. The benefit is that we directly model temporal behavior inside the embedding, which is fundamental for tasks that are highly time dependent, such as travel time predictions. Furthermore, we propose an attention-based extension to the base T-GTC model, which focuses on learning global dynamics of the temporal behavior.

This is the first work that directly incorporates trajectory semantics into road segment representations. We contribute novel model concepts and comprehensive evaluations in this thesis. All proposed models are evaluated in conductive experiments on five real world downstream tasks and three different datasets against several SOTA baselines. We further provide an extensive evaluation of the GTN, GTC and TSD model, which shows the advantages of incorporating traveling semantics into the learning process of road segment representations. The results showed that our models outperform current SOTA road network representation learning methods and standard graph models across tasks and datasets. Specifically, the combination of the GTC and TSD generated embeddings showed superior performance against combinations of technically similar models. We achieved even better results by further integrating trajectory information into the combined embeddings through the GTN model.

Furthermore, this is also the first work on incorporating temporal dynamics into road segment representations. We evaluate our T-GTC model and its variant on four different ITS downstream tasks, including structural and temporal dependent tasks, against SOTA static models and a SOTA temporal model from the literature. We show that our proposed T-GTC is highly superior on time-dependent tasks, like travel time prediction, in contrast to current SOTA static models, while it does not lack performance on static structure dependent tasks.

This thesis is structured into nine chapters. Following the introduction, chapter 2 introduces the addressed problem of the thesis and comprehensively describes the goal. We continue by explaining recent efforts in the field of graph-based machine learning and road network representation learning in chapter 3. Afterwards, the preliminaries in chapter 4 will cover the necessary fundamental knowledge on which this thesis is built. This includes knowledge about OpenStreetMap (OSM), graph-based machine learning and transformers. Subsequently, the approach explains how we prepared the utilized datasets. Furthermore, the chapter introduces our novel models by explaining the idea and architecture on which they are based. In the evaluation chapter, we extensively evaluate the proposed models against current SOTA models from the literature and commonly used graph-based baselines. This includes experiments on various ITS downstream tasks, an ablation study, a parameter study, an explanatory data analysis, a feature analysis and an embedding analysis. Additionally, in chapter 7 we conduct a second evaluation on our proposed T-GTC model and its variant, which incorporate dynamic temporal properties. As in the main evaluation, we compare our models with a SOTA model from recent literature and common graph-based methods. This is followed by a discussion of our findings and the limitations of this thesis in chapter 8. Finally, we conclude the thesis in chapter 9 with additional remarks on possible future work. In the appendix, instructions to reproduce the results (chapter B) and additional approaches with unpromising results (chapter A) can be found.

Chapter 2

Problem Statement

In this work, we target the problem of learning generic and robust representations of road segments inside a road network using trajectories, which can be used for various downstream tasks such as road label classification, travel time estimation, or destination prediction.

Definition 1. Road Network
A road network $G_{rn}^{L} = (V, E, F_v)$ is a directed line graph [8, 9], where V is a set of vertices and E is a set of directed edges $E \subseteq (V \times V)$. Each vertex $v \in V$ represents a road segment and each edge $e = (u, v)$ represents an intersection between the road segments u and v. In addition, a road network has a feature set F_v representing features on road segments, such as coordinates or road type. The line graph $G_{rn}^{L} = L(G_{rn})$ is generated from the original graph $G_{rn} = (V, E, F_e)$, where road segments are represented as edges and intersections are represented as nodes.

For example, the features of a road segment could be represented as follows (the example is from three road segments in Porto and illustrates features that can be extracted from OSM [10]):

Id	**highway**	**oneway**	**maxspeed**	**length**
479127843	*motorway_link*	*true*	–	32.4
4256507	*motorway_link*	*true*	60	115.1
734630278	*secondary*	*true*	–	48.6

To model the traffic behavior as realistically as possible, we use trajectories generated in the form of Global Positioning System (GPS) data by different road vehicles. Examples of road vehicles are cabs or passenger cars.

Definition 2. Trajectory
A trajectory $T = \{p_1, p_2, ..., p_n\}$ is an ordered sequence of spatio-temporal points generated from cars driving trips. Each point $p_n = (x_n, y_n, t_n)$ in a trajectory contains the geo-information (x_n, y_n) (i.e., longitude and latitude) and a timestamp t_n.

For instance, the trajectory of a cab could look like this:

Id	timestamp	longitude	latitude
52	1372636858	−8.618643	41.141412
52	1372636876	−8.618499	41.141376
52	1372636891	−8.620326	41.14251

A set of trajectories is defined as $D = \{T^{(i)}\}_{i=0}^{|D|}$. The traversed road segments, i.e., streets in the road network following the GPS points of a trajectory T are defined as a road segment sequence.

Definition 3. Road Segment Sequence
A road segment sequence $R = \{r_1, r_2, ..., r_n\}$ is a sequence of road segments from a road network, where $r_i \in V$ represents the *i*-th road segment.

A road segment sequence can be generated from a trajectory using a map matching algorithm [11, 12] or by sampling from the road network. A set of road segment sequences is defined as $P = \{R^{(i)}\}_{i=0}^{|P|}$. Then the problem of learning road segment representations is defined as follows:

Problem Statement. Given a road network G_{rn}^{L}, trajectories D, and the corresponding road segment sequences P, the road segment representation learning aims to learn a function $f_v : V \to \mathbb{R}^d$ which maps each road segment $v \in V$ to a low dimensional vector representation, i.e., an embedding.

The goal is to learn generic and robust embeddings that capture the geo-locality and homophily relationships between road segments, but furthermore incorporate traversing behavior induced by trajectories. The resulting representations should be usable for various ITS downstream tasks, while constantly achieving high performance.

Chapter 3

Related Work

Representation learning [13] has received considerable attention in the field of machine learning recently. The goal of this technique is to compress data into a low-dimensional vector that reflects certain properties in a latent space, such as the similarity between data objects. To generate these embeddings, neural networks can be used, which have led to great successes in different areas of machine learning like computer vision [14], natural language processing (NLP) [15], and graph-based learning [16, 17].

Methods on graphs, which are known under the term GNNs, are used to embed the nodes and their features into a low dimensional representation. Most modern methods focus on capturing different graph properties like homophily [18] or proximity, which means that nodes that are closer to each other in the graph are more similar and therefore should be closer to each other in the embedding space.

Earlier approaches [19, 20, 21, 22, 23], which use random walks to embed nodes and graph structure, include DeepWalk [21] and Node2Vec [19]. DeepWalk uses an unbiased random walk and trains a skip-gram [24] model by predicting the local neighborhood based on the random walk samples. Node2Vec uses a biased random walk. The biased random walk is controlled by two parameters p and q, which control the probability of the random walk going back or exploring forward. Going back simulates breadth-first search, and exploring forward simulates depth-first search. With this sampling strategy, more diverse neighborhoods can be efficiently explored [19].

Great successes were subsequently achieved with convolution-based methods, especially in the field of computer vision and signal processing [25]. The idea of convolutions was later adapted and applied to graph structures. The basis of most modern methods is the GCN [26], which aggregates information (features) from neighboring nodes. This aggregation is propagated forward with each layer to enable the final prediction. Graph Attention Networks (GATs) [27, 28] is another variant of a graph neural network that uses attention [29] in the aggregation process to weight the neighborhood of a node by importance. GraphSage [30] generates inductive node embeddings by sampling and aggregating feature information from neighboring nodes. The proposed framework uses learnable aggregation functions, which are applied at inference time to generate embeddings for entirely unseen nodes. However, these methods are designed for general networks and not for road networks, which show a discrepancy to common graphs regarding certain assumptions such as homophily [4].

Recently, several extensions to graph representation learning methods specifically for road networks [4, 5, 6, 7, 31, 32] have been proposed to incorporate the special characteristics of road networks, such as geo-locality of road segments, into the embedding process. RN2Vec [5] jointly learns embeddings for road segments and intersections while considering the geo-locality, homogeneity, and topological structure of the road network. Another framework named Toast [4] learns robust representations that can be used in road segment and trajectory-based applications. This approach uses a traffic-aware skip-gram module to predict traffic context for road segments and a trajectory-enhanced transformer [29] module to capture traveling semantics on road networks.

Different from previous works on representation learning for road networks, our approach incorporates the travel semantic data in the form of trajectories directly into the road segment embedding generation process. Thus, effective, realistic, and robust road segment representations are learned, which can be applied to various ITS applications.

Chapter 4

Preliminary

This chapter explains the basic theoretical concepts that are essential for the work in this thesis. The first section introduces OpenStreetMap and explains its basic components. Afterwards, a closer look at the skip-gram and Node2Vec model is given, followed by a presentation of current graph convolution-based methods. Finally, a detailed explanation of the idea and functionality behind the transformer model is presented.

4.1 OpenStreetMap

This section gives an overview of how OSM [10] stores and manages data. We describe the relevant data types and their relationship with each other.

Overview

OSM is an open-source project founded in 2004 by Steve Coast. The project provides user-generated geographic information in the form of a free, editable map of the world. Figure 4.1 shows an example map section of Hanover taken from OSM.

Figure 4.1: Extracted map section of Hanover, taken from OSM.

Through growing user participation, the information became much more accurate and comprehensive. There are now over 8 million registered accounts on OSM and several editing tools to enable user contributions.

The data provided by users must follow defined principles. The base object is called an element. This abstract concept is subdivided into more specific types, including nodes, ways, and relations. In addition, these data elements can be assigned tags that describe their properties and meaning. Since nodes and ways are of particular interest for this thesis, they will be described in more detail in the following sections.

Nodes

An OSM node [33] represents a single point in space defined by an id and coordinates in the form of longitude and latitude. Optionally, an altitude and tags in the form of key-value pairs can be included. For example, a node defining a roundabout between streets in Porto is visualized in 4.2.

$$\begin{bmatrix} \textbf{\textit{osmid}} & 25620759 \\ \textbf{\textit{latitude}} & 41.148801 \\ \textbf{\textit{longitude}} & -8.612139 \\ \textbf{\textit{street\_cou}} & 3 \\ \textbf{\textit{highway}} & mini\_roundabout \\ \textbf{\textit{ref}} & None \end{bmatrix}$$

Figure 4.2: Representation of a node from Porto defining a roundabout. The red circle on the left shows the geographic position on a map. The right part shows the corresponding representation in OSM.

Ways

A way [34] in OSM represents a linear feature on the ground, which can be a road, river, or other objects that can be represented as a line. A way is built up by an ordered list of nodes defining the line. There exist open ways and closed ways. Unlike open ways, closed ways have the same node as the start and end points. For example, closed ways can represent areas. In our work, we will use roads, which are mostly represented by an open way. Figure 4.3 shows an example of a way in Porto extracted from OSM.

osmid	838000172
u	3139458017
v	1725589743
oneway	$true$
highway	$primary$
length	885.775
lanes	2
ref	$EN12$
maxspeed	50
name	Estrada Interior...

Figure 4.3: Representation of a way from Porto, defining a road by two nodes u and v. The red line on the left shows the geographic representation on a map. The right part shows the corresponding representation in OSM.

Tags

An OSM tag [35] is a key-value pair, which describes the properties of an element (node, way, relation). The key is used to describe the topic or type of feature, and the value provides details for the feature. Both key and value are free-format text fields, which often represent numeric or structural items. OSM offers conventions on the meaning and use of tags, which can be viewed in the OSM Wiki [36]. Those conventions are not mandatory, resulting in a wide variety of different tags.

4.2 Skip-Gram

The skip-gram model was originally proposed in the context of NLP [24]. It is used to learn representations for words based on the similarity between them. The model was also proposed for learning node embeddings in the context of graph representation learning (see section 4.3) where nodes are treated like words and edges are treated like links between neighboring words in a sentence.

The main intuition behind the skip-gram model is that words appearing in the same context tend to have similar meanings and hence should have similar vector representations [37]. Given a sentence and a target word inside, the model is trained to predict words that surround the target word. Figure 4.4 shows how the training samples are generated for a single sentence.

Architecturally, the skip-gram model consists of a single fully connected hidden layer with a softmax activation function [38] as output. For each word in the vocabulary, the weight matrix $W \in \mathbb{R}^{|V| \times d}$ of the hidden layer contains a d-dimensional row, which represents the vector representation for the respective word. The input is a one-hot vector, which has the length of the number of words in the vocabulary. All entries in the vector are zero except for a single one, which indicates the corresponding word in the vocabulary and the weight matrix. The model is trained to optimize the probability of predicting the correct neighboring words, i.e., the context.

This can be seen as maximizing the average log-likelihood:

$$\frac{1}{T}\sum_{t=1}^{T}\log(P(v_{t-c},\ldots,v_{t+c}|v_t) \tag{4.1}$$

where T is the word length, c is the context size and v_t denotes the current target word in the training sentence at index t.

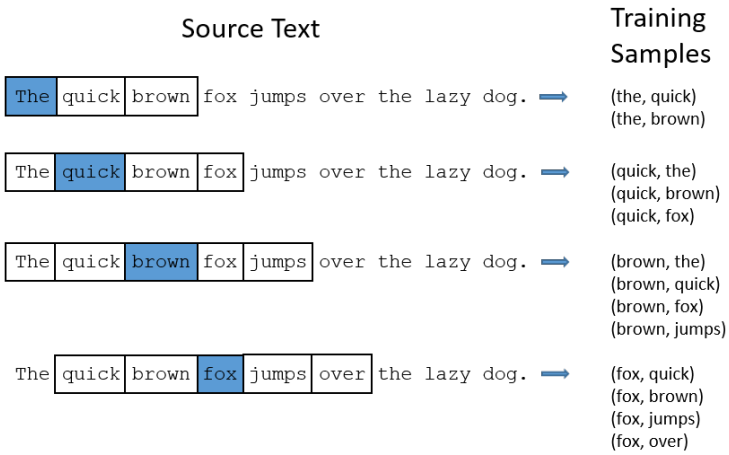


Figure 4.4: Example of the training samples generation for a context size of $c = 2$. The example is for a single sentence, and the word highlighted in blue marks the target word [39].

The output of the model is the softmax probability vector containing, for each word in the vocabulary, the probability that the word is in the context of the target word. After the training the word embeddings can be extracted from the hidden layer, i.e., the embedding is the weight matrix W of the hidden layer. The overall described architecture can be seen in figure 4.5. Since in the standard setting the skip-gram model updates all weights for each training sample, which gets more computationally expensive with larger vocabulary size, negative sampling [40] can be used to improve training time. The basic idea behind this concept is instead of updating all weights, each training sample updates only a small percentage of the weights. This fraction is determined by n randomly selected negative samples (in praxis n is set to be between $5 - 20$ [24]). Negative samples are words that are not in the context of the target word, i.e., words where the probability in the softmax vector should be zero. The weights are then only updated for the positive and the selected negative words.

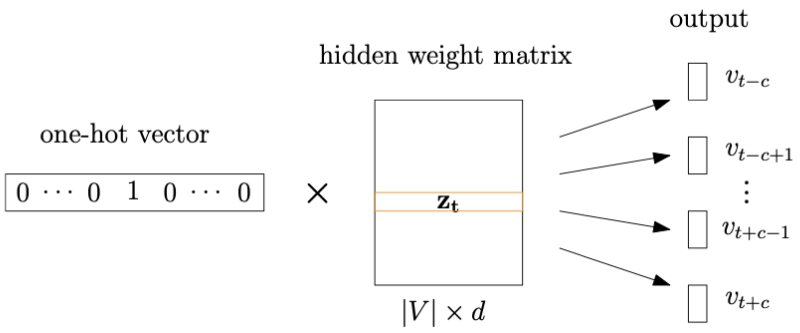


Figure 4.5: Architecture overview of the skip-gram model [41].

4.3 Node2Vec

The Node2Vec model [19] is an algorithm based on biased random walks. It learns continuous feature representations for nodes in graphs while maximizing the likelihood of preserving the neighborhood structures of nodes. A normal random walk traverses the graph step by step while deciding at each node where to travel next based on a transition probability, which is uniform over all neighboring nodes for a standard random walk. Node2Vec combines two searching strategies, namely depth-first search (DFS) and breadth-first search (BFS) [42] which results in a biased random walk balancing an exploration and exploitation tradeoff. In BFS the walk focuses on local nodes, while in DFS the random walk is sequentially building up with increasing distance to the starting node, which results in an exploration of larger parts of the graph. Figure 4.6 shows the BFS and DFS search strategies on an example graph.

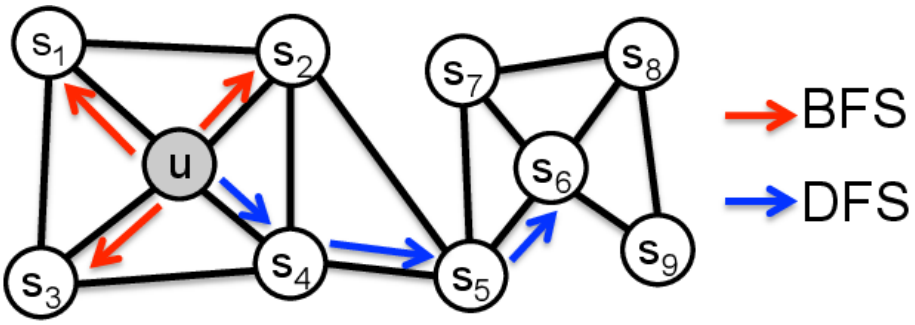


Figure 4.6: BFS and DFS search strategies starting from node u with walk length $l = 3$ [19].

The biased random walk is a second-order random walk that reweights the transition probabilities depending on the previous state t and potential next state x with a bias factor α, which is set to $\frac{1}{q}$ if the two states are not connected and $\frac{1}{p}$ if the two states are the same, i.e., the walk is going back to the previous state t. If the two states are connected, α is set to 1. See equation 4.2 for a formal definition, where d_{tx} denotes the shortest path distance from node t to node x.

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p}, & \text{if } d_{tx} = 0 \\ 1, & \text{if } d_{tx} = 1 \\ \frac{1}{q}, & \text{if } d_{tx} = 2 \end{cases} \tag{4.2}$$

The q parameter is called *in-out* parameter and controls the probability of visiting nodes far away from the current node, i.e., moving outwards the local neighborhood. The p parameter is called *return* parameter and controls the probability of revisiting the current state in the next step. Figure 4.7 visualizes this procedure for an example step. After generating the walk sequences, the model applies skip-gram (see section 4.2) to learn the representations.

When p and q are both set to 1, the model performs a first-order random walk, i.e., the strategy used by DeepWalk [21]. This makes the model flexible in terms of the search strategy.

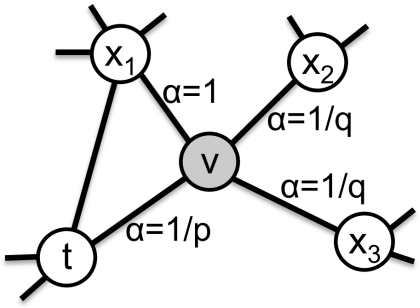


Figure 4.7: Visualization of the random walk procedure in Node2Vec with search bias α. The walk transitioned from node t to node v and evaluates the next step from node v [19].

4.4 Graph Neural Networks (GNN)

GNNs in general are a set of methods that apply deep neural networks to unstructured data in the form of graphs. They can be seen as a process of representation learning on graphs. At the node level, GNNs aim to learn representative features for each node such that the target task can be facilitated [43]. In the following sections, various SOTA methods from the literature are explained in detail.

4.4.1 Graph Convolutional Network (GCN)

The GCN model [26] is a generalization of Convolutional Neural Networks (CNNs) [44] for unstructured data in the form of graphs. It is based on spectral graph convolutions [45, 46] and simplifies this approach, leading to faster training times and higher predictive accuracy.

A GCN uses, similar to a CNN, convolutions to aggregate features of neighbors in graphs. The major difference is that the convolution in a CNN works on regular structured euclidean data and a GCN works on irregular unstructured data. Figure 4.8 visualizes the difference between the two convolutions. The red node is the target node, and the blue boundary marks the area where the features of the nodes inside are aggregated. For a CNN this is always structured and for a GCN the node count inside the filter is dependent on the neighbor node count of the target node.

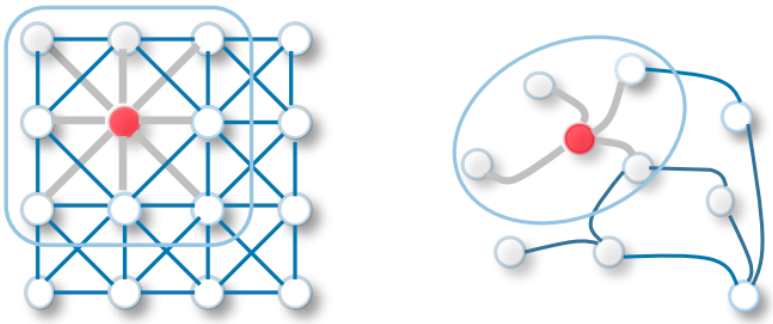


Figure 4.8: Comparison of 2D CNN (left) and GCN (right) [47].

A GCN model takes as input a feature matrix $X^{N\times D}$ and an adjacency matrix $A^{N\times N}$, where N is the number of nodes and D is the number of features. The feature transformation with neighborhood aggregation in a single GCN layer for a single node can be described as follows:

$$h_v^k = ReLU(W_k \sum_{u\in N(v)\cup v} \frac{h_u^{k-1}}{\sqrt{|N(u)||N(v)|}}) \tag{4.3}$$

where v is the target node h^k is the k-th hidden layer, ReLU [48] is a nonlinear activation function, W_k is a learnable weight matrix with parameter sharing across the layer, and $\sum_{u\in N(v)\cup u} \frac{h_u^{k-1}}{\sqrt{|N(u)||N(v)|}}$ is the aggregation of the $k-1$ hidden state of the neighboring nodes from v and v itself, which is normalized by the corresponding neighborhood count. Each neighbor contributes equally to updating the representation of the target node. Stacking multiple GCN layers corresponds to aggregating the k-hop neighborhood of the nodes. After the aggregation, this method produces a node-level output $Z^{N\times F}$ also known as node embedding, which represents the learned features for each node in the graph, where F is the number of feature dimensions.

4.4.2 Graph Attention Network (GAT)

A GAT model [27], like GCN, uses a convolution-based approach. The difference is that here a learnable attention mechanism [29] is incorporated to weight the contribution of the neighbors in the aggregation process. The function to compute the latent representation h of the node i in a single layer is defined as follows:

$$h_i = ReLU(\sum_{j\in N(i)\cup i} \alpha_{ij} W h_j) \tag{4.4}$$

where $N(i)$ are the neighbors of node i, α_{ij} is the attention value between node i and node j, W is a learnable-shared weight matrix and h_j is the hidden state of node j. The calculation of the attention value is defined as follows:

$$\alpha_{ij} = \frac{\exp(LeakyReLU(a^T[Wh_i \| Wh_j]))}{\sum_{k\in N(i)} \exp(LeakyReLU(a^T[Wh_i \| Wh_k]))} \tag{4.5}$$

where a^T is a learnable weight vector, W is a learnable and shared weight matrix (the same as in equation 4.4), $\|$ is the concatenation operator and LeakyReLU [49] is a non-linear activation function. The term $LeakyReLU(a^T[Wh_i \| Wh_j])$ can be seen as a single feed-forward network that outputs the attention score between nodes i and j. The main advantage in contrast to GCN is to learn the different contributions from each neighborhood node.

Furthermore, it is also possible to apply multiple attention heads to enable simultaneously focusing on multiple important nodes. This is made possible by incorporating a multi-attention mechanism into the aggregation. This way, the different aggregated latent representations are concatenated from the attention heads, and the average is taken in

the last layer (see figure 4.9 for visualization of the procedure). Formula 4.6 shows the adapted multi-attention head version of formula 4.4,

$$h_i = \Big\|_{k=1}^{K} ReLU\Big(\sum_{j \in N(i) \cup i} \alpha_{ij}^k W^k h_j\Big) \tag{4.6}$$

where $\|_{k=1}^{K}$ is the concatenation operator used on K attention heads. This operator is replaced with a mean aggregator function in the last layer to preserve the output dimension.

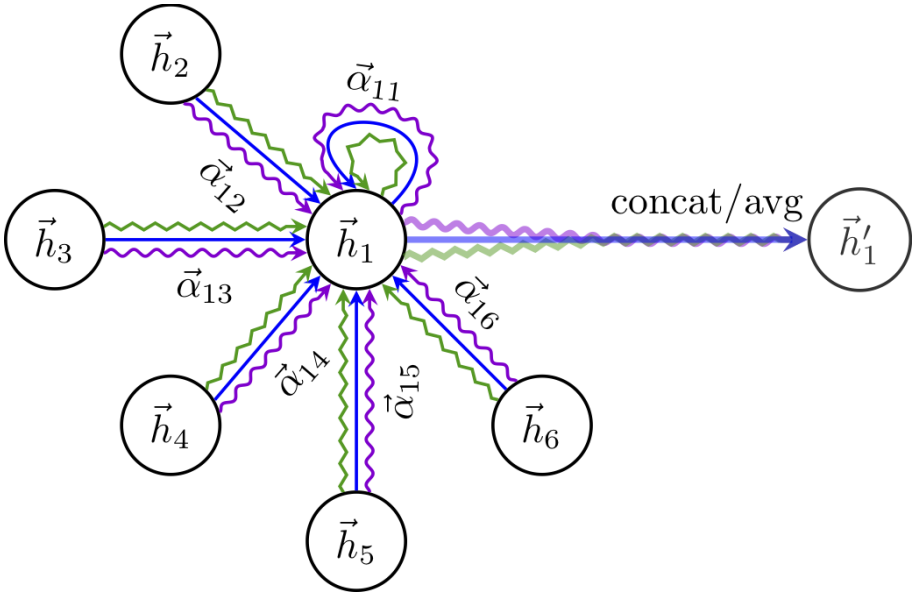


Figure 4.9: Visualization of multi-head attention mechanism (with $K = 3$ heads), where each arrow style denotes a different independent attention calculation [27].

4.4.3 Graph Auto Encoder (GAE)

A Graph Auto Encoder (GAE) [50] is a special variant of the traditional autoencoder [51, 52], which uses a neural network with an encoder and decoder part. The encoder takes a data sample X as input and outputs a low dimensional representation Z called embedding. The reverse operation is done by the decoder, which reconstructs the original input X' from the embedding Z. The goal is to capture as much information as possible into the embedding Z to get a good reconstruction X'. Figure 4.10 shows the autoencoder architecture on an example for image representation learning. The encoder takes an image as input and generates a compressed representation. The decoder takes the compressed representation and outputs the reconstructed image. For traditional autoencoders, the loss is measured as the difference between the original input and the reconstruction, known as reconstruction loss. In most cases, this is measured with the mean squared error (MSE) loss function [53].

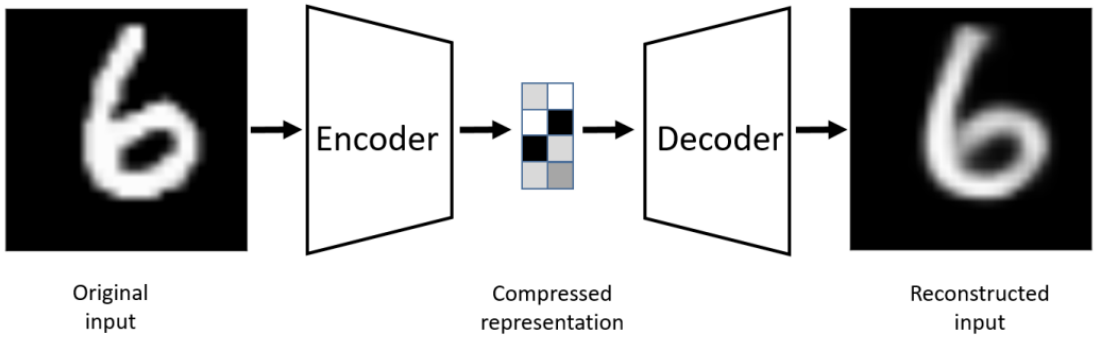


Figure 4.10: An autoencoder example for representation learning on images. The input image is encoded to a compressed representation and then decoded to a reconstruction of the original input image [54].

A GAE extends this idea to graph-structured data. This model takes as input the adjacency matrix A of a graph, which represents the graph structure, and a feature matrix X. In contrast to a normal autoencoder, it uses a GCN (see section 4.4.1) as encoder. The output of the encoder is the embedding representation Z. The decoder is an inner product function defined as:

$$\hat{A} = \sigma(ZZ^T) \tag{4.7}$$

where $\hat{A}$ is the reconstructed adjacency matrix and ZZ^T is the inner product between the latent representation Z, which calculates the pair-wise distance between the node embeddings, followed by a logistic sigmoid function $\sigma(\cdot)$. The network is trained to minimize the difference between the original and reconstructed adjacency matrix. There are several variants of the standard GAE proposed in the literature. For example, variants that incorporate regularization [55, 56] to generate a more robust embedding of the graph structure.

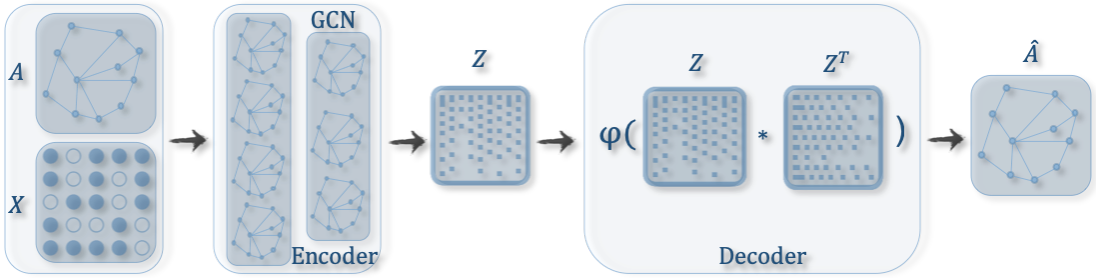


Figure 4.11: The architecture of a Graph Auto Encoder (GAE). The input adjacency matrix and corresponding feature matrix are encoded to a compressed representation and then decoded with an inner product calculation followed by a non-linear activation function to reconstruct the adjacency matrix [47].

4.5 Transformer

The transformer [29] is an encoder-decoder model, which was proposed to overcome the sequential nature of Recurrent Neural Networks (RNNs) [57]. RNNs process sentences word-by-word, which results in a computational overhead. The transformer model employs a self-attention mechanism to parallel account for every word in the input sequence. The originally proposed model consists of six encoder and decoder layers. A single encoder layer is composed of a multi-head self-attention mechanism, normalization, a feed-forward network, and residual connections [58]. The decoder part adds another multi-head self-attention sub-layer. Figure 4.12 shows the overall architecture of the transformer, where the left part shows a single encoder and the right part a single decoder layer.

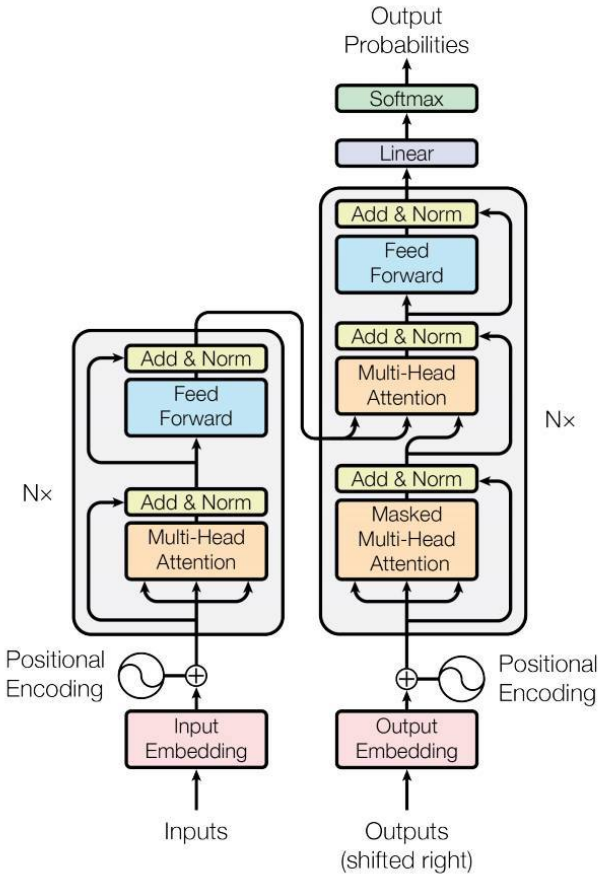


Figure 4.12: Overview of the transformer model architecture [29].

The self-attention in the transformer is based on the idea to incorporate information about the context into the representation of a token inside a sentence. The algorithm procures three vectors for a token, namely the query q, keys k, and values v. For an entire sequence, each vector is stored in matrices Q, K, and V. The attention score for each token in the sequence is then calculated as follows:

$$Attenion(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}}V) \tag{4.8}$$

where d is the dimension of the key vector and the softmax is used to get a probability distribution over the tokens in the sequence. This procedure can be further extended by applying multiple heads. The initial transformer uses eight attention heads in each self-attention layer. Each head calculates different attention scores of the input sequence by generating its own Q, K, and V matrix, where the matrices are randomly initialized to cause variation and therefore capture different aspects. The outputs from the different attention heads are concatenated and transformed through a linear transformation using a weight matrix W. This procedure is defined as follows:

$$Multihead(Q, K, V) = Concat(head_1, head_2, \ldots, head_n)W \tag{4.9}$$

Another noticeable feature of the transformer model is the positional encoding, which is applied to each input embedding, as seen in figure 4.12. Without the positional encoding, the model is not aware of the ordering regarding the input tokens, since the self-attention computes the scores independently of the order. To overcome this problem, a vector is added to each input embedding calculated by sinus and cosine functions of different frequencies. This vector determines the position of the individual token inside the input sequence.

Bidirectional Transformer (BERT)

The Bidirectional Encoder Representations from Transformers (BERT) model [59] is a language deep learning model based on the transformer. The model is pre-trained on large language corpora and afterwards fine-tuned for specific tasks like question answering or next word prediction. It uses a bidirectional training process for the pre-training of the transformer model by introducing two different NLP tasks, namely masked language modeling (MLM) and next sentence prediction (NSP). The MLM task works by hiding a word in the sentence and letting the model predict the masked word given the word's context, i.e., the rest of the input sequence (left and right part from the masked token). The objective of the NSP task is to predict if two given sentences share a logical connection or whether their relationship is random. Figure 4.13 shows the overall procedure with the pre-training on the left side and the fine-tuning on the right side. This approach reached state-of-the-art performance on 11 different NLP tasks.

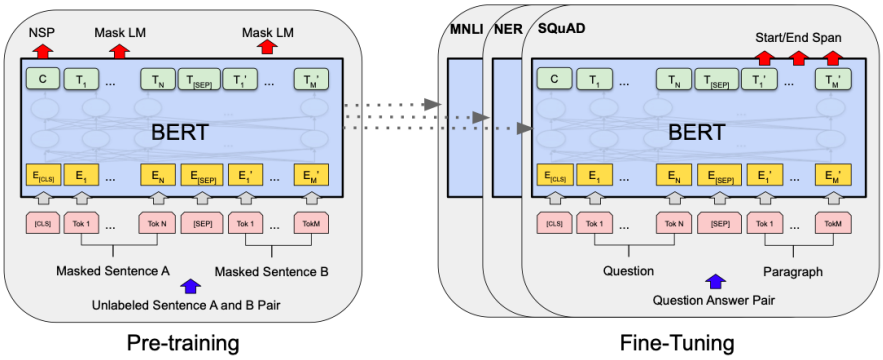


Figure 4.13: Overall procedure for pre-training and fine-tuning the BERT model. The left side shows the pretraining with the NSP and the MLM tasks. The right side shows the fine-tuning on different downstream tasks like question answering on the SQuAD [60] dataset [59].

Chapter 5

Approach

In the following, we explain our approach to solving the problem described in chapter 2. We first discuss the data pipeline and the preprocessing of the data that we use for training and evaluation. Afterwards, the developed ideas and algorithms for road segment embedding generation are presented and explained in detail.

5.1 Data Preprocessing

To train and evaluate our models we use publicly available road network data from OSM (see section 4.1) and trajectory data from different cities (e.g., Porto, San Francisco, and Hanover). As the data is in raw form, we preprocessed it to create a consistent dataset with additional features. In the following sections, the preprocessing of the OSM and trajectory datasets will be explained and illustrated.

5.1.1 OpenStreetMap

We use OSM, which provides a publicly available database with data about road networks, to obtain road network data of different cities. The required data is extracted using the OSMNX [61] framework. The tool can query geospatial data from OSM, which is further converted to a NetworkX [62] graph. The graph G, which represents a road network, is a multi-directed graph, where edges represent roads and nodes represent intersections between roads. This graph contains all structural and other context-related information (i.e., road type, speed limit, etc.) about the road segments and connections within the requested area or city. For the training of the models, we converted the road network graph to a line graph as described in the problem statement (see chapter 2).

Since the data extracted from OSM is based on voluntary contribution, information is missing for many roads, such as the speed limits, the width of the roads, or whether it is a bridge. Furthermore, the data is in raw form and therefore not normalized or encoded. As a basis for our training and evaluation, our approach aims to generate a complete dataset with proper data cleaning.

Initially, features that were mostly incomplete or did not add significant meaning were discarded. These included for example the width, the reference number, and the name of a road. Since often the encoding of categorical data can have a positive effect on the performance of a neural network [63], categorical data was converted using one-hot encoding [64], which resulted in numerical representations. Features that indicate binary behavior, such as whether a road is of a specific type, were encoded in boolean values (0 or 1). The number of lanes, the type, and the prescribed maximum speed of a road segment has been one-hot encoded because these features are categorical data that can take more than two different values.

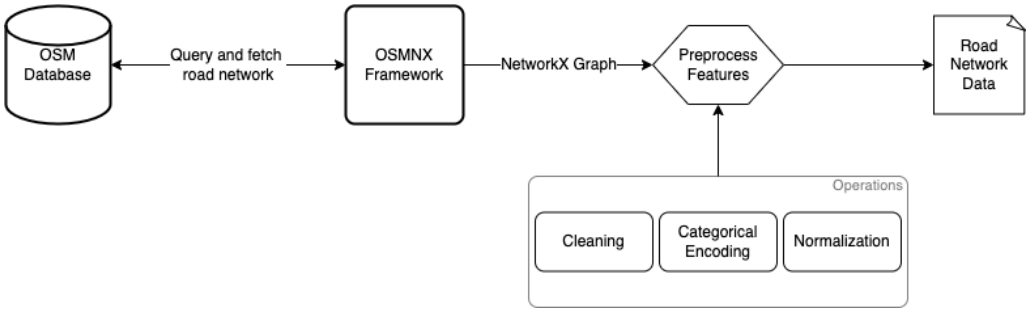


Figure 5.1: OSM road network data preprocessing pipeline, which we used to generate the network training dataset.

Some features were partially incomplete, as some road segments were not tagged. These included the number of lanes and the maximum allowed speed. To fill in the missing data, K-Nearest Neighbor (KNN) imputation [65] was applied. The most similar neighbor of a sample with a missing value in the feature space was chosen, and the feature value of the neighbor was used as fill. Further, continuous features were normalized column-wise by min-max normalization [66]. For graph-based approaches such as GCN and GAT, row-wise normalization was further performed to map the relation between features of a road segment. The process resulted in a complete dataset with features for all road segments in the considered road network area. Figure 5.1 shows our complete road network data preprocessing pipeline.

5.1.2 Trajectories

For trajectory data, we use public and private datasets of vehicle trajectories. These datasets are in raw form and contain GPS and time information about the vehicle's movement paths. The raw data had to be preprocessed to match the needs of our evaluation. Figure 5.2 shows an overview of the preprocessing procedure, which we explain comprehensively in the following.

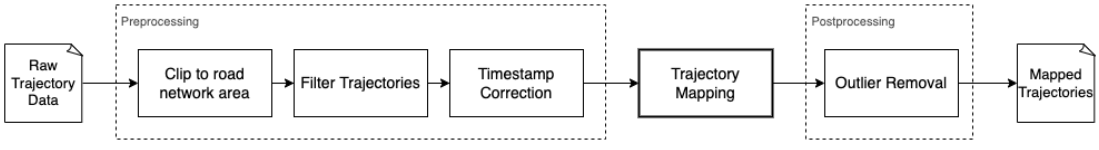
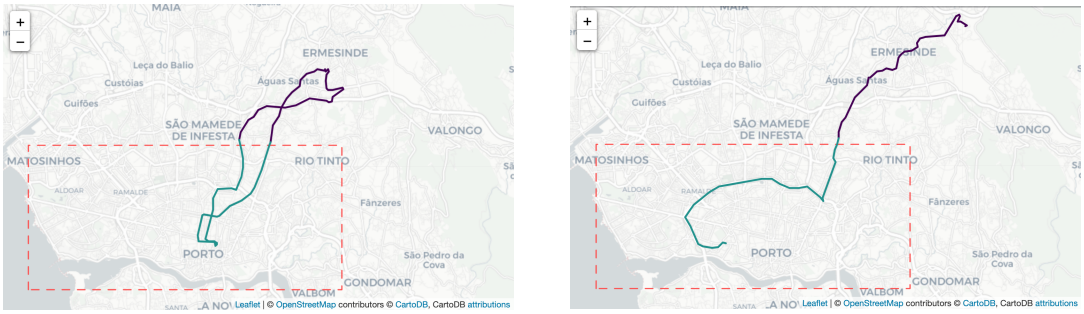


Figure 5.2: Overview of the preprocessing pipeline for the trajectory data.

Since trajectories often include observed GPS points from locations outside the associated road network, the trajectories were clipped to the spatial bounds of the corresponding network. For this purpose, we created a bounding box around the road network of the city we consider. Trajectories that lie completely outside this bounding box have been removed.

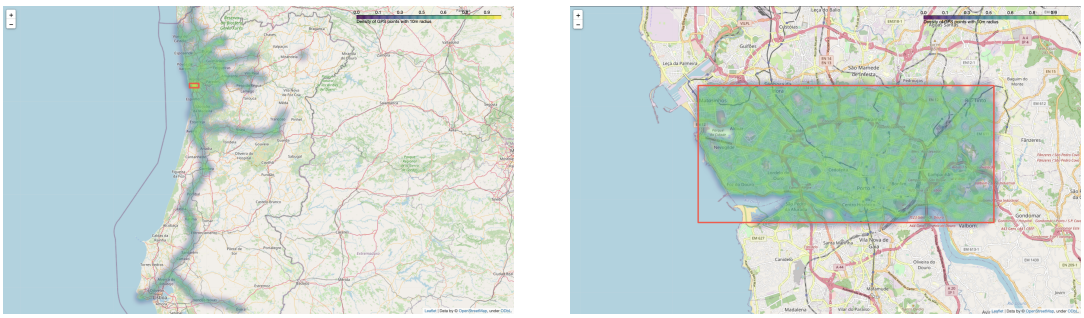


(a) Trajectory which is divided into sub-trajectories

(b) Trajectory which is truncated

Figure 5.3: Example of the trajectory splitting process. The red dotted box in (a) and (b) shows the bounds of the road network. In (a) the trajectory moves outside the bounding box and returns later. The turquoise lines are the two trajectory paths inside the box, which we keep (they are not connected inside the box). The paths are split from the original trajectory path and saved as two separate trajectories. The darker purple path is removed since it is out of the area of interest. Part (b) shows the same process for a trajectory that does not return into the bounding box. Here the trajectory gets cut into two parts, where the purple part is removed, and we only use the turquoise part.

It occurs that trajectories start inside the bounding box, move outside, and eventually return to the box. These trajectories have been split so that the trajectories used for training and evaluation only contain GPS points that are in the bounding box area. Figure 5.3 visualizes this process using an example trajectory from Porto. Subsequently, trajectories containing less than 10 sample points were removed, since they have little significance. Figure 5.4 shows how the GPS points distribute before and after the clipping process on the example of the Porto dataset (see figure C.1 and C.2 for the San Francisco and Hanover dataset GPS clippings).



(a) GPS points before clipping

(b) GPS points after clipping

Figure 5.4: Example of trajectory GPS point clipping on the Porto trajectory dataset. The heatmap describes the density of GPS points within 10 meters, where bluish areas denote fewer GPS points and yellow in proportion many (10% of the available GPS points are used for the plots). The bounds of Porto's road network are marked with a red rectangle in (a) and (b).

A fundamental problem caused by the splitting of the trajectories is the discrepancy in the timestamps. The timestamps have to be rearranged when splitting a trajectory since it is not guaranteed that there is the same time difference between recorded points. Since we first clip the trajectories, we use the following function:

$$f(C_n, C_o, T_o) : T_o \mapsto T_n \tag{5.1}$$

which searches for the corresponding timestamps T_n given a new sub trajectory $C_n \in C_o$, the original trajectory C_o and the original timestamp sequence T_o.

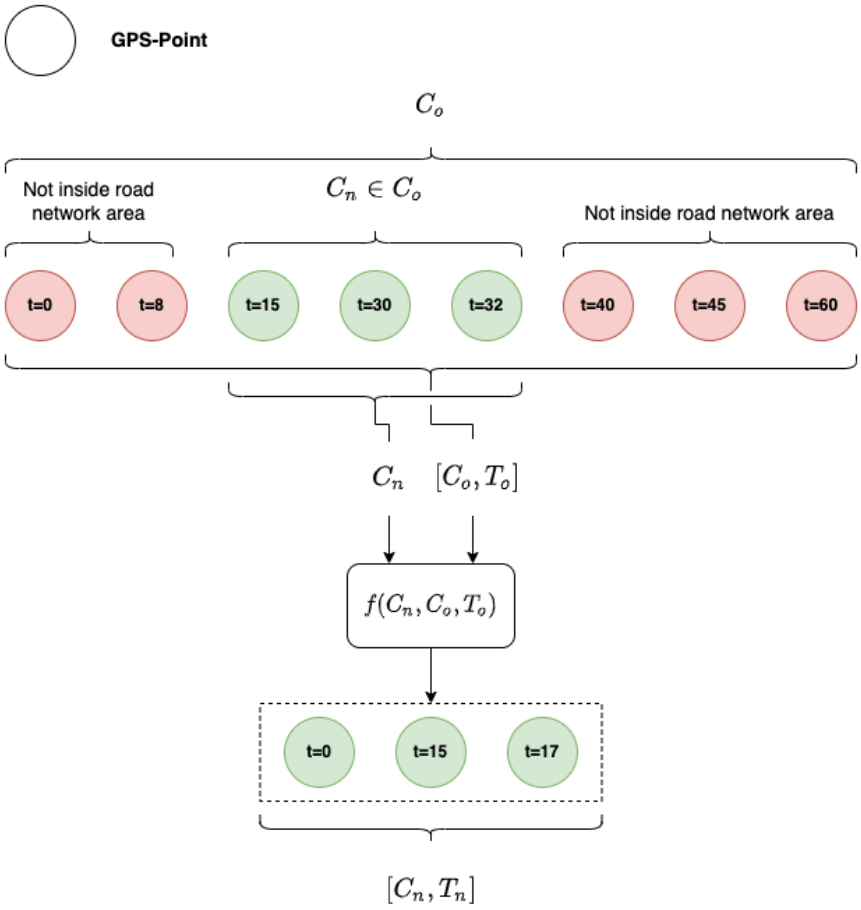


Figure 5.5: Visualization of the timestamp correction algorithm for a single trajectory.

Figure 5.5 visualizes the idea behind the timestamp correction algorithm for a single matching sub trajectory, where for each GPS point in the sub trajectory, the algorithm searches for the corresponding point in the original trajectory and extracts the matching timestamp. All timestamps are concatenated, and the chronologically oldest timestamp is subtracted, resulting in a corrected timestamp sequence T_n corresponding to the sub-trajectory C_n.

The algorithm also works for trajectories that are split into multiple sub-trajectories. Specifically, instead of a single sub-trajectory that fits into the road network area, the clipping algorithm could return two or more such trajectories from a single original trajectory. In this case, the procedure of correcting the timestamps can be done for each sub-trajectory individually.

To extract features from the trajectories, they must be mapped to the corresponding road network. For this purpose, the GPS points of a trajectory are mapped to the closest contiguous road segments. As a matching algorithm, we use Fast Map Matching (FMM) [12], which uses a Hidden Markov model [67] to predict the route of the vehicle in the road network based on the GPS points and the topological constraints. The matching result is a sequence of road segments that the vehicle has traversed, the road segment for each GPS point in the trajectory, and the shortest path distance between each tuple of consecutive GPS points. Since the time difference between GPS points is known, the average speed s between two consecutive points can be additionally calculated by FMM as follows:

$$s(x_i, x_{i+1}) = \frac{d(x_i, x_{i+1})}{t(x_i, x_{i+1})} \tag{5.2}$$

where $d(x_i, x_{i+1})$ is the shortest path distance and $t(x_i, x_{i+1})$ is the time in seconds between consecutive points x_i and x_{i+1}.

The resulting dataset is again filtered for outliers and anomalies. The average velocity of a whole trajectory may be zero. For example, if a vehicle is stationary during the entire GPS recording. Furthermore, despite enough GPS points, only a few road segments may have been traveled (for example because the vehicle stops during the recording). Trajectories with an average speed less than and equal to 0 or with less than three road segments traversed were removed due to the lack of validity.

In some cases, between two consecutive GPS points inside a trajectory, the average speed could not be calculated and was marked as infinite by FMM. Typically, this was a single value in the whole trajectory. Therefore, we used a sliding window approach to calculate the velocity from the values of the neighboring velocities. The missing velocities are calculated as follows:

$$s_m(x_i, x_{i+1}) = \begin{cases} s(x_{i+1}, x_{i+2}), & \text{if } i = 0 \text{ (i)} \\ s(x_{i-1}, x_i), & \text{if } i = n-1 \text{ (ii)} \\ \frac{s(x_i-1, x_i)+s(x_{i+1}, x_{i+2})}{2}, & \text{otherwise (iii)} \end{cases} \tag{5.3}$$

Case (i) and (ii) of equation 5.3 describe the calculation when the velocity at the beginning or end of a trajectory is missing, respectively. The predecessor or successor velocity is used to approximate the missing value. Case (iii) shows the calculation when there are predecessor and successor velocities, i.e., when the missing value is not at the border of the trajectory. This is where the average of the neighboring velocities is calculated.

The preprocessing of the trajectories results in a complete expressive dataset without numerical anomalies in terms of velocities or travel times. The processed datasets will be used later in this thesis to generate features, training models, and for various tasks in the model evaluation.

5.2 Models

Recent work indicates that the incorporation of trajectories can significantly improve the performance of models for predictions on road networks [68, 69, 70]. Trajectories reflect the behavior of the road network, which should be included in a latent representation to get a better estimation of functional and structural dependencies between road segments. Our ideas are based on actively exploiting spatio-temporal movement patterns and dependencies derived from trajectories by directly integrating this information into the learning process. In the following sections, we present our individual approaches comprehensively.

5.2.1 Graph Trajectory Convolution (GTC)

Standard graph convolutions suffer from the problem of not being able to model long-term structure dependencies. Since with each layer, the next k-hop neighborhood is aggregated, an oversmoothing of the features occurs after a few successive layers [71, 3]. Another problem for standard graph convolutions is the sparsity in terms of neighborhood degree, which is a well-known phenomenon in road networks [72, 73]. To overcome these problems and simultaneously learn real road network behavior, we propose GTC.

The GTC model is a novel approach to incorporating behavior knowledge from trajectories into graph convolutions. The main idea behind this layer is to aggregate information along trajectories, while the nodes are weighted by the probability of occurrence to counteract oversmoothing and sparsity. With this approach, the transition flows between road segments are modeled, which approximates the travel behavior in the road network.

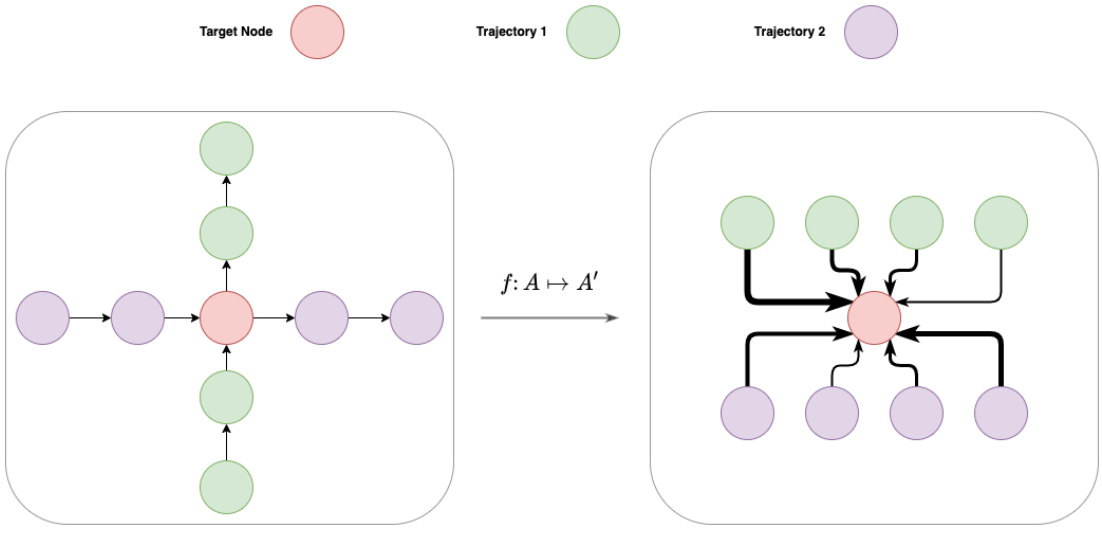


Figure 5.6: Example of the transformation process for a single node denoted in red color. The green and purple nodes are distinct trajectory paths. The left box displays the original graph structure with equally weighted edges. The right box shows the transformed graph structure for $k = 2$ and the bidirectional window setting. The arrow direction denotes the aggregation direction, whereas the larger edge width denotes higher weight (transition probability in the reverse direction of the arrow).

Mathematically, we define a function (see equation 5.4), which transforms the adjacency matrix A into a hidden Markov transition matrix A' given the trajectory sequences D. The transition matrix A' can be derived from the trajectory sequences D, which are modeled as a k-order Markov process [74, 75]. By aggregating the Markov processes, we can estimate A'.

$$f(A, D) : A \mapsto A' \tag{5.4}$$

Equation 5.5 shows this aggregation for a single entry between nodes i and j in A', where $d(D, i, j, b, k)$ is a function that counts the sub-trajectories $T \in D$ that go through node i and j, where node j is no farther than k nodes from node i in any direction on the trajectory. This can also be seen as a bidirectional (denoted by b) window S of size $k * 2$ starting from node i. Specifically, the window S is expanded in both directions (forward and backward) starting from a node i along the trajectory T. The length of the window in the backward direction is $min(k, x)$ and in the forward direction $min(k, |T|)$, where x is the index of node i on the trajectory. The count of matching sub-trajectories is normalized by the adjacency row summation (N in the denominator summation denotes the number of nodes in A) to get a transition probability distribution for each node. The connections from the original adjacency matrix A are furthermore included to make sure that in case of missing trajectory data, the new graph contains the same connections as the original one.

$$A'_{ij} = \frac{d(D, i, j, b, k) + A_{ij}}{\sum_{n=0}^{N}(d(D, i, n, b, k) + A_{in})} \tag{5.5}$$

Figure 5.6 visualizes this aggregation step for a single example node with hyperparameter $k = 2$ and bidirectional setting. As shown in the figure, the aggregation models long-term dependencies depending on trajectories and the parameter k, which makes the graph denser without risking oversmoothing because nodes that occur less frequently in trajectories through the target node are given a lower weight and those that occur more frequently are given a higher weight.

In addition to the bidirectional aggregation variant, we propose a forward-only method in which the window S is only considered in the direction of trajectory movement and not in both directions as in the bidirectional variant. This could lead to a better estimation of the spatial dependencies in the network regarding trajectories since in the bidirectional variant a synthetic backward movement is modeled. The difference between the two variants is visualized in figure 5.7. A comprehensive comparison between different k-values and the proposed variants can be found in section 6.5.

$$h_v^k = W_k \sum_{u \in N(v) \cup u} \frac{h_u^{k-1} A'_{uv}}{\sqrt{|N(u)||N(v)|}} \tag{5.6}$$

After creating the trajectory transition matrix A', we apply a standard graph convolution on the graph, where each edge is weighted by the trajectory flow probability (see equation 5.6). This is defined by the value in the transition matrix A'_{uv} between nodes u and v. The convolution can be seen as a weighted version of the Simple Graph Convolution (SGC) [76] with layer count parameter $K = 1$.

$$A'' = \sigma(ZZ^T) \tag{5.7}$$

We use the reconstruction loss (see section 4.4.3) between the reconstructed adjacency matrix A'' (see equation 5.7) and the original adjacency matrix A as a loss function.

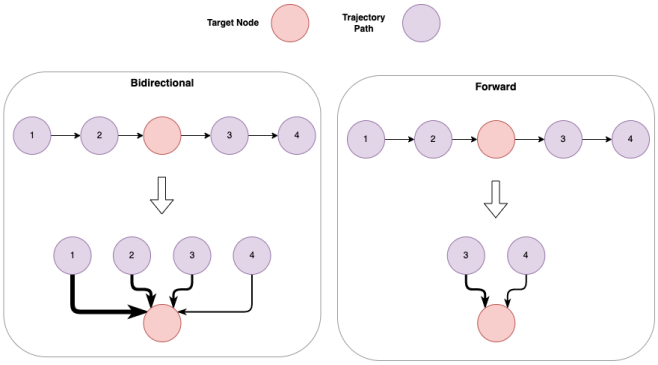


Figure 5.7: Visualization of the difference between the bidirectional and forward trajectory aggregation. The red node denotes the target node, and the purple nodes are nodes in a trajectory sequence including the target node. In the bidirectional case, connections in both directions are added in the new representation. The forward aggregation only considers nodes in the movement direction of the trajectory starting from the target node.

5.2.2 Trajectory Smoothed DeepWalk (TSD)

The Trajectory Smoothed DeepWalk (TSD) is an extension of the standard DeepWalk algorithm [21], which incorporates information from trajectories into the sampling process of the random walks. In contrast to a normal DeepWalk, this approach learns road network behavior induced by trajectories in combination with structural properties. Hence, the learned embedding should work better for tasks that are based on road network behavior like travel time or destination prediction. Similar to the GTC model (see section 5.2.1) a trajectory transition matrix is created. Since we want to model the direct flow transitions in the original graph, we set $k = 1$ and use the forward method to create the matrix. In addition, the original connections are included. Self-loops are removed, except for nodes that have no outgoing connections, so-called dead ends. The trajectory transition adjacency matrix is used to sample the walks through the graph. With the transition probabilities, the transition flows are modeled and nodes that are visited more frequently in the trajectories are more likely to be visited in the random walks. The architecture and training are the same as in DeepWalk or Node2Vec (see section 4.2 and 4.3 for a comprehensive explanation). Figure 5.8 shows an overview of the training process.

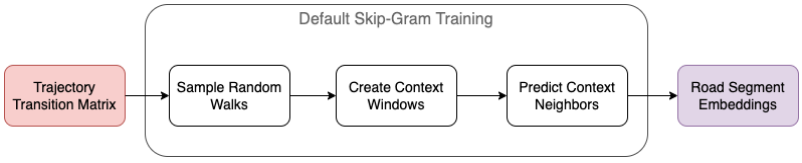


Figure 5.8: Overview of the trajectory smoothed DeepWalk module. The red box shows the modified part in contrast to a normal DeepWalk model.

5.2.3 Graph Trajectory Network (GTN)

The main intuition behind the GTN is to combine the previously proposed models, namely GTC and TSD into a single encoder model. By combining both representations, we aim to create a more robust embedding. To further fine-tune the embedding capturing structural and behavior information inducted by trajectories, we propose to use a trajectory-enhanced transformer-based encoder module inspired by the Toast and BERT model [4, 59].

As seen in figure 5.9 the pre-trained embeddings from the GTC and TSD models are concatenated and used as input embeddings for the transformer module. Furthermore, to preserve the sequential information on the trajectories a learnable positional encoding is injected into the input embeddings as described in section 4.5. Without the positional encoding, the self-attention mechanism would treat the trajectory as an unordered set. The initial input layer is followed by a multi-head self-attention mechanism [29], which maps the representations of a trajectory from the input layer to a transformed representation Z (see section 4.5 for details). The output Z gets processed by a feed-forward network. This sequence of sub-layers (multi-head self-attention and feed-forward layers) can be stacked to extract more complex patterns from the trajectories. Residual connections [58] and layer normalization [77] are applied between the layers to counteract overfitting due to the increasing depth of the network [29, 4].

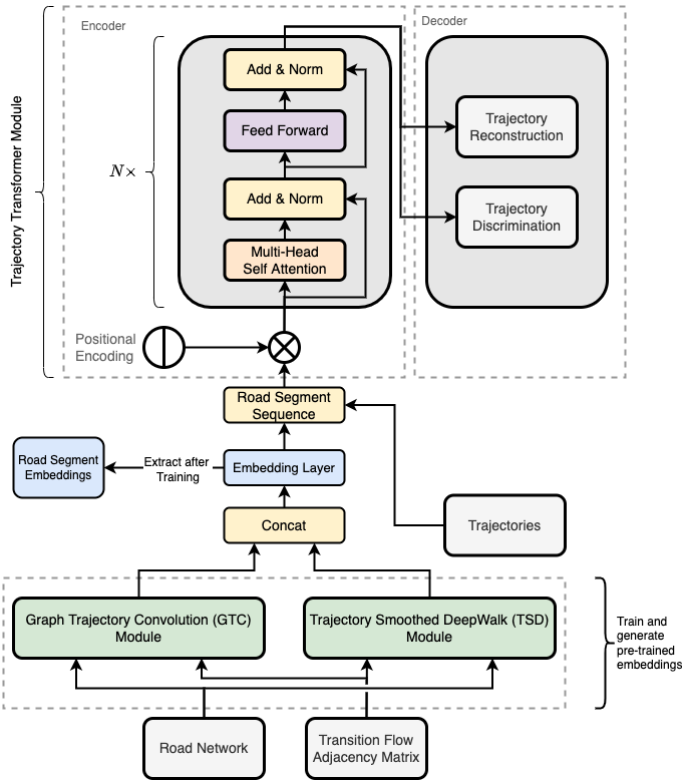


Figure 5.9: Architecture overview of the proposed GTN model.

The learning targets for the transformer in the training are the same as proposed for the Toast model, namely trajectory reconstruction and trajectory discrimination. These tasks are inspired by the original tasks (MLM and NSP) used in the self-supervised learning process of the BERT model (see section 4.5 for details). In the trajectory reconstruction task, a consecutive sequence of road segments is masked out from the trajectory. The

target for the model is to predict the masked-out road segments given the rest of the trajectory. In our model, we randomly mask out 25% of consecutive road segments in the trajectory. The loss function is the cross entropy loss between the predicted road segments and the masked ones. The trajectory discrimination task is applied to distinguish between real trajectories and random sampled walks through the network. The authors of Toast argue that this task motivates the model to further learn transition patterns and an overall understanding of travel semantics on the road network [4]. The model is trained by sampling either a random walk or a real trajectory from a dataset with equal probability, and afterwards letting the model predict if it is a real trajectory. The loss function for this task is the cross entropy loss between the prediction of the trajectory type (fake or real) and the actual type.

Different from the originally proposed model, which used the learned route representations, our goal is to learn robust static embeddings for the road segments. The transformer module generates contextual embeddings (route representations), which are based on the sequences given as input to the model. Specifically, when doing a forward pass through the transformer, each generated embedding is based on the road segments in the current trajectory and is therefore not robust to other constellations. Hence, after the training of the model, we extract the static input embeddings from the transformer's embedding layer. These embeddings are pre-initialized before training as described above and refined in the training process so that they capture additional transition patterns and semantic properties of the road network.

5.2.4 Temporal Graph Trajectory Convolution (T-GTC)

An essential component of road networks is the temporal layer because the behavior of the network is strongly based on structural as well as temporal properties. Dynamic traffic information, like the utilization of road segments, is dynamic, complex, and non-linear as they depend on various factors such as time of day, events, or day of the week. Recent work indicates that modeling the temporal layer in addition to the spatial layer has a positive impact on various ITS challenges such as traffic forecasting or travel time prediction [78, 79, 80, 32].

Our previous proposed model (see section 5.2.3) only considers the spatial and transition dependencies and learns static embeddings without considering the dynamic change over time. To capture the spatial as well as the temporal dependencies of a road network, we propose a novel unsupervised neural network architecture, namely T-GTC, which combines the GTC model with a LSTM model. The GTC models the spatial and transition dependence regarding real trajectories, while the LSTM captures the temporal dependence regarding dynamic traffic information. As an extension to the base variant, we further propose a second attention-enhanced variant of the T-GTC model, which incorporates global temporal dynamics into the representation.

We model the temporal component in the road network as a sequence of graphs denoted as S, where dynamic properties of the graph such as the average travel speed may change over the steps in the sequence. Accordingly, the spatial structure of the graph remains the same over the sequence. Figure 5.10 visualizes the described modeling.

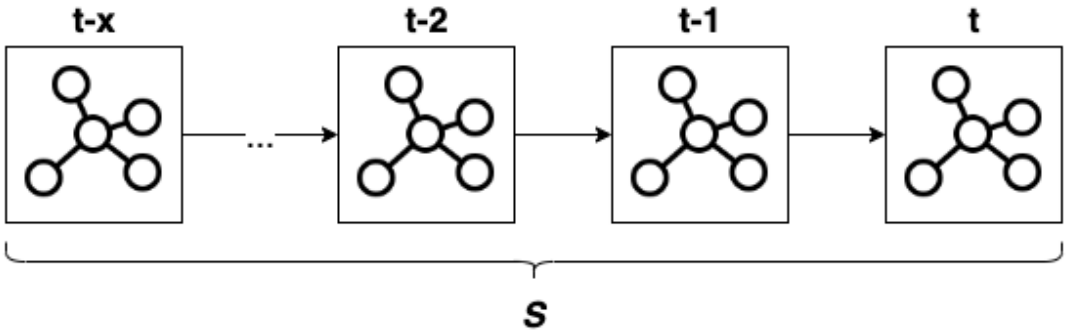


Figure 5.10: Visualization of a sequence containing graphs. Additionally, to static features, each of the graphs holds dynamic properties that can change over time, like utilization or driving speed.

Each graph $G \in S$ in a sequence is first embedded by a single graph trajectory convolution, resulting in a vector representation h_t^g (see equation 5.8). The convolution uses static features like road type or speed limit and dynamic features like measured driving speed on the road segments, which are stored in a matrix denoted by X. For a single road segment, v connected with a segment u, the feature vector X_u is multiplied with the value A'_{uv} in the calculated transition flow adjacency matrix. The result is normalized over the number of neighbors (see section 5.2.1 for further details on the trajectory convolution).

$$h_t^g = W_g \sum_{u \in N(v) \cup v} \frac{X_u^t A'_{uv}}{\sqrt{|N(u)||N(v)|}} \tag{5.8}$$

The resulting representation then serves as input to a LSTM cell. The cell consists of a forget gate f_t, input gate i_t and output gate o_t [81]. The gates are calculated by multiplying an unshared weight matrix with the concatenation of the last hidden state h_{t-1}^{gl} and the current input h_t^g. A bias term b is added additionally (see equation 5.9-5.11).

$$f_t = \sigma(W_f[h_{t-1}^{gl}, h_t^g] + b_f) \tag{5.9}$$

$$i_t = \sigma(W_i[h_{t-1}^{gl}, h_t^g] + b_i) \tag{5.10}$$

$$o_t = \sigma(W_o[h_{t-1}^{gl}, h_t^g] + b_o) \tag{5.11}$$

The new cell state C_t is calculated by multiplying the old cell state C_{t-1} with the forget gate f_t and adding the new candidates vector C'_t, scaled by i_t. Finally, the output hidden state h_t^{gl} can be calculated by deciding which information of the cell state is further propagated (see equation 5.12-5.14). Normally, the first hidden and cell state are initialized randomly or with zeros. However, we propose to initialize the first hidden state with a pretrained TSD embedding to directly induce structural properties in the initial state. The first cell state is initialized with zeros.

$$C'_t = \tanh(W_C * [h^{gl}_{t-1}, h^g_t] + b_C) \tag{5.12}$$

$$C_t = f_t * C_{t-1} + i_t * C'_t \tag{5.13}$$

$$h^{gl}_t = o_t * \tanh(C_t) \tag{5.14}$$

The hidden and cell state of a single LSTM cell are furthermore propagated to the next LSTM cell and combined with the representation of the next graph G in the sequence S [78]. The resulting last hidden state h^{gl}_t encapsulates the representation of the graph over the sequence, and thus contains both spatial and temporal dependencies. In the base variant of the T-GTC model, we propagate this representation to the decoder part of the model.

As an extension, we propose to incorporate a multi-head self-attention mechanism into the model. The self-attention mechanism is used to learn the global traffic variation information [82]. The layer takes as input the hidden states H^{gl} of the LSTM layer and returns a weighted representation, which is calculated as follows [29]:

$$Attention(H) = softmax(\frac{(HW^Q)(HW^K)^T}{\sqrt{d}})(HW^V) \tag{5.15}$$

The input representation H^{gl} can be projected into multiple subspaces, where each projection indicates an attention head. The heads are concatenated and projected again, which results in the output representation H^a (see section 4.5 for further details on multi-head self-attention).

Afterwards, the weighted hidden states H^a are propagated through a standard LSTM layer. The corresponding last hidden state h^l_t, representing the embedding of the sequence, is extracted. We further apply a residual connection [58] by adding h^l_t to the last hidden state h^{gl}_t of the first graph LSTM layer. This is followed by a batch normalization [83], resulting in the final encoder representation Z. Specifically, Z is defined as:

$$Z = Batchnorm(h^{gl}_t + h^l_t) \tag{5.16}$$

Regarding the decoder part of the model, it is firstly important to note, that the model should learn a robust representation that can be used for different downstream tasks. Therefore, as with most of our previously proposed models, the learning process is unsupervised.

The goal of the model is to learn both temporal and static properties of a road network, therefore the decoder part of the model is divided into two fundamental parts. The first part is the reconstruction of the road network from the hidden state Z generated by the encoder part. As in the GTC model training, we reconstruct the adjacency matrix A by taking the inner product between Z and Z^T followed by a sigmoid function $\sigma(\cdot)$ (see equation 5.7). The loss of this part is the reconstruction loss between the reconstructed adjacency matrix A'' and the original unweighted adjacency matrix A. This part is responsible for pushing the learning process to capture the structural properties of the road network.

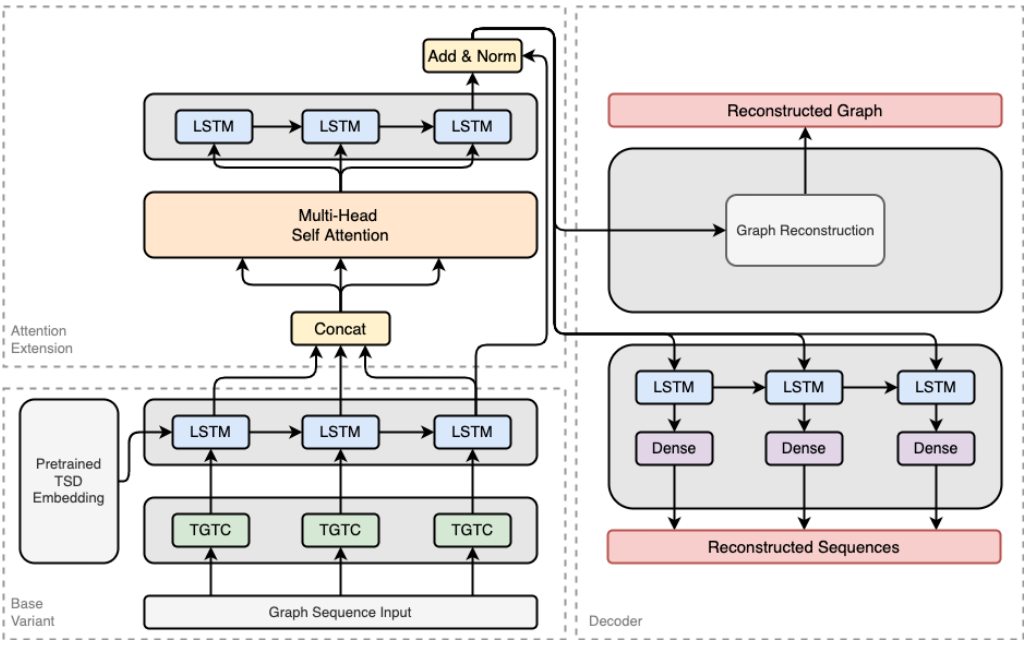


Figure 5.11: Architectural overview of the proposed T-GTC model and the attention extension.

The other part of the decoder focuses on learning the temporal dependencies of the road network by reconstructing the time series of a dynamic feature (for example, the average driving speed) for each road segment in the road network. To decode the encoder hidden state Z to the road segment's time series, we utilize a decoder LSTM model, which takes at each time step t the hidden state Z and outputs another hidden state Z'_t for each time step. Each resulting hidden state $Z'_{t-n:t}$ is further propagated through a linear layer, which outputs a continuous value for each road segment in the graph. The concatenation of the values for each time step creates a time series for each road segment, which is compared with the corresponding original time series. We use the MSE loss between the reconstructed time series for all road segments against the original ones. Figure 5.11 shows a comprehensive overview of the models architecture.

Chapter 6

Evaluation

The evaluation is divided into seven main parts. We start with an explanation of the experimental setup regarding datasets, tasks, baselines, hyperparameters, and metrics. The second part investigates the considered datasets in detail with a comprehensive data analysis. The analysis is followed by the performance evaluation of our proposed GTN model against the competitor baselines. The fourth part examines the performance of individual components of our model in the form of an ablation study. The ablation study is followed by a parameter study regarding parameters of the GTC model and an analysis of the impact on performance when incorporating trajectory-based features. The final part deals with the analysis of the generated GTN embeddings.

6.1 Experimental Setup

This section describes the evaluation setup for our experiments. First, the datasets used, and their characteristics are introduced. We distinguish between road network data and trajectory datasets for the considered cities. Afterwards, we declare the downstream tasks on which the generated embeddings are evaluated.

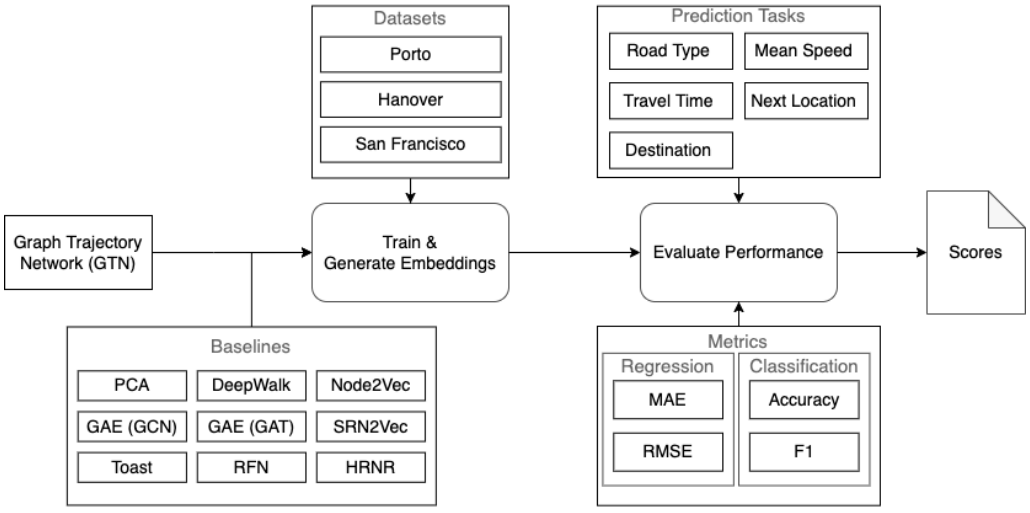


Figure 6.1: Overview of the evaluation process.

Additionally, the baselines which we compare against our model are explained. Finally, we introduce the hyperparameter setup for our models and the evaluation metrics used for the different tasks. Figure 6.1 shows an overview of the evaluation process and its components.

6.1.1 Datasets

In our evaluation, we use road networks and trajectories of three different cities, namely Porto, San Francisco, and Hanover. As described in section 5.1.1 the road networks are extracted from OSM and furthermore preprocessed through our proposed pipeline. The trajectories are obtained from different datasets, with Porto (Portugal) [84] and San Francisco (USA) [85] coming from a public dataset and Hanover (Germany) coming from a private dataset provided by the L3S research center [86]. The Porto dataset includes trajectories from 442 cabs collected over the timeframe of a complete year (from 01/07/2013 to 30/06/2014). The San Francisco dataset includes trajectories from 500 cabs collected over the timeframe of one month (05/17/2008 to 10/06/2008). The Hanover dataset includes trajectories from various vehicle navigation systems over the timeframe of around four months (15/08/2019 to 01/12/2019). The three trajectory datasets are initially converted into a uniform format (see appendix C) and subsequently preprocessed as described in section 5.1.2. Table 6.1 depicts the basic statistics of the datasets, where *#Intersections* and *#Road-Segments* show the count of respective road network elements. Average degree describes the node degree averaged over the whole road network, i.e., how many road segments are connected to an individual road segment on average. The *#GPS-Points* column shows the count of the GPS points before preprocessing and *#Trajectories* the resulting count of trajectories after preprocessing and mapping, i.e., the count of trajectories that are used for training and evaluation. Coverage shows the proportion of how many road segments of the road network are covered through the trajectories, i.e., how many road segments are at least visited once. A more comprehensive explanatory analysis is shown in section 6.2. The trajectory data is split into a training set (70% of the data) for training the GTN model and an evaluation dataset (30% of the data) for the task-dependent decoder model. The evaluation dataset is further separated into a training (80%) and a test dataset (20%) for the decoder model.

	Road Network			Trajectory		
	#Intersections	*#Road-Segments*	Average Degree	*#GPS-Points*	*#Trajectories*	Coverage (%)
Porto	5,358	11,331	2.3490	74,269,739	1,544,234	0.979
San Francisco	9,739	27,039	3.0980	11,219,955	406,456	0.944
Hanover	7,674	18,799	2.7990	17,851,818	50,411	0.838

Table 6.1: Fundamental statistics of the datasets.

6.1.2 Tasks

This section describes road network-based classification and regression downstream tasks, on which we evaluate our approach against common baselines and several SOTA methods from the literature. The tasks are ordered by difficulty and are adopted from recent work regarding representation learning for road networks [4, 5, 6]. These tasks can be divided

into two main categories. Specifically, we evaluate on tasks that test the capability of learning static properties of road segments like road label or mean driving speed prediction and tasks that test the knowledge of structural dynamics based on trajectories. This includes travel time estimation, next location, and destination prediction.

In the following, we comprehensively describe the individual tasks, starting with simple tasks and increasing the difficulty in terms of complexity with each task. Furthermore, we show the architecture and hyperparameter setup of the tasks underlying decoder models.

Road Label Classification

For the road label classification task, the decoder model is trained to predict the road types of road segments based on the individual embeddings generated by the evaluated encoder models. The dataset contains different road types, which we use as labels (see section C.5 for a description of types and section 6.2 for detailed analyses of the distribution inside the datasets). In the evaluation stage, the learned road segment embeddings are used as input to a simple Logistic Regression classifier [87, 64], which acts as the decoder model and predicts the corresponding road label for each road segment in the road network. We use the standard hyperparameters proposed in [64], which are shown in table 6.2. With this task, we can determine how well our approach captures spatial and road segment-based properties of the road network in contrast to the competitor baselines.

Logistic Regression Classifier			
penality	*C*	*solver*	*max_iter*
l2	1.0	lbfgs	1000

Table 6.2: Hyperparameter setup for the Logistic Regression model used in the road label classification downstream task.

Mean Speed Prediction

Mean Speed Prediction is a regression task, where the decoder model predicts the mean driving speed on a road segment given the corresponding embedding without considering any temporal dynamics. As a decoder model, we use a single-layer feed-forward network [88, 64] with 1024 neurons. The full hyperparameter setup for this task can be seen in table 6.3. An advantage of this model is that it has no advanced tuneable hyperparameters [64] in contrast to models, such as tree-based regressors [89] and is therefore stable between embeddings from different encoder models because the performance does not depend on proper tuning. Furthermore, since the goal is to evaluate the performance difference between the embeddings, simpler models are appropriate.

Single-Layer Perceptron Regressor					
hidden_layer_size	*activation*	*optimizer*	*batch_size*	*learning_rate*	*max_iter*
1024	ReLU	Adam	200	0.001	30

Table 6.3: Hyperparameter setup for the neural network used in the mean speed prediction downstream task.

Travel Time Estimation

Travel time estimation [90, 91] is a regression task, where the model predicts the travel time of a trajectory given the corresponding sequence of road segment embeddings that form a path in the road network. As a decoder model, we use a LSTM model [81], which takes the embedding sequence as input followed by a three-layer dense network with ReLU activations [48] between each layer. The dense layer in the middle has twice the hidden size as the input hidden size. The decoder yields the travel time prediction as a single value. Figure 6.2 shows the architecture of the LSTM model. We use the mapped trajectories from the Porto, San Francisco, and Hanover trajectory datasets (see section 6.1.1) as training data and the corresponding calculated travel times as ground truth for the training. The hyperparameter setup is shown in table 6.4. This task is especially useful to determine how well the learned representations of the road segments reflect the spatial and traveling semantics of the road network.

Trajectory LSTM Decoder Model					
hidden_layer_size	*lstm_layers*	*activation*	*optimizer*	*batch_size*	*learning_rate*
128	2	ReLU	Adam	512	0.001

Table 6.4: Hyperparameter setup for the LSTM decoder model used in the travel time, next location, and destination prediction downstream tasks.

Next Location Prediction

In the next location task [92], the goal is to predict the next visited road segment based on a past trajectory. We use the last element of a trajectory as next location ground truth and the predecessors as historical trajectory. A historical trajectory consisting of road segment ids is converted into a sequence of corresponding embeddings and put as input into the LSTM network described in figure 6.2. The LSTM network is followed by a three-layer dense network with ReLU activation functions between each layer. Different from the model used in travel time prediction, the prediction target changes to a multi-class classification problem. Specifically, the goal is to predict the next road segment on the trajectory, which possibly could be every road segment inside the road network. Therefore, the output dimension of the dense layer is not a single value but a logit for each road segment. After the output layer of the dense layer, we apply a custom masked log softmax function. To speed up learning, the possible road segments for prediction are restricted to the neighbors of the second-last road segment of a trajectory [92, 93] when training the model (inference works without restriction). For this purpose, the softmax input is masked so that the values for road segments outside the neighborhood of the last road segment are close to zero. Equation 6.1 shows the calculation of this mask. We take $log(0 + a)$ if the road segment is a neighboring node, where a is a minimal value near zero ($1e - 44$ in our experiments). The resulting value is significantly divergent from 0. If the road segment is a node outside the neighborhood of the target node, the mask value should be near zero, which is realized by taking $log(1 + a)$ as value for the mask.

$$M_{i,j} = \begin{cases} \log(0+a), & \text{if } r_i \text{ can reach } r_j \\ \log(1+a), & \text{otherwise} \end{cases} \tag{6.1}$$

Equation 6.2 shows the state-constrained *log-softmax* function mathematically. We use negative log-likelihood as a loss function, which gets the output of the above described *log-softmax* as input.

$$\text{MaskedLogSoftmax}(x_i) = \log\left(\frac{\exp(x_i + M_i)}{\sum_j \exp(x_j + M_j)}\right) \tag{6.2}$$

The overall hyperparameter setup is the same as for the travel time estimation task and can be seen in table 6.4. The model uses the cross entropy loss function to optimize the classification of the next visited road segments.

Destination Prediction

The destination prediction task [94, 95] is relatively similar to the next location prediction task, but more complex in terms of the prediction goal. This task removes k percent from the end of the trajectory. The remaining $1-k$ percent of the trajectory are used as input into the LSTM model visualized in figure 6.2. Accordingly, the last road segment of the trajectory is the target to be predicted. In our experiments we set k to 0.3, i.e., we remove 30% of the trajectory and predict the destination road segment given the first 70% of a trajectory. The architecture of the LSTM model is the same as for the next location prediction task, except for the masked softmax function. Since the prediction area cannot be restricted, we use a standard softmax for this task. The output of the dense layer is fed into the softmax function to extract the road segment with the highest predicted probability. The hyperparameter setup and loss function are the same as described in the next location prediction task section (see table 6.4 for hyperparameter setup).

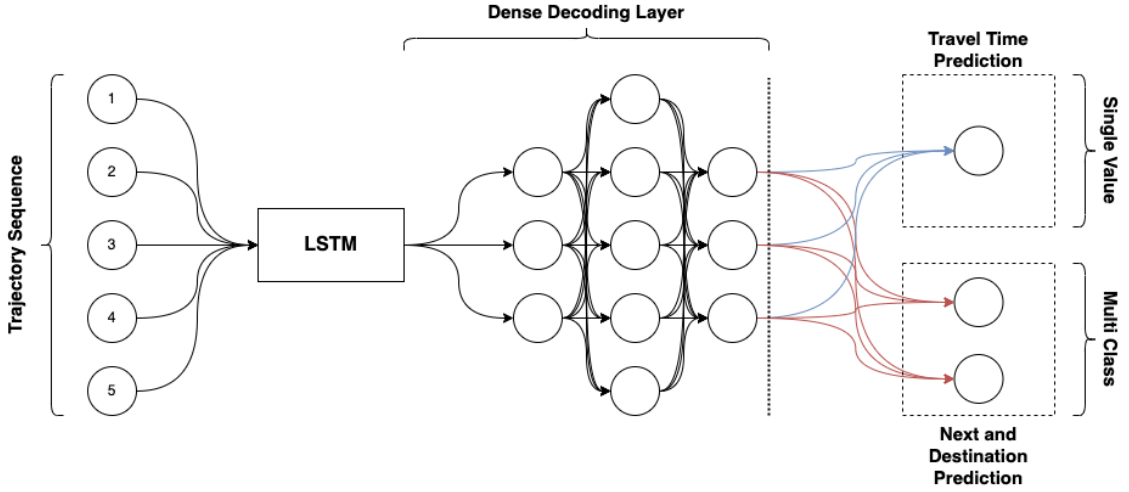


Figure 6.2: Visualization of the LSTM decoder model architecture used in the trajectory-based downstream tasks. The part before the dashed line is the same for all tasks. Architecturally, only the prediction target changes between regression and multi-class prediction. The input to the LSTM part is a trajectory in the form of a sequence consisting of corresponding embedded road segments generated by the evaluated encoder model.

6.1.3 Baselines

To compare our proposed approaches, we choose several baselines from the field of machine learning for graphs and especially from the area of representation learning for road networks. We start explaining simpler concepts and move to SOTA techniques with a focus on road networks from the literature. We compare our approach against methods from three different categories, including feature, structure, and mixed feature-structure-based methods. For the road label and mean speed prediction task, all models are evaluated using a five-fold cross validation [96]. Due to high training times, the models are evaluated without cross validation on the trajectory-based tasks. The embedding dimension is set to 128 for every model in the evaluation. In the following, the baselines and their hyperparameter setup used in our evaluation are briefly described.

Principal Component Analysis (PCA)

Principle Component Analysis (PCA) [97] is used to reduce the feature dimensionality of large datasets, while preserving most of the information. It trades dimensionality against accuracy. In our case, we use the PCA to transform the features of road segments into transformed representations containing the $k = 4$ most important components. This method does not consider any structural information besides the raw features.

DeepWalk

DeepWalk [21] uses unbiased random walks to sample node sequences from the network. Afterwards, skip-gram is applied to the node sequences, which predicts the context of a node given the neighbors in the node sequence (see section 4.2 for a more detailed explanation). This method learns structural information but does not consider any road segment-specific features, such as road types or speed limits. The walk length, the window size, the number of walks, and the number of negative samples per node are set to be 30, 5, 25, and 7, respectively.

Node2Vec

Node2Vec [19] is the generalization of DeepWalk, which applies a biased random walk to learn local and global structural dependencies (see section 4.3 for a comprehensive explanation). The walk length, the window size, the number of walks, and the number of negative samples per node are set to be the same as in the DeepWalk model, i.e. 30, 5, 25, and 7 respectively. Furthermore, the model-specific hyperparameters p and q are set to 1, and 4, respectively [4].

Graph Auto Encoder

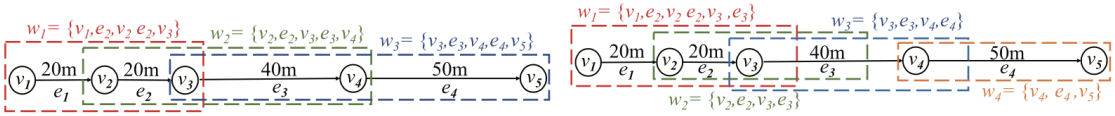
The GAE [50] is a convolution method that learns the graph structure by aggregating information from neighboring nodes and optimizing the reconstruction of the adjacency matrix (see section 4.4.3 for comprehensive explanation). We use two different variants, each with a different encoder model.

The first variant uses a normal GCN model (see section 4.4.1) as encoder and the second one a GAT model (see section 4.4.2). The layer count is set to two in each variant, which generally shows the best performance [26].

SRN2Vec

SRN2Vec [5] is a module that is specifically designed to learn representations for road segments. It learns road segment representations by considering geo-locality and feature properties of road segments.

The module samples shortest paths using the Dijkstra algorithm [98] between random nodes in the graph. Afterwards, the generated random walk sequences are transformed into geo-distance-based groups using sliding windows with width m meters. Figure 6.3 shows the difference between a hop-based window and the proposed distance-based window. For each pair of edges (e_x, e_y) within a window, a positive sample $\langle e_x, e_y, 1, same_{type}(e_x, e_y)\rangle$ and n negative samples [40] $\langle e_x, e_r, 0, same_{type}(e_x, e_r)\rangle$, where the second edge is replaced with another random edge, are created. The third entry in the sample determines the positive geo-locality between the pair of edges, i.e., if the edges are located within a m-meter window in the road network. The $same_{type}$ entry indicates if the edges share the same road type label. In the training process, SRN2Vec takes the pairs of road segments e_x, e_y as input and predicts the geo-locality and same road type relationship between them. We use the same hyperparameters as proposed in the original paper [5], except for the output dimensionality. The dimensionality, sliding window range, number of negative samples per positive sample, and number of shortest paths sampled per road segment are set to 128, 900 meters, 7, and 1280, respectively. We implemented the algorithm from scratch according to the descriptions inside the corresponding research paper.



(a) Hop-based Neighborhood: 2-hop (b) Distance-based Neighborhood: 50m

Figure 6.3: Difference between a hop-based window (a) and a distance-based window (b) regarding the included road segments inside the different windows [5].

Toast

Toast [4] is a framework that uses a context-aware skip-gram model to generate road segment representations. Furthermore, the framework applies a trajectory-enhanced transformer module to generate route representations. Since we are only interested in road segment representations, we discard the transformer module and only use the proposed traffic context-aware skip-gram module as a baseline. An overview of the complete framework can be seen in figure 6.4, where the left box describes the traffic context-aware skip-gram module and the right box describes the trajectory-enhanced transformer module, which is inspired by the BERT model [99].

Specifically, the proposed skip-gram module aims to incorporate traffic patterns into the representations by extending the default skip-gram model (see section 4.2) with an auxiliary traffic context prediction task. Given a target node and its context neighbors, Toast predicts the traffic context (i.e., the road type label or speed limit) and the context neighbors. Therefore, the road segment embeddings are optimized for traffic and neighborhood context predictions.

We use the road type label as traffic context in our setting and the same hyperparameter setting as in the original implementation, i.e., 20 walks per node, context window of size 5, and 10 negative samples per positive sample. The walk length is randomly sampled between 5 and 100 nodes per walk. For this model, the authors provided the original code base of their model, which we replicated and modified to work with our datasets.

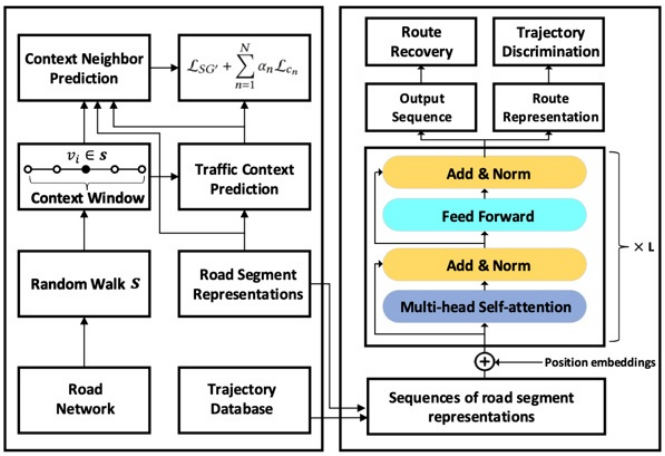
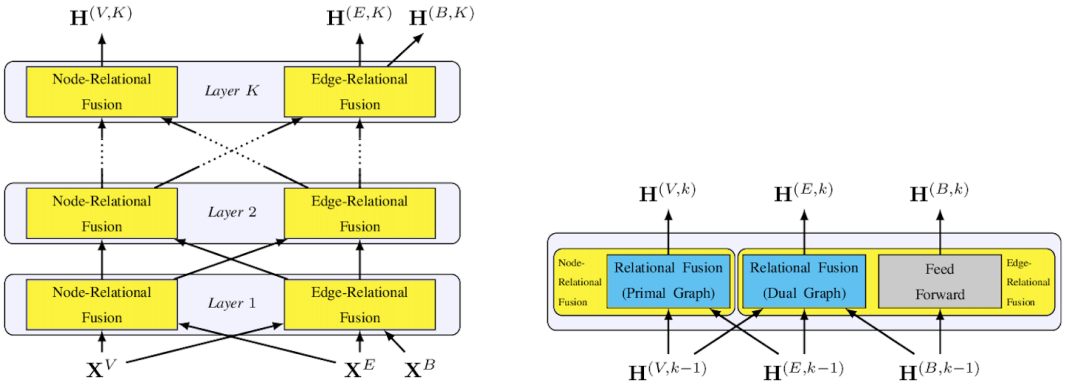


Figure 6.4: Overview of the Toast framework. We only use the enhanced skip-gram module (left part of the image) as baseline [4].

Relational Fusion Network

The Relational Fusion Network (RFN) [7] is a GCN architecture designed for machine learning tasks on road networks. The method learns representations based on node and edge-relational views, where nodes refer to intersections and edges refer to road segments. For the node-relational view, a primal graph structure [100] is used, where nodes represent intersections and edges represent relationships between the intersections given by the original road network. For the edge-relational view, a dual graph structure [100] is used, where nodes represent edges in the original road network and edges represent relationships, such as the angle between two road segments, denoted as between edges. Figure 6.5 shows an overview of the RFN architecture, where on each fusion layer a fusion is performed on the node and edge view which uses the primal and dual graph representation, respectively, generating a representation for intersections and road segments in the road network. The proposed fusion operator combines the representations of a source and target node along with the representation describing their relation and outputs a relational representation. The resulting representation is further aggregated and normalized. In the figure, the matrices X^V, X^E, and X^B denote the attributes for nodes, edges, and between edges, respectively. The matrices $H^{(V,k)}$, $H^{(E,k)}$, and $H^{(B,k)}$ denote the corresponding hidden states after the k-th layer. The outputs of the model are the embeddings for the road segments, intersections, and between edges (i.e., the edges in the dual representation).

We only use the embeddings for the road segments as input for our evaluation tasks. In our setup, we use graph reconstruction as the training objective. The number of fusion layers is set to two, and we use interactional fusion with attention aggregation as proposed by the authors. For this model, we were able to derive the original code from the authors, where we modified the training objective and the input pipeline to fit our data.



(a) Relational Fusion Network (b) Relational Fusion Layer

Figure 6.5: Overview of the relational fusion method, showing (a) a K-layered relational fusion network and (b) a relational fusion layer [7].

HRNR

The Hierarchical Road Network Representation (HRNR) model [6] is a multi-level neural architecture specially designed for learning on road networks. It consists of a hierarchical three-layer architecture corresponding to *"functional zones"*, *"structural zones"*, and *"road segments"*, respectively.

The module applies spectral clustering [101] on the road network graph to generate structural regions. Road segments that are assigned to the same cluster are aggregated into a structural region. The structural regions are furthermore aggregated into functional zones, which consist of functionally related structural regions. The model uses two different learnable assignment matrices which assign road segments to structural regions denoted as A^{SR} and structural regions to functional zones denoted as A^{RZ}, respectively. The matrices hold the probability distributions that a road segment is in a specific structural region and a structural region is in a specific functional zone.

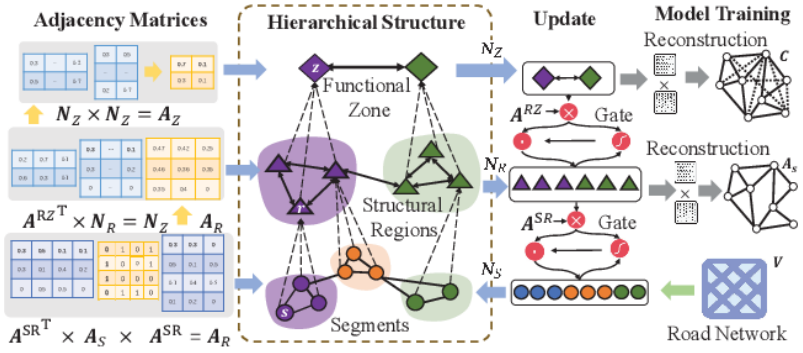


Figure 6.6: Architecture overview of the HRNR model [6].

Furthermore, the model proposes a three-level hierarchical update mechanism for learning the road segment embeddings, which updates the different levels using graph convolution message passing between the levels. This approach enables the embeddings to capture structural and functional dependencies. Figure 6.6 shows the overall architecture with the described components. In our setup, we use the same hyperparameters and training objectives as proposed by the authors in their experiments. For this model, the authors provided the original code base of their model. We extended the original code by a wrapper to work with our datasets.

6.1.4 Hyperparameter Setup

This section explains the hyperparameter setup for our proposed models considered in this evaluation, namely GTC, TSD, and GTN. Foremost, all models, including the baselines, output representations with 128 dimensions. An exception is the GTN model which outputs representations with 256 dimensions since the initial embedding is the concatenation of the GTC and TSD embeddings. The GTC, TSD, and GTN models are trained using the Adam optimizer [102] with a learning rate of 0.01, 0.01, and 0.0005, respectively. For TSD the batch size, walk length, walks per node, context size, and the number of negative samples are set to 128, 30, 25, 5, and 10, respectively. The k parameter for the GTC model is set to 2 with the bidirectional variant. The GTN model uses 2 transformer layers with 4 attention heads per layer and a batch size of 128.

6.1.5 Metrics

The following introduces the metrics we use to evaluate the proposed models on different tasks. The section is divided into subsections explaining the metrics used for regression and classification separately. The metrics are derived from evaluations in related papers [6, 4].

Classification

For road label classification we use accuracy, F1-macro and the weighted-averaged F1 metric [103] for evaluating the performance of the models. For trajectory-based classification tasks including next location and destination prediction, only accuracy is considered as an evaluation metric [6].

Accuracy Accuracy measures how many samples were correctly classified, both negative and positive, and is one of the most used metrics in multi-class prediction problems. The metric is defined as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.3}$$

where TP are the true positive, FP false positive, TN true negative and FN false negative predicted samples. True positives and true negatives are the elements correctly classified by the model, while false positives and false negatives are wrongly classified. The accuracy is therefore the proportion of correctly classified samples.

F1-Score The F1-Score [104] combines precision and recall by calculating the harmonic mean and is defined as:

$$\text{F1-Score} = 2 * \frac{precision * recall}{precision + recall} \tag{6.4}$$

where precision and recall are defined as:

$$precision = \frac{TP}{TP + FP} \tag{6.5}$$

$$recall = \frac{TP}{TP + FN} \tag{6.6}$$

Precision expresses the proportion of samples that are predicted to be positive and are actually positive. Recall measures the ability of the model to find all positive samples in the dataset, i.e., the predictive accuracy for the positive class. The F1-score can be seen as the weighted average of precision and recall. For the multi-class case as in our experiments, there are two variants of the F1-score that we consider, namely macro and weighted-averaged F1-score. Since the F1-score is calculated for each class, the variants adopt different averaging methods in the calculation. Macro F1 gives all classes the same weight by calculating the arithmetic mean (unweighted) of the per-class F1-scores. In contrast, the weighted average F1-score takes the weighted mean of all per-class F1-scores. The weight is determined by the proportion of class occurrences in the dataset relative to all occurrences. This is specifically a useful metric for imbalanced class distributions since the contribution to the score of a class is weighted by its size.

Regression

For regression tasks, including mean speed and travel time prediction, we use MAE and root mean squared error (RMSE), which are common metrics to measure the accuracy of continuous variables [105].

MAE The MAE measures the average magnitude of the errors in a set of predictions, without considering any directions. It is defined as follows:

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{6.7}$$

where y_i is the ground truth and $\hat{y}_i$ is the prediction of sample i. The metric is calculated as the absolute difference between prediction and actual observation without any weighting.

RMSE The RMSE measures the average magnitude of the error. It is defined as follows:

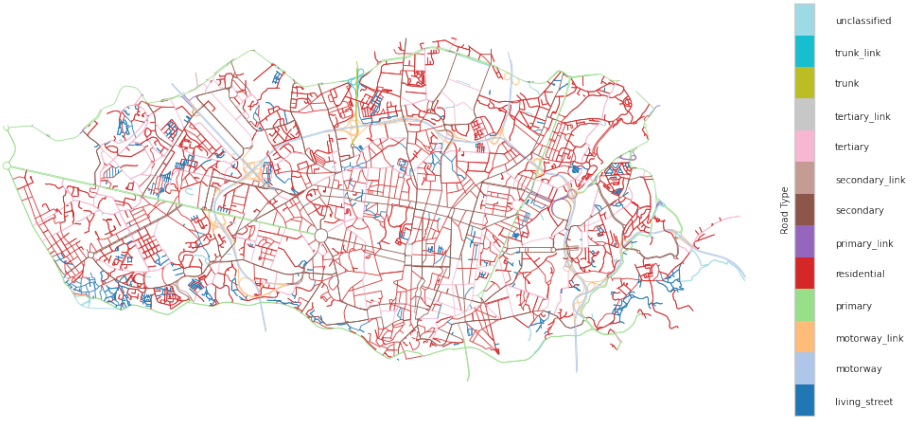
$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{6.8}$$

where y_i is the ground truth and $\hat{y}_i$ is the prediction of sample i. The metric is calculated as the square root of the squared differences between prediction and actual observation.

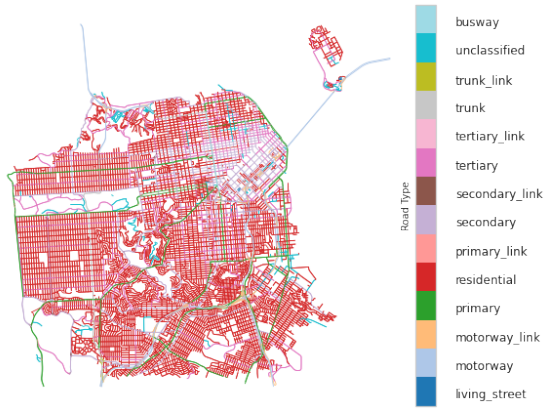
6.2 Explanatory Data Analysis

In section 6.1.1, a rough statistical overview of the used datasets was given. This section will explanatory examine the datasets in more detail to get a more profound insight into the available data. Structural properties as well as trajectory data of the cities are investigated in the following.

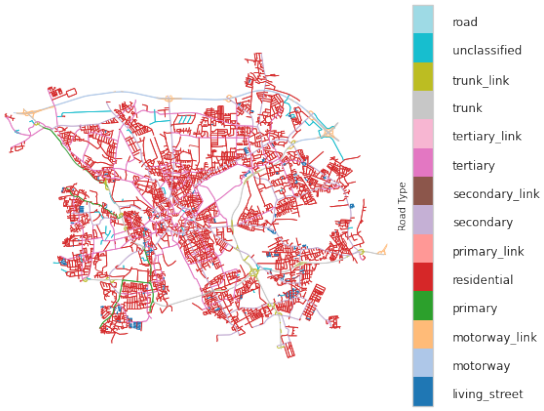
Figure 6.7 visualizes the used road networks of (a) Porto, (b) San Francisco, and (c) Hanover. The edges are assigned to a street category by color. The images share the same color map, but the color maps of (b) and (c) are slightly different since they each have an extra road category tag, namely "busway" for San Francisco and "road" for Hanover. An explanation of the individual tags can be found inside appendix C in table C.5. It is noticeable that the road networks of San Francisco and Hanover are much denser than in Porto, with San Francisco having by far the densest road network. San Francisco contains mostly residential areas (red blocks) that are linked by primary roads. Residential streets in San Francisco are arranged very close together in squares to each other. A combination of other street types is located in the center of San Francisco in the upper-right corner of the road network.



(a) Porto



(b) San Francisco



(c) Hanover

Figure 6.7: Road network visualization of (a) Porto, (b) San Francisco, and (c) Hanover. The road segments are colored according to the corresponding road type.

Hanover's street network also consists largely of residential streets. City districts, however, are more visible in this network and are more clearly delineated than in San Francisco. The districts are connected by tertiary and primary streets. In the upper area is the "Autobahn", which is marked as a motorway (bright blue line segment).

Porto, on the other hand, has a very open street network with mainly residential streets and living streets similar to Hanover. However, individual districts are not easily identifiable. The city is encircled by a primary street construct, which leads from the left into the interior to a large traffic circle in the center. The city center is circled by a motorway, which connects the outer parts of Porto with the center.

The networks are fundamentally unique in their structure and road type occurrences. This is also reflected in Figure 6.8, which visualizes the occurrence of the road categories.

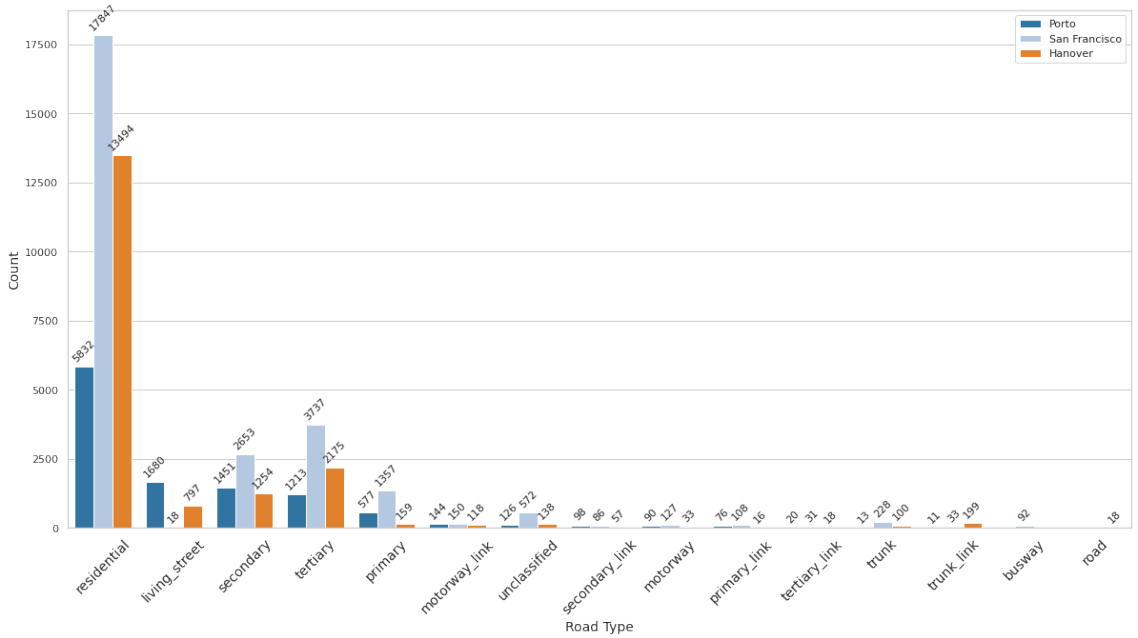
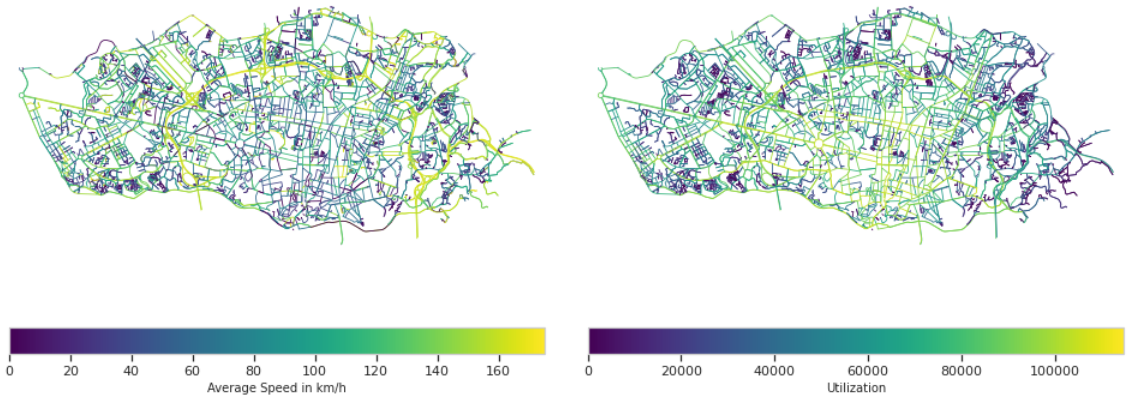


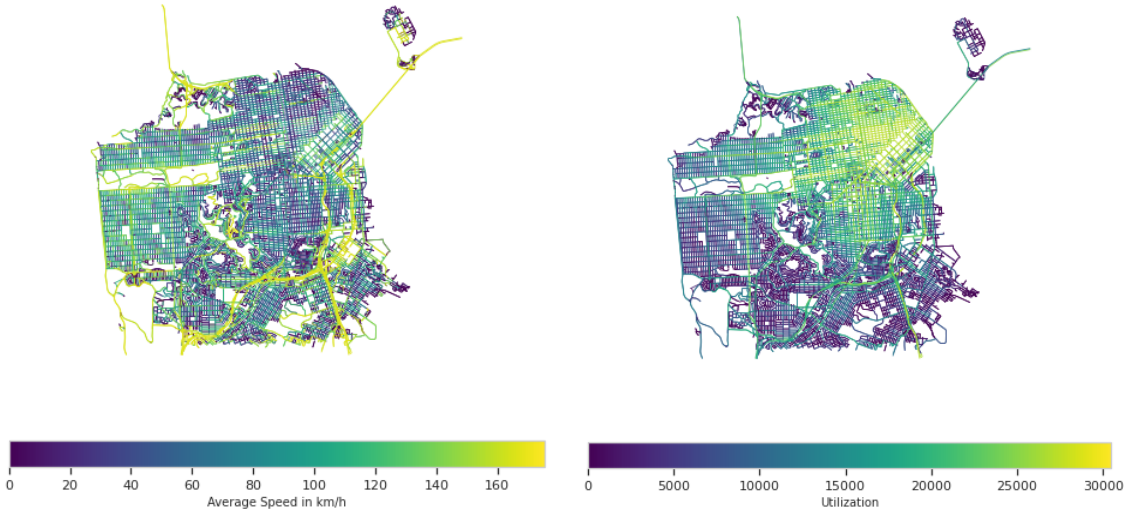
Figure 6.8: Road type distribution for Porto, San Francisco, and Hanover. The number above the bars shows the exact count of the respective road type in the corresponding road network.

All three cities have numerous residential streets, but it should be noted that San Francisco has three times as many of these streets as Porto and nearly twice as many as Hanover. Living streets are almost non-existent in San Francisco, while Porto and Hanover have some of these streets. In all cities, there are additionally primarily, secondary and tertiary streets. The figure further shows that many street categories are relatively rare in all cities, such as linking streets because they often occur in combination with the main street category, for example, "motorway" and "motorway_link". This makes the distribution of road categories extremely unbalanced.

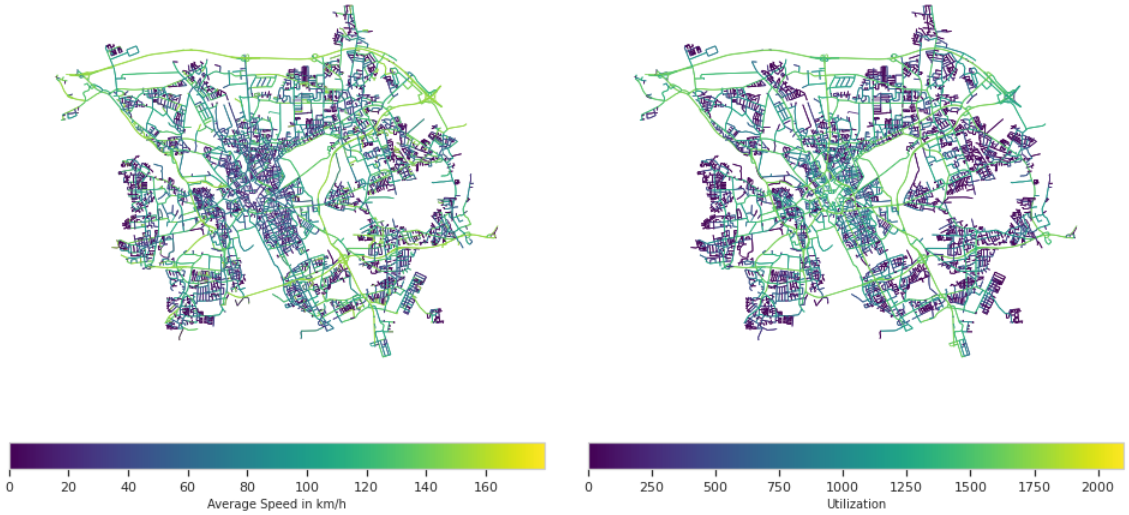
To get a better understanding of the behavior inside the road network and the quality of the trajectory data, we visualized the aggregated utilization and the average mean driving speed in the available period for each city under consideration. The visualization can be seen in figure 6.9, where the left images show the average driving speed in km/h and the right images show the aggregated utilization for (a) Porto, (b) San Francisco and (c) Hanover. Brighter color expresses a higher average driving speed or higher utilization.



(a) Porto



(b) San Francisco



(c) Hanover

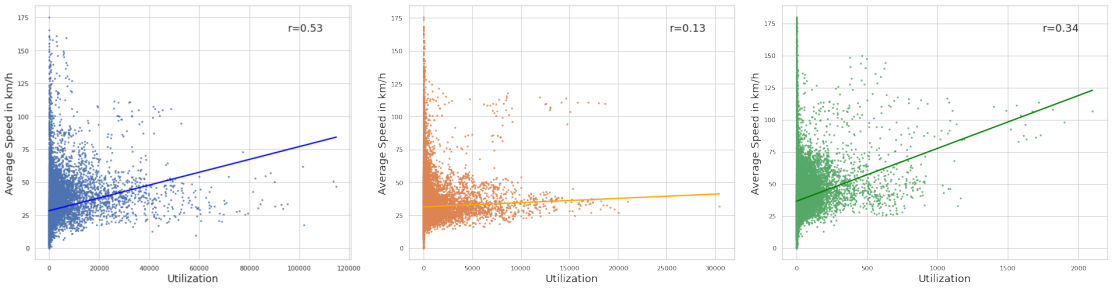
Figure 6.9: Average driving speed and utilization visualizations for (a) Porto, (b) San Francisco, and (c) Hanover. The left image shows the average driving speed and the right image the utilization of the corresponding road networks. The shown data is aggregated over the complete available timeframe of the trajectory data. The colors are determined by grouping the values into bins. For the average driving speed plot, we choose a bin size of 100, 20, and 20, while for the utilization plot a bin size of 200, 50, and 20 for Porto, San Francisco, and Hanover is used, respectively.

Starting with Porto, it is evident that the highest average speed is on the motorway around the center of Porto. In addition, the primary streets in the outskirts of the city also have a relatively high average speed. The secondary and residential streets in the center tend to have a low average speed. This is due to the high occupancy rate, which can be seen in the utilization plot. High utilization tends to cause congestion, which lowers the average speed. The trajectory data covers the road network comparatively well. The center and the connecting primary streets are highly utilized, while the areas around have low utilization or even no utilization at all. Specifically, the living streets have mostly zero utilization.

Further interesting patterns can be observed in the visualization for San Francisco. Foremost, the coverage of trajectory data on this road network does not seem as severe as in Porto, which is partly due to the smaller number of trajectories in the dataset. The utilization plot shows that the main utilization is in the center of the city and gets less in the outgoing directions. In the residential areas around the center, it is less to zero utilization, except for the primary roads that run through the residential areas. Similar to the center, these have a high utilization but, at the same time, a high average driving speed. However, the driving speed on the primary roads decreases towards the center, which is due to higher utilization and congestion. The center, on the other hand, has a relatively low average driving speed due to speed limits and high traffic load.

Hanover has very similar traffic patterns. The most utilization is in the center of the city, on the in- and outgoing tertiary streets, and on the motorway (Autobahn at the top of the plot). Specifically, the outer districts which are mostly living, and residential streets have zero utilization in the available time frame. One reason for this is that the trajectory dataset for this city contains the fewest trajectories $(50,411)$, which results in a low coverage (0.838%) of the road network with the available trajectory data.

To summarize, the traffic patterns seem very similar between the cities. In all cities under consideration, the center and the primary/tertiary/motorway streets are highly utilized. The surrounding areas are mostly not as utilized as the center. Furthermore, the average speed is comparatively low in the center areas of the cities, while it is high in larger street segments like on motorways or primary streets. However, the trajectory datasets also have different qualities due to different coverage. The Porto dataset models the road network behavior most extensively, followed by San Francisco and Hanover.



(a) Porto (b) San Francisco (c) Hanover

Figure 6.10: Scatter plot of the utilization against the average driving speed for each road segment in the corresponding road network. The line shows a linear regression through the points, and the value in the upper-right corner denotes the corresponding Spearman correlation coefficient [106]. Figures (a), (b), and (c) show the distributions for Porto, San Francisco, and Hanover, respectively.

Figure 6.10 shows the average driving speed plotted against the utilization for all three cities. The line in each plot is a linear regression, showing how the average speed and utilization correlate to each other. The number in the upper-right corner shows the Spearman correlation coefficient [106]. We chose this metric because it is not as sensitive to outliers as other correlation metrics [107]. The plot illustrates the relationship between utilization and driving speed. It is noticeable that Porto and Hanover have a relatively high correlation between utilization and average speed, with Hanover having the largest, which can be attributed to road types like the Autobahn. This type of road allows high utilization and high driving speeds. San Francisco has a comparatively low correlation since most of the more utilized road segments are in the lower average speed range. Overall, the distribution and relations between speed and utilization are very diverse between the cities, especially for Hanover, where often high-speed road segments have high utilization values. However, there is no significant correlation between the utilization and the average driving speed, which holds for all cities under consideration.

Figure 6.11 shows the density distribution of the average driving speed on the road segments calculated from the trajectory datasets for all three cities. To get a better estimate of the available data, we removed the road segments that have an average speed of zero, because of missing data, from the visualization.

The distributions fall out differently, with Porto and Hanover being the most similar. The shapes of both curves are close to each other, but the curve of Hanover is shifted in the right direction, i.e., the average driving speed is much higher (32 km/h for Porto and 41 km/h for Hanover). Regarding the mean speed, Porto and San Francisco have similar characteristics, while the shapes of the curves are fundamentally different. Both mean driving speeds are around 32 km/h, while San Francisco has a very compressed distribution, with a peak at approximately 30 km/h and Porto has a more scattered distribution with many road segments that have a speed between 10 and 50 km/h. In the distribution from San Francisco, most speeds tend to be in the range between 20 and 40 km/h. Nevertheless, the average of all speeds is very similar to Porto, since Porto has relative to San Francisco many road segments with higher average driving speeds.

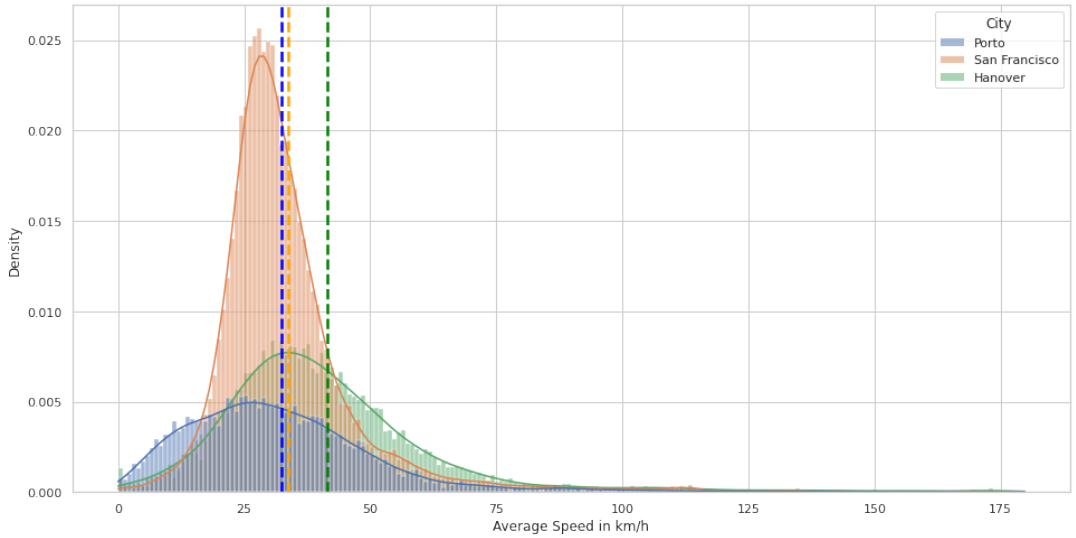


Figure 6.11: Average driving speed density distribution over all road segments for Porto, San Francisco, and Hanover. The dotted vertical lines mark the mean speed of the distributions for each city.

In conclusion, the explanatory analysis of the three examined datasets shows that the cities are unique in their structure and size. In all cities, the street segment categories are unevenly distributed, with the majority being residential streets. Regarding the behavior in the road network, some commonalities can be derived from the trajectories. For all cities, the highest utilization is in the center and correlates with a lower average speed. In addition, especially primary, secondary, and motorways show high utilization at relatively high average speeds. It must be noted that the analysis has shown that the trajectory datasets differ significantly in their scope and quality, which can have an impact on the performance of the models.

6.3 Experimental Results

This section presents the main results of this work. We evaluate our proposed model, namely the GTN model, against several baselines and SOTA approaches from the literature, as described in section 6.1.3. All models are evaluated on five different train-test splits for each task to get a better estimate of the performance variance and to reduce the probability of a randomly good performance because of a lucky seed [108]. An exception is the road label classification task, where the models are only evaluated on a single seed since the task is not dependent on trajectory data and is furthermore evaluated using five-fold cross validation. The downstream decoder LSTM networks in the trajectory prediction tasks are trained for 10 epochs for each evaluated model individually.

The tables 6.5-6.9 show the results of our evaluation for all proposed tasks (see section 6.1.2 for details on the tasks). Each row shows the results for a single model regarding different metrics on all datasets. The best value for each column is marked in bold.

	Road Label Classification								
	Porto			San Francisco			Hanover		
	F1 Macro	Accuracy	F1 Weighted	F1 Macro	Accuracy	F1 Weighted	F1 Macro	Accuracy	F1 Weighted
PCA	0.103	0.565	0.45	0.076	0.677	0.57	0.233	0.841	0.83
Deepwalk	0.257	0.403	0.379	0.171	0.648	0.584	0.209	0.616	0.584
Node2Vec	0.224	0.396	0.37	0.176	0.65	0.585	0.187	0.626	0.589
GAE (GCN)	0.411	0.661	0.624	0.07	0.664	0.546	0.334	0.794	0.783
GAE (GAT)	0.393	0.652	0.619	0.131	0.676	0.588	0.327	0.761	0.745
RFN	0.088	0.498	0.383	0.126	0.672	0.572	0.128	0.734	0.655
SRN2Vec	0.068	0.505	0.352	0.078	0.66	0.527	0.076	0.723	0.612
HRNR	0.132	0.541	0.445	0.147	0.692	0.641	0.14	0.749	0.701
Toast	0.206	0.441	0.399	0.069	0.662	0.532	0.097	0.715	0.622
GTN	**0.478**	**0.670**	**0.66**	**0.411**	**0.77**	**0.746**	**0.532**	**0.859**	**0.857**

Table 6.5: Results for the road label classification task.

The first table (6.5) shows the evaluation results for the road label classification task. Our model, namely the GTN, performs best on all datasets across all metrics in this task. It performs 16.3% better on the Porto dataset, 133.5% better on the San Francisco dataset, and 64% better on the Hanover dataset than the respective second-best performing model on each dataset regarding the F1 macro score. On the Porto and Hanover dataset, our model is primarily followed by convolution-based approaches (GAE with a GCN encoder and a second variant with a GAT encoder), which are furthermore followed by walk-based approaches (Node2Vec and DeepWalk). The approaches which focus on road networks perform comparatively poorly on this task, which can be attributed to the fact that the road labels were removed from the training process for this task. Unfortunately, road

labels are an important part of the learning process for these models. For this task, however, it is important that the models have not seen the labels beforehand, since the goal is to test how well the structure of the network and the semantic connections between the road segments are captured by the models. The success of convolution-based approaches can be attributed to the usage of features inside the aggregation, some of which, such as the speed limit, can correlate with the road type.

Similar trends can be observed in the mean speed prediction task. Moreover, on this task, the performance of our model is significantly better than that of competing models. To be exact, 11% better on the Porto dataset, 14.5% better on the San Francisco, and 6% better on the Hanover dataset than the respective second-best model for each dataset. Besides our model, the distribution of performance across the datasets is extremely diverse. On the Porto dataset, HRNR and Toast seem to perform particularly well, and RFN and SRN2Vec perform worst. Standard methods like Node2Vec and GAE models perform in the average range. A different distribution can be observed on the San Francisco dataset. Apart from our model, walk-based approaches perform particularly well in this case. They are followed by RFN, PCA, convolution-based methods, HRNR, and Toast. As on the Porto dataset, SRN2Vec performs the worst on this task. Interestingly, on the Hanover dataset, after our model, the convolution-based models perform best (GAE with different encoders) followed by HRNR, PCA, RFN, and Toast. Again, the SRN2Vec model performs worst. Remarkable is that over all datasets a simple PCA gives average results and is never worse than other models or even better. This could indicate that the mean driving speed is heavily dependent on road segment features rather than pure structure.

	Mean Speed Prediction					
	Porto		San Francisco		Hanover	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
PCA	14.506 ± 0.001	20.532 ± 0.0	9.921 ± 0.032	15.908 ± 0.016	17.244 ± 0.029	24.616 ± 0.015
Deepwalk	14.562 ± 0.001	20.229 ± 0.001	9.579 ± 0.032	14.733 ± 0.019	18.05 ± 0.036	24.815 ± 0.032
Node2Vec	14.293 ± 0.0	19.954 ± 0.001	9.717 ± 0.028	14.87 ± 0.016	18.058 ± 0.047	24.852 ± 0.032
GAE (GCN)	14.211 ± 0.004	20.378 ± 0.002	10.484 ± 0.034	16.34 ± 0.026	16.639 ± 0.072	24.017 ± 0.123
GAE (GAT)	14.256 ± 0.004	20.394 ± 0.003	10.091 ± 0.019	15.734 ± 0.028	16.823 ± 0.042	23.993 ± 0.024
RFN	15.169 ± 0.08	20.947 ± 0.042	9.833 ± 0.032	15.686 ± 0.026	17.881 ± 0.064	24.925 ± 0.047
SRN2Vec	15.914 ± 0.001	22.185 ± 0.001	12.245 ± 0.058	17.734 ± 0.064	19.69 ± 0.169	26.787 ± 0.115
HRNR	13.754 ± 0.002	19.843 ± 0.006	10.108 ± 0.029	15.331 ± 0.055	17.154 ± 0.088	24.185 ± 0.069
Toast	13.869 ± 0.003	19.547 ± 0.001	10.114 ± 0.026	15.638 ± 0.02	18.004 ± 0.064	25.114 ± 0.058
GTN	$\mathbf{12.238 \pm 0.019}$	$\mathbf{18.209 \pm 0.02}$	$\mathbf{8.193 \pm 0.056}$	$\mathbf{12.846 \pm 0.077}$	$\mathbf{15.642 \pm 0.061}$	$\mathbf{23.088 \pm 0.109}$

Table 6.6: Results for the mean speed prediction task.

Table 6.7 shows the results for the travel time estimation task. Our model outperforms all competitor models on the Porto and San Francisco datasets on both metrics. The results on the Porto dataset are very close overall, with SRN2Vec performing particularly well alongside our model. The worst performer on this dataset is the PCA followed by the RFN model as the second-worst performer. On the San Francisco dataset, SRN2Vec again performs well alongside our model, followed by walk-based approaches and HRNR. Convolution-based approaches, along with PCA, are the worst-performing models on this dataset. On the Hanover dataset, our model has the best performance regarding the MAE metric but is only the second-best model concerning the RMSE metric. It gets slightly outperformed by SRN2Vec on the latter metric. SRN2Vec is furthermore the second-best model on the MAE metric, with a small gap to the result of our model. This is followed by Toast and HRNR regarding the MAE with a considerable gap on both metrics.

Standard convolutional and walk-based methods perform equally in this case and are in the average field regarding their performance placement. The PCA and the RFN models have the overall worst performance.

Overall, our model and SRN2Vec perform extraordinarily well on this task, and it seems like both models learn a stable representation that yields high performance under different conditions regarding the trajectory data.

	Travel Time Estimation					
	Porto		San Francisco		Hanover	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
PCA	93.405 ± 1.693	129.516 ± 1.7	121.061 ± 3.746	245.33 ± 43.754	124.81 ± 10.035	198.254 ± 7.437
Deepwalk	76.778 ± 0.876	110.633 ± 1.272	102.721 ± 4.792	226.619 ± 45.759	117.888 ± 4.298	187.396 ± 8.723
Node2Vec	76.764 ± 1.25	110.771 ± 1.306	102.663 ± 3.202	226.644 ± 46.043	117.639 ± 1.658	188.254 ± 4.705
GAE (GCN)	75.894 ± 0.574	109.481 ± 0.856	118.656 ± 1.597	243.448 ± 44.329	117.849 ± 4.128	189.181 ± 4.56
GAE (GAT)	76.67 ± 1.315	109.639 ± 1.046	105.425 ± 0.894	234.666 ± 50.229	117.783 ± 10.444	188.175 ± 7.149
RFN	79.292 ± 1.141	112.554 ± 1.495	110.341 ± 11.116	235.036 ± 46.146	125.278 ± 7.554	194.96 ± 7.951
SRN2Vec	75.199 ± 0.918	108.721 ± 1.732	102.396 ± 5.279	230.99 ± 45.103	110.315 ± 8.204	$\mathbf{180.965 \pm 9.248}$
HRNR	77.989 ± 0.8	112.726 ± 2.192	104.062 ± 9.505	228.572 ± 45.045	116.977 ± 6.47	187.816 ± 4.741
Toast	76.34 ± 0.717	110.205 ± 1.121	107.1 ± 4.251	232.12 ± 45.037	116.305 ± 5.941	189.952 ± 5.511
GTN	$\mathbf{74.985 \pm 0.64}$	$\mathbf{108.683 \pm 0.673}$	$\mathbf{99.925 \pm 2.123}$	$\mathbf{226.618 \pm 40.529}$	$\mathbf{109.001 \pm 3.507}$	181.896 ± 5.253

Table 6.7: Results for the travel time estimation task.

Regarding the next location task, our model does not perform better than other models. However, the GTN model is the second-best-performing model on the Porto and San Francisco dataset. On the Hanover dataset, our model performs comparatively poor and is therefore only in eighth place in terms of performance. On the Porto dataset all models, except the PCA seem to perform relatively equivalent with a maximal difference of around 18% between the best and worst performing model. All models could learn a good representation of the local structure for Porto, which the decoder model was able to exploit. A larger difference regarding the performance can be observed on the San Francisco dataset. The best performing model, namely HRNR is 18.3% better than the second best performing model, namely our model, and around two times better than the worst performing model (except PCA), namely Toast. This is probably because the trajectories in this dataset are not as covering as in the Porto dataset and are largely concentrated in the city core, with occasional trajectories extending further out (see figure 6.9). In addition, the average node degree (see table 6.1) is significantly higher on this dataset, making it more difficult for the models to predict the next location.

	Next Location Prediction		
	Porto	San Francisco	Hanover
	Accuracy	Accuracy	Accuracy
PCA	0.566 ± 0.013	0.092 ± 0.018	0.051 ± 0.033
Deepwalk	0.635 ± 0.005	0.301 ± 0.01	0.197 ± 0.05
Node2Vec	0.635 ± 0.007	0.296 ± 0.013	$\mathbf{0.211 \pm 0.023}$
GAE (GCN)	0.628 ± 0.012	0.203 ± 0.006	0.169 ± 0.007
GAE (GAT)	0.618 ± 0.01	0.310 ± 0.005	0.17 ± 0.014
RFN	$\mathbf{0.67 \pm 0.006}$	0.324 ± 0.01	0.14 ± 0.019
SRN2Vec	0.627 ± 0.004	0.303 ± 0.014	0.127 ± 0.009
HRNR	0.647 ± 0.006	$\mathbf{0.422 \pm 0.028}$	0.151 ± 0.012
Toast	0.639 ± 0.007	0.202 ± 0.009	0.16 ± 0.046
GTN	0.667 ± 0.005	0.355 ± 0.009	0.137 ± 0.013

Table 6.8: Results for the next location prediction task.

Similar reasons can be given for the generally poorer performance on the Hanover dataset, where the number of available trajectories and coverage is again significantly lower

than on the San Francisco dataset. Since our model is partly based on trajectory data and incorporates transition flows, the focus on wide local structure decreases with less variational trajectory data and therefore the performance on this task.

On the destination prediction task, the GTN model performs much better relative to the next location task. Our model performed superior on all evaluated datasets. Regarding the Porto dataset, the accuracy is about 2% higher than that of the second-best model, namely SRN2Vec. On the San Francisco dataset, our model is 11% better than the second-best model (Node2Vec) on this dataset. On the Hanover dataset, our model performs 73% better than the second-best model, namely DeepWalk. The generally good performance of our model on this task can be explained by the inclusion of trajectory data in the training process. The transformer model in GTN learns the behavior through the trajectories and thus also the patterns induced by them. The competing models do not have this advantage, which is also reflected in the performance.

	Destination Prediction		
	Porto	San Francisco	Hanover
	Accuracy	Accuracy	Accuracy
PCA	0.261 ± 0.002	0.028 ± 0.003	0.022 ± 0.004
Deepwalk	0.297 ± 0.002	0.112 ± 0.002	0.063 ± 0.006
Node2Vec	0.301 ± 0.002	0.115 ± 0.002	0.062 ± 0.005
GAE (GCN)	0.285 ± 0.003	0.049 ± 0.004	0.041 ± 0.005
GAE (GAT)	0.286 ± 0.002	0.091 ± 0.002	0.046 ± 0.005
RFN	0.286 ± 0.003	0.079 ± 0.001	0.023 ± 0.004
SRN2Vec	0.303 ± 0.004	0.108 ± 0.003	0.04 ± 0.009
HRNR	0.297 ± 0.003	0.109 ± 0.001	0.054 ± 0.006
Toast	0.299 ± 0.002	0.084 ± 0.002	0.044 ± 0.004
GTN	$\mathbf{0.309 \pm 0.001}$	$\mathbf{0.128 \pm 0.001}$	$\mathbf{0.109 \pm 0.005}$

Table 6.9: Results for the destination prediction task.

Our initial goal was to develop a model that can learn representations that work robustly on different tasks. To make the robustness of our model and the baselines measurable by a metric, we propose the generalization score. This metric measures the generalization, i.e., the robustness, of a model based on results across different tasks and datasets. For this purpose, the score is considered in relation to other evaluated models and is calculated as follows:

$$S_g(m) = \sum_{t=0}^{T}(\sum_{d=0}^{D}(\frac{S_{td}(m) - min(S_{td})}{max(S_{td}) - min(S_{td})}) * \frac{1}{D}) * \frac{1}{T} \tag{6.9}$$

where T is the number of different evaluated tasks, D is the number of different datasets and $S_{td}(m)$ is the score of model m (optionally aggregated over several seeds) on task t and dataset d. The formula calculates the relative advantage/disadvantage of a model in comparison to competing models and forms the average over all tasks and datasets. Specifically, a score of 1 would denote a model that outperforms all other competing models on all tasks, and datasets. For road label classification we use the F1 macro score, for mean speed and travel time prediction the MAE score and for next and destination prediction the accuracy score to calculate the generalization score. Table 6.10 shows the generalization scores for each evaluated model. Our model is superior on all tasks, except for the next location task, where it reached the second-best score. The GTN model performs, in comparison to other models, stable and simultaneously with high scores across datasets on the tasks. This results in high generalization scores over the tasks

and a superior overall generalization score with around 0.953, which is an improvement of 57% to the second-best performing model.

	Generalization Score					
	Average Score over all Datasets					
	Road	Mean Speed	Travel Time	Next Location	Destination	**Final Score**
PCA	0.152	0.520	0.010	0.000	0.000	0.136
Deepwalk	0.351	0.477	0.741	0.737	0.694	0.600
Node2Vec	0.313	0.489	0.748	0.760	0.723	0.607
GAE (GCN)	0.469	0.551	0.507	0.556	0.313	0.479
GAE (GAT)	0.509	0.564	0.703	0.631	0.480	0.577
RFN	0.110	0.415	0.424	0.752	0.348	0.410
SRN2Vec	0.010	0.000	0.930	0.567	0.629	0.427
HRNR	0.176	0.581	0.717	**0.802**	0.645	0.584
Toast	0.127	0.500	0.713	0.571	0.540	0.490
GTN	**1.000**	**1.000**	**1.000**	0.767	**1.000**	**0.953**

Table 6.10: Generalization score for our model and all baseline models. The best score for each task is marked in bold. The last column on the right shows the final score calculated as the average of all task-specific scores for the corresponding models.

Overall, it can be summarized that our proposed GTN model performs exceptionally well on static tasks such as road label classification and mean speed prediction, as well as on dynamic trajectory-dependent tasks including travel time, next location, and destination prediction. The model captures road segment dependent properties as well as structural and trajectory-induced properties better than current SOTA models and can be applied to a wide variety of tasks, yielding robust results.

6.4 Ablation

This section shows the ablation studies we conducted for our proposed models. We start with the study for the GTN model and proceed with an ablation study for the proposed modules, namely GTC and TSD. All studies are based on the Porto dataset, as it is the most complete dataset with the most trajectories and the highest trajectory-to-road coverage (see section 6.1.1).

6.4.1 Study for GTN

In the following, we perform an ablation study in which we individually remove several key components from the GTN model, allowing us to evaluate the impact on the performance of each component [109].

Specifically, we evaluate the following model variants:

1. **GTN**: the base variant as described in section 5.2.3
2. **GTN-GAE**: a variant where we replace the GTC encoder with a GAE, which uses a two-layer GCN model as encoder
3. **GTN-DW**: a variant where we replace the TSD with a default DeepWalk model
4. **GTN-T**: a variant that trains the transformer without pre-trained embeddings

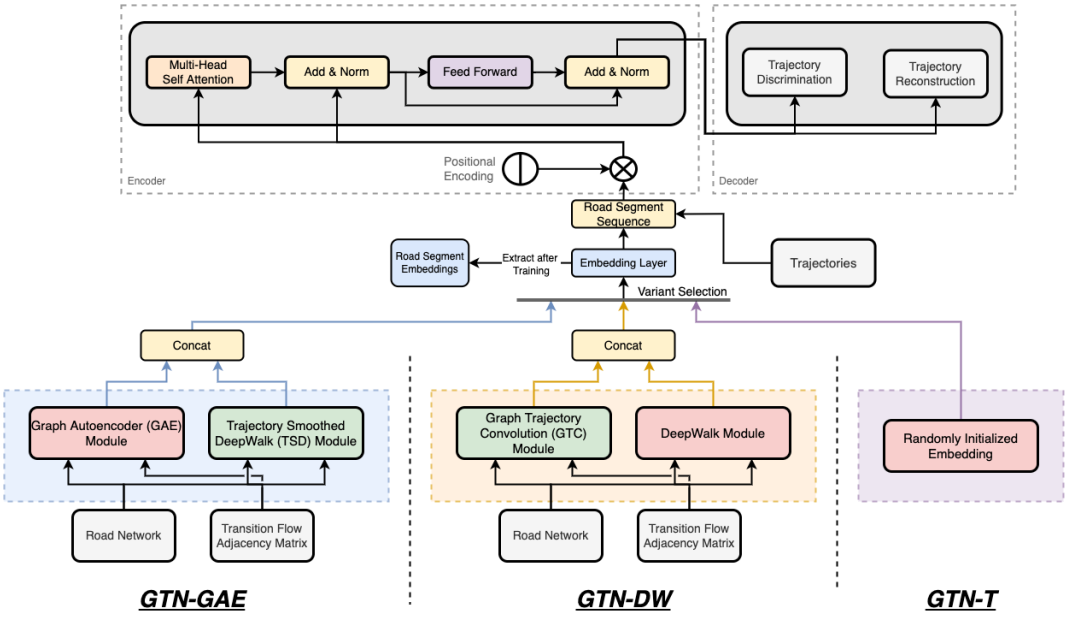
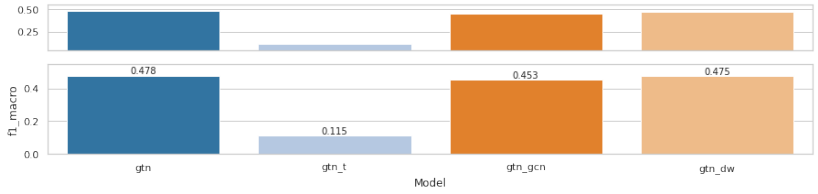


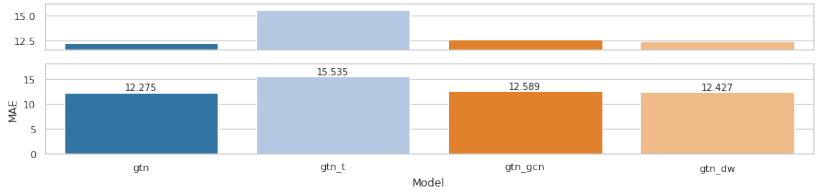
Figure 6.12: Architecture overview of the model variants for the ablation study. The basic architecture is the same as described in section 5.2.3 for the GTN model. The red box marks the replaced component for each variant.

An architectural overview of the variants can be seen in figure 6.12. All hyperparameters are the same as described in section 6.1.3 and 6.1.4 for the exchanged and the original modules. The performance is evaluated on all proposed tasks (see section 6.1.2).

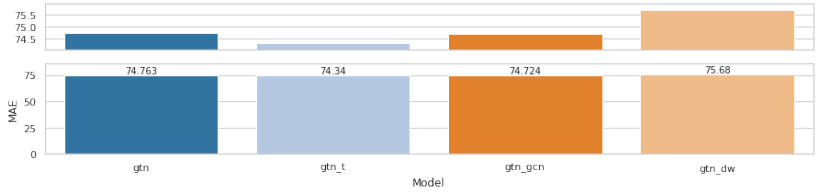
Each variant is trained for 10 epochs on 70% of the trajectory dataset. The remaining 30% of the dataset is used for evaluating on the trajectory tasks including travel time, next location, and destination prediction. From the remaining 30% of the dataset, we use 80% for the training of the LSTM networks and 20% for the testing. The downstream decoder LSTM networks in the trajectory prediction tasks are trained for 10 epochs. We use the same single seed for all variants, i.e., the split of the train and test set is always the same to guarantee a fair evaluation. A single seed run should approximate the overall performance of the models, which is evident from the lower standard deviation between seeds of the results in the main evaluation for the Porto dataset.



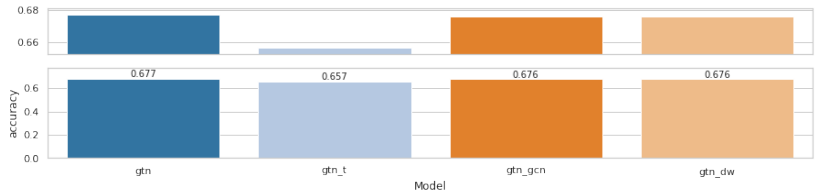
(a) Road Label Classification



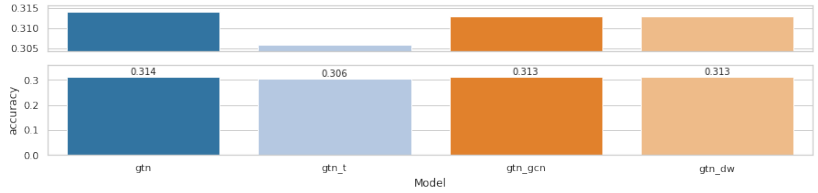
(b) Mean Speed Prediction



(c) Travel Time Estimation



(d) Next Location Prediction



(e) Destination Prediction

Figure 6.13: Results of the ablation study for the GTN model. All variants are trained and evaluated on the Porto dataset. The plot for each task is divided into two sub-plots, where the upper one visualizes the performance difference between the models and the lower one the absolute performance difference to the null point.

The results for each task and each variant can be seen in figure 6.13. Figure 6.13a shows the results for the road label classification task. The variants with pre-trained embeddings heavily outperform the variant which uses randomly initialized embeddings. This can be explained by the fact that the pre-trained embeddings capture a lot of information regarding the features and structural role of the road segments. Furthermore, it is noticeable that our basic variant additionally performs better than the variants with replaced modules. The GTC module seems to have a bigger impact on this task than the TSD module because removing the GTC module yields a lower score than removing the TSD module. Removing the TSD module makes no significant change in predictive power.

A similar trend can be seen in figure 6.13b showing the results for the mean speed task. The GTN-T variant performs worse than all other variants, and removing the GTC module has a larger negative impact than removing the TSD module. Our base model manages to outperform all variants on this task as well.

In the travel estimation task (see figure 6.13c), the results of all variants are very similar. It doesn't seem to make a difference whether pre-trained embeddings are used or which modules are applied. It is particularly noticeable that GTN-T is just as performant as the other variants, which is not the case with the other tasks. This phenomenon could be because travel time is highly dependent on temporal features [110] like day-time, weekday, or if it is a special day in the year, which GTN does not incorporate. Furthermore, this task is more dependent on the trajectories themselves, which all variants incorporate in the training process equally.

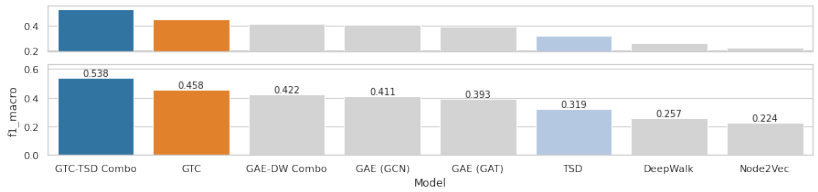
Figure 6.13d and 6.13e show the results of the next location and destination prediction tasks, respectively. On both tasks, our base model slightly outperforms the other variants. Furthermore, replacing the GTC or TSD module has the same effect on the performance. In contrast to the static tasks, GTN-T performed much better on the trajectory-based tasks, which is because those tasks are based on structural trajectory knowledge. The transformer in the GTN learns the road network transitions and travel semantics independently of the initial embeddings. Therefore, it makes sense, that the GTN-T model has competitive performance in contrast to the performance on the static structural tasks, such as road label classification and mean speed prediction.

In summary, the findings show that our variant with the proposed modules performs superior overall and therefore verifies the effectiveness of the used model components, namely GTC and TSD.

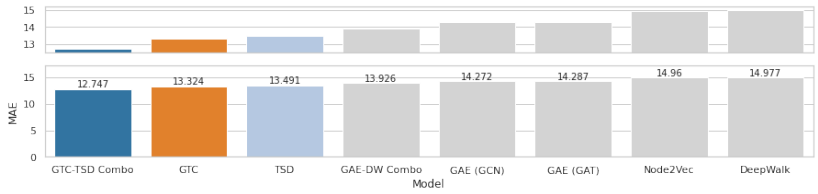
6.4.2 Study for GTC and TSD

To get more insights into how the different submodules of the GTN model perform, we do an ablation study regarding the GTC and TSD models. The goal of this study is to evaluate how well our submodules of the proposed model perform in comparison to similar approaches. This study evaluates the embeddings generated by the models on different ITS tasks and compares the performance to equivalent methods, namely a GAE with an GCN encoder, a second variant with an GAT encoder, Node2Vec, and DeepWalk. Our decision for the mentioned comparison models is because they use fundamentally similar methods to our submodules. The GTC model is just like a GAE model convolution-based and TSD is based on walk methods similar to Node2Vec and DeepWalk. Furthermore,

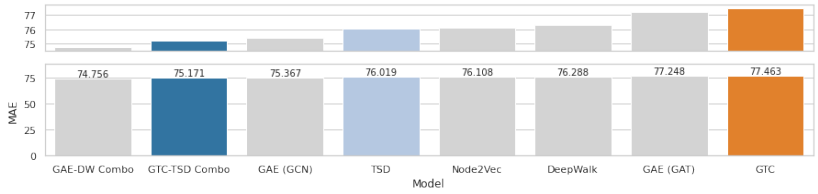
we compare the concatenation of the embeddings of our proposed models with the concatenation of the DeepWalk and GAE with an GCN encoder embedding. We use the same tasks and hyperparameters for the decoder models as in the main evaluation from section 6.3. The LSTM models in the trajectory tasks (next location, destination, and travel time) are trained for 10 epochs on the same train-test split as in the main evaluation. However, for this ablation, we only use a single seed for the train-test split.



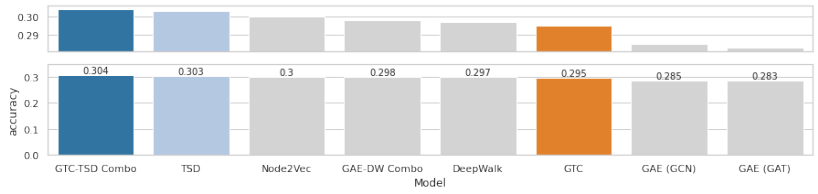
(a) Road Label Classification



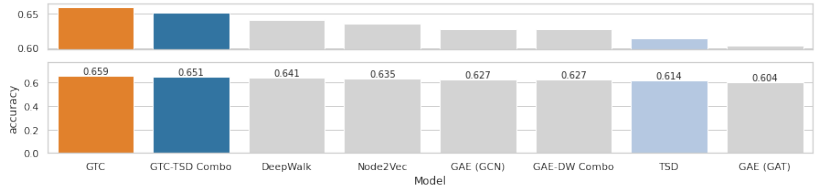
(b) Mean Speed Prediction



(c) Travel Time Estimation



(d) Next Location Prediction



(e) Destination Prediction

Figure 6.14: Performance results of the ablation study for the GTC and TSD models.

Figure 6.14 shows the results of the ablation study for all tasks. The bars are sorted by performance, from left (best performing) to right (worst performing). The plot for each task is divided into two sub-plots, where the upper one visualizes the performance difference between the models and the lower one the absolute performance difference to the null point. The bars of our proposed models are colored to highlight the performance positions between all models. The GTC model is colored orange, the TSD model is colored light blue, and the combination of both embeddings is colored dark blue.

Figure 6.14a shows the results for the road label classification task. It is evident that our models perform very well on this task. By far, the best score is achieved by the combined embedding of GTC and TSD, followed by our GTC model and the combination of a GAE and DeepWalk. Our models (TSD, GTC) perform better on this task than similar approaches. The worst performers on this task are walk-based approaches (Node2Vec, DeepWalk) since they do not include features, but only consider the structure of the road network.

Similar observations can be made on the mean speed prediction task (see figure 6.14b). Again, the combination of GTC and TSD performs best by far, followed by the respective unique models, namely GTC and TSD. The other walk-based approaches have the worst performance, followed by the convolutional methods and the combination of these. The reason that our methods perform relatively well on this task can be attributed to the fact that they learn representations based on trajectories. Trajectories, or rather the utilization that can be modeled with them, correlate with the average speed (see section 6.2). Road segments with many trajectories flowing through them and which have a low-speed limit or a certain road type like residential are most likely to have a lower average speed. Our models exploit, among other things, this knowledge, and therefore perform better than models that do not.

In the travel time estimation task (see figure 6.14c), our models do not perform significantly better in comparison to the other models. What is also noticeable is that here the combination of GTC and TSD performs better than the individual components. However, in this task, the combination of GAE and DeepWalk is slightly better than our combination. This is because a normal GAE performs much better on this task than our GTC model. The GTC model performs the worst of all models, which in turn propagates into the combined representation. TSD outperforms the other walk-based approaches, but not the GAE model with GCN encoder.

Regarding the location forecasting tasks, namely next location and destination prediction (see figure 6.14d and 6.14e, respectively), a reverse trend regarding the performance of our models is noticeable. In the next location prediction task, the GTC model performs much better than the TSD model. It even outperforms the combination of both embeddings slightly. However, on the destination prediction task, the GTC model gets outperformed by the TSD model, while the combination of both embeddings is slightly better than the TSD model. It appears that the TSD model learns long-term dependencies of road segments better than the GTC model, while the latter learns close dependencies more effectively. In contrast to the other models, our combination of embeddings outperforms all on both of the tasks. Despite our models, walk-based methods perform better than convolution-based methods on both tasks.

In summary, our models perform best on average across tasks in comparison to technically similar, more general models. Especially the concatenation of the embeddings shows a very positive effect on the performance of all tasks and compensates for the weaknesses of the individual representations. The concatenation creates a more stable embedding that captures both near and far dependencies between road segments. The study confirms our approach of incorporating trajectories into the learning process of the models and shows the effectiveness of this technique for road segment representation learning.

6.5 Parameter Study

One of the most important and interesting hyperparameters for our GTC model is the k-parameter. As described in section 5.2.1, the parameter controls the range over the trajectories for the convolution, i.e., $k = 1$ is equal to a weighted GCN convolution with one layer and higher k means considering more road segments along a trajectory. In the following, we will conduct a hyperparameter study to analyze the impact of k on the performance of GTC on different ITS tasks. Specifically, we evaluate different k-values ($k \in \{1, 2, 3, 4, 5, 6\}$) on both, the forward and bidirectional variants of the convolution. The evaluation tasks are the same as described in section 6.1.2. Each GTC model is trained for 5000 epochs with equal hyperparameters, except for the k parameter. The learning rate is set to 0.01 and the embedding dimension is 128. We evaluate 10 different seeds to get a better estimate of the performance variations between different train and test arrangements. For the mean speed and road category classification tasks, the dataset is randomly divided based on the seeds. For the remaining trajectory tasks, including time travel estimation, next, and destination prediction, we randomly sample 100000 trajectories from the test dataset used in the main study (see section 6.3) and split this subset into train and test datasets. The train-test ratio is $80/20$, i.e., 80% of the dataset is used for training and the remaining 20% for testing the downstream decoder LSTM model. The study is conducted on the Porto dataset (see section 6.1.1).

Figure 6.15 shows the results of the study for each task. Each dot marks the result for a single seed, either for the bidirectional (blue) or forward method (orange). The x-axis shows the different k-values and the y-axis the task-dependent metric. To visualize the deviation between seeds, we used a violin plot for the road label classification and box plots for the remaining tasks.

Figure 6.15a shows the results for the road label classification. It can be seen that the deviation between the different seeds is small and has hardly any influence on the performance. The same is true for the choice of method. The bidirectional method performs approximately equivalent to the forward method with the same k. The parameter k, however, is a significant factor for the performance. The model performs best by far when k is set to 2. With larger k the performance decreases, whereby $k = 6$ performs again clearly worse than $k = 5$.

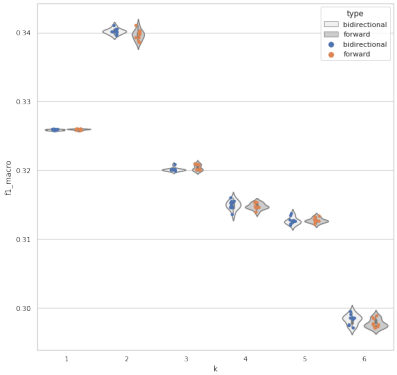
In the mean speed prediction task (see figure 6.15b), similar to the road label classification task, the performance decreases as k increases. Higher k values $(4, 5, 6)$ perform much worse than lower k values $(1, 2, 3)$. On average, the model with $k = 2$ performs best, with the bidirectional method being slightly better than the forward method.

Interestingly, the next and destination prediction tasks (see figure 6.15d and 6.15e) perform opposite to the k value. With higher k, the performance of the next location task gets worse, while the performance of the destination task gets better. The best performance on the next location task is achieved with $k = 2$ and the bidirectional method. For the destination task, the best performance is obtained with $k = 6$ and the forward method.

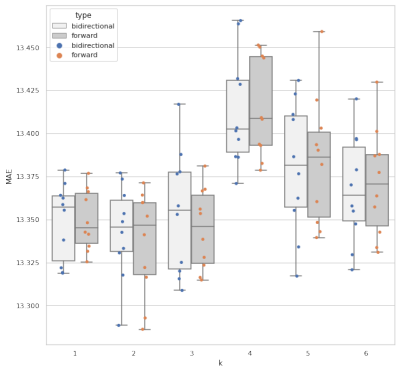
No real trend can be seen in the results for the travel time task (see figure 6.15c). The model performs similarly for all k values, with the forward method performing better on average than the bidirectional method. Another striking observation is the high deviation between the seeds. For some seeds, the performance fluctuates extremely.

For example, with $k = 1$ and the forward method. On average, the MAE is around 83 here, but for one seed it jumps to 92. This phenomenon seems to be spread over all k values and methods on this task.

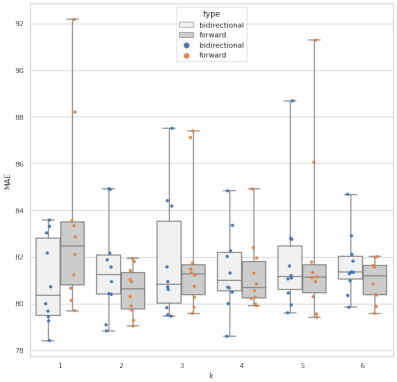
Overall, it can be summarized that the choice of the k value has a large impact on the performance of the model and differs depending on the task. The selected method (bidirectional or forward) does not have such a large impact on the performance. On average, over all tasks, $k = 2$ performs best with the bidirectional method. We use this configuration as the hyperparameter value for the GTC model in all evaluations inside this thesis.



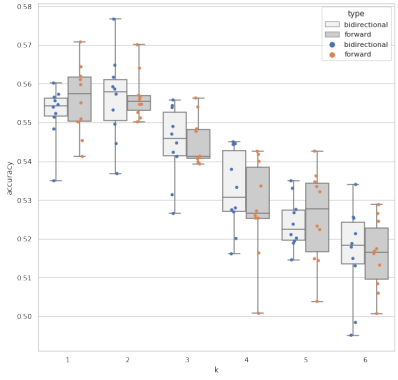
(a) Road Label Classification



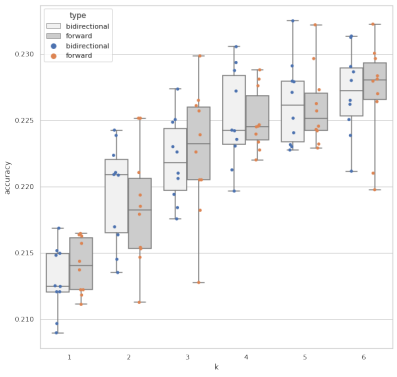
(b) Mean Speed Prediction



(c) Travel Time Estimation



(d) Next Location Prediction



(e) Destination Prediction

Figure 6.15: Results of the k-evaluation for different ITS tasks. The blue dots show the results for the bidirectional variant and the orange dots for the forward method. Each dot marks the performance of a single seed.

6.6 Trajectory Feature Analysis

In the following section, we will investigate the impact of features derived from trajectories on the performance of different ITS tasks. For this purpose, the average speed and the total utilization for each road segment were extracted based on aggregated trajectory information (see section 5.1.2 for details of the extraction process) and added as a feature in the learning process of the GTC model. Four variants of the GTC model are examined, as described in the following:

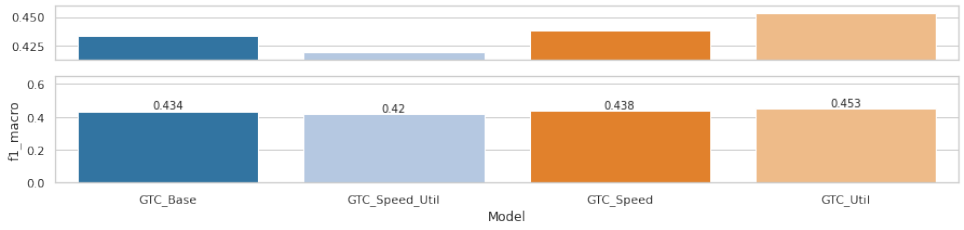
1. **GTC-Base**: the base variant with the basic features used in the main evaluation
2. **GTC-Speed**: a variant where we add the average speed for each road segment as a feature
3. **GTC-Util**: a variant where we add total utilization for each road segment as a feature
4. **GTC-Speed-Util**: a variant where we add the average speed and the total utilization for each road segment as features

The additional features for each variant are normalized and integrated into the feature matrix used for the training of the GTC model. Each variant is trained for 1000 epochs with the same base features and random seed. The training and evaluation are conducted on the Porto dataset. We test all model variants on all tasks, except for the mean speed task, since the prediction target is already included in the variants with the speed feature. It makes little sense to evaluate a variant where the dependent variable is already present in the training data. Accordingly, in the road label prediction task, the road label is removed as a feature in the training.

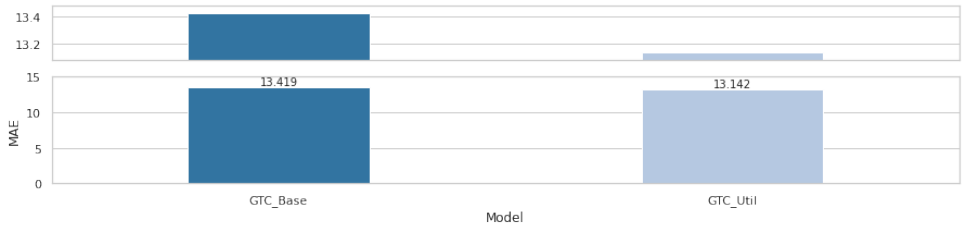
Figure 6.16 shows the results of the trajectory feature evaluation for all tasks. The performance of each variant is shown by a colored bar in the plots. Figure 6.16a shows the performance of the variants on the road label classification task. It can be seen that the addition of mean speed or utilization as a feature increased the performance in comparison to the base model. However, the utilization feature increased the performance significantly more than the mean speed feature. On the other hand, adding both features slightly decreased the performance of the model. Similar observations can be made on the mean speed prediction task (see figure 6.16b), where the utilization feature considerably improved the performance of the model as well.

Different insights can be gained from the results of the trajectory-based tasks. In the travel time task (see figure 6.16c), the trajectory features improved the score of the model only slightly, with the average speed having a more positive impact than the utilization. In the next location and destination prediction task (see figure 6.16d and 6.16e), the utilization feature has noticeably increased the performance on both tasks. It is remarkable that in the destination task, the performance is even slightly worsened by the average speed feature. We assume that this is because the average speed hardly correlates with the prediction of movement patterns. The same phenomenon can be observed in the next location task. Here, the performance of the model is only minimally improved by the aforementioned feature, which can certainly be attributed to fluctuations in the prediction.

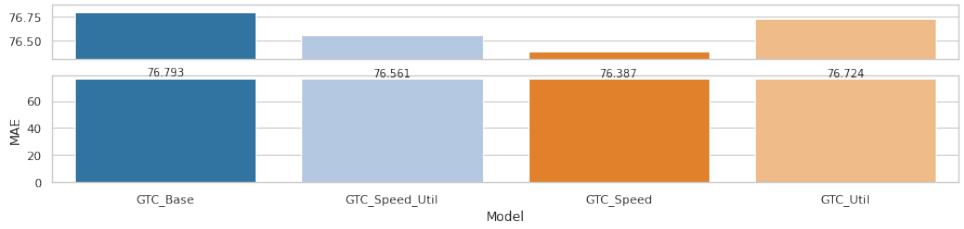
Overall, it can be concluded from the study that features from trajectories do have positive effects on the performance of the models for different tasks. It should be noted, however, that the impact on performance depends strongly on the quality and quantity of available data. More trajectory data approximate the behavior of the road network much better, and therefore features of better quality can be derived accordingly. Conversely, too little data leads to poor-quality features.



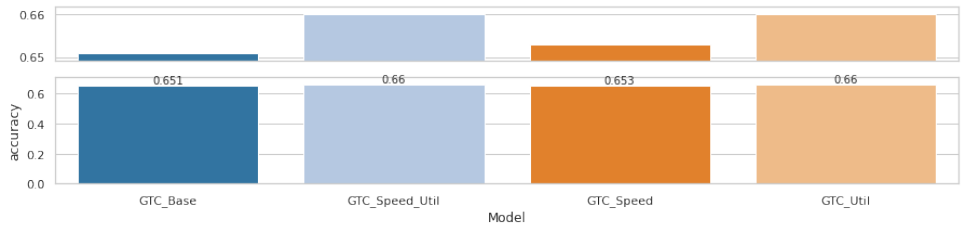
(a) Road Label Classification



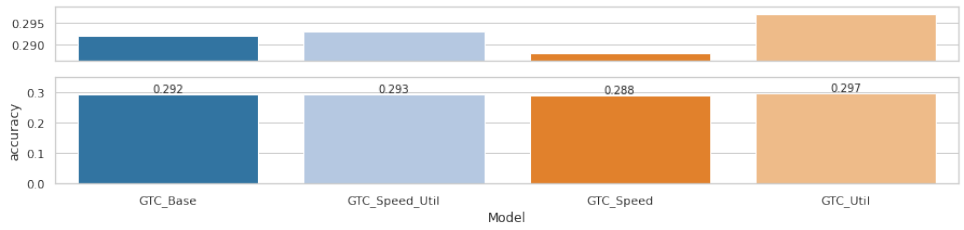
(b) Mean Speed Prediction



(c) Travel Time Estimation



(d) Next Location Prediction



(e) Destination Prediction

Figure 6.16: Results of the trajectory feature impact study for different ITS tasks.

6.7 Embedding Analysis

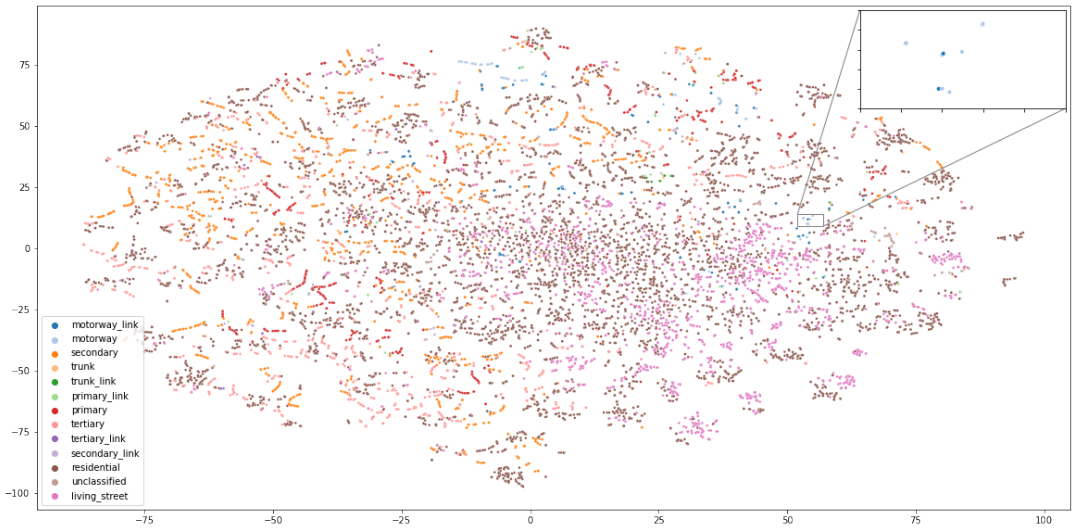
This section presents a visualization of the learned road segment embeddings generated by the GTN model for the Porto dataset, along with selected competitor models, namely GAE with an GCN encoder and Node2Vec. The visualization shows how the model positions the embeddings of the road segments in the vector space and can give an understandable intuition for how the model assigns road segments.

Since the embeddings consist of at least 128 dimensions for each road segment, and it is not possible to visualize such high dimensions for the human eye, we reduce the dimensions. There are several approaches to dimension reduction, both linear and non-linear [111, 112]. We use the T-Distributed Stochastic Neighbor Embedding (T-SNE) algorithm [113] to map the embeddings into the two-dimensional space. This method is non-linear and generates a 2D plane of the data while preserving the relative distance of the observations.

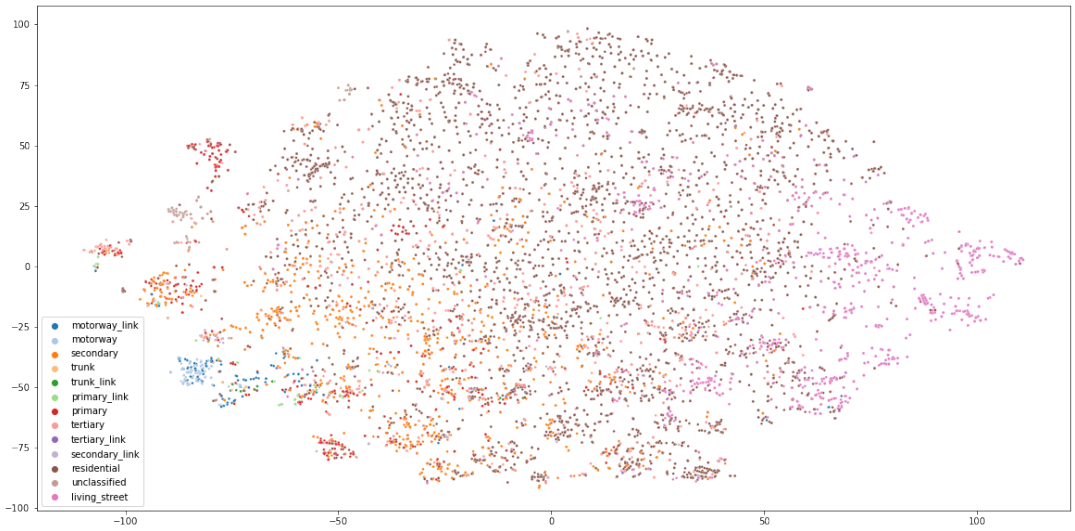
First, we visualize the road segment embeddings in dependence on the road type categories to analyze if similar road categories are close to each other in the embedding space. The embeddings are generated from model variants that did not include the road category in the training process. Figure 6.17 shows such a visualization for the GTN, GAE and Node2Vec model, where dots represent road segments and the color marks the corresponding road type category.

The clusters in the visualization of the models differ strongly in their structure. Especially, the embedding of the Node2Vec model arranges similar road segments in line-shaped structures. In contrast, the road segments in the GTN and GAE models are more scattered and groups are more recognizable as round clusters. However, the GTN model also has line-shaped structures in comparison to the GAE model. At first glance, the Node2Vec model seems to generate more differentiated clusters based on the road category. This is due to the walk-based approach, which results in more differentiable groups of road segments. The model captures well-defined structures such as city districts, while the roads of a district are clustered together.

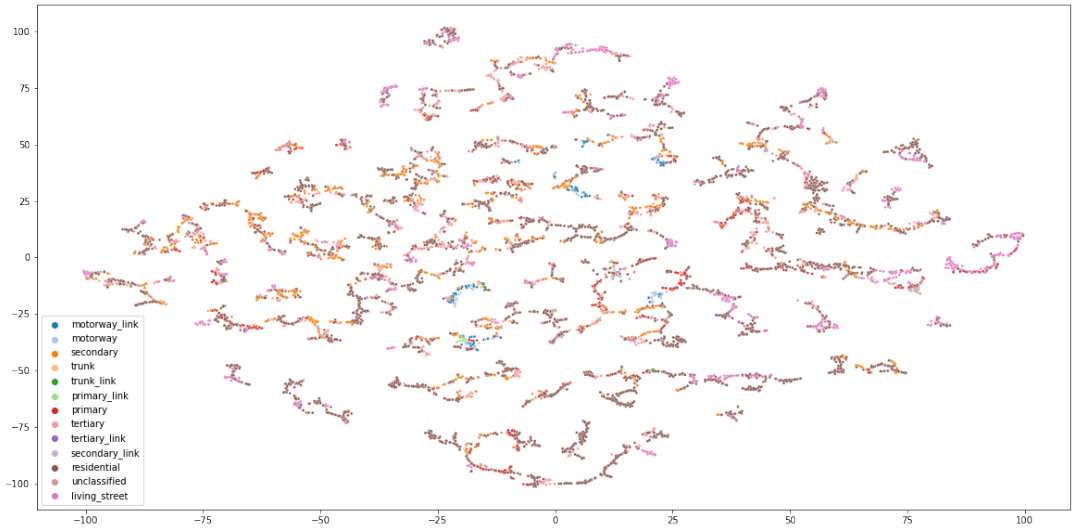
Furthermore, it is interesting that the GTN model clusters different road categories together. For example, motorways and motorway links are often clustered together, which are also close to each other in the real road network. The same is valid for living and residential streets (pink and brown color respectively), which often appear close to each other in the road network and are placed very close to each other in the embedding space of the GTN model. In the GAE embedding, they are rather clustered from each other. Many living streets are located in the right part of the visualization, and residential streets are more recognizable as single clusters. It is also interesting to note that the GTN model displays secondary and primary streets as linear structures, in contrast to the GAE model, which displays them in an unsorted manner. The linear arrangement is more realistic since primary and secondary streets have a low node degree and are therefore not as connected as residential or living streets. This also becomes clear in the analysis and visualization in chapter 6.2, where residential and living streets are most highly connected to each other, while secondary and primary streets are connected to other road categories via secondary or primary linking segments (see table C.5 for an explanation of linking tags). Those linking road segments occur less frequently, which results in a low connection degree for primary, secondary, and motorway road segments.



(a) GTN



(b) GAE with GCN encoder



(c) Node2Vec

Figure 6.17: Visualization of the road segment embeddings generated by the (a) GTN, (b) GAE, and (c) Node2Vec model. The colors mark the various road type categories denoted in the legend.

Another interesting aspect is whether the generated embedding has learned structures induced by trajectories. The previous figures show that structures are learned regarding road categories and geographic proximity of road segments. Using trajectories, it can be investigated whether the behavior and the structure induced by it are learned additionally. Figure 6.18 shows the embedding space generated by the GTN model, where we randomly took a trajectory from the dataset and color-coded the road segments that occur in the trajectory in the embedding space according to their road segment category. The trajectory consists of 38 distinct road segments with road categories including primary, tertiary, and primary_link types.

The figure shows that the embedded nodes within the trajectory are very close to each other in the vector space. The structure of the trajectory and the resulting arrangement of the road type categories can be recognized almost identically in the embedding space. On the left side, the trajectory starts with primary roads, goes over to tertiary, then primary again, and finally a primary_link segment. The primary_link is separated from the other road segments, but relative to the whole vector space, it is very close.

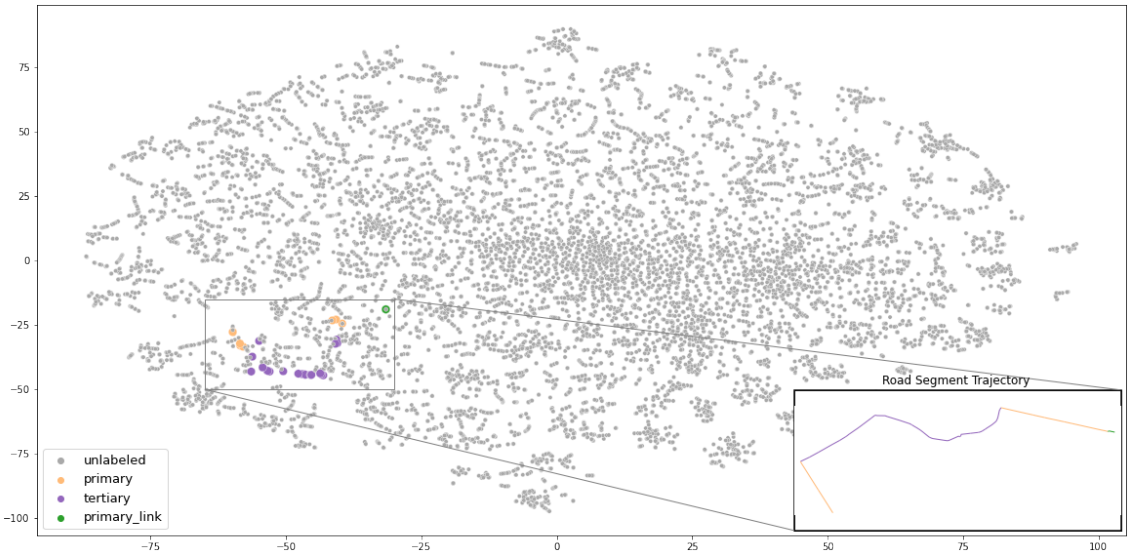


Figure 6.18: Illustration of a random trajectory inside the embedding space generated by the GTN model trained on the Porto dataset. The colored points indicate the road segments on the trajectory, and the image in the right corner shows the complete trajectory as a connected sequence of road segments within the road network. Each road segment in the point plot as well as in the trajectory plot is colored according to the road type category.

Overall, the embedding analysis showed that the GTN model can learn trajectory structures and the underlying user behavior of the road network in a representative way. Furthermore, connected streets in the road network are close in the embedding space, which indicates that the model also learned the general structure of the road network. It seems to combine representation techniques from the walk and convolution-based methods, which gives the model more power in terms of generalization to different tasks.

Chapter 7

Temporal Evaluation

In chapter 5.2.4, we proposed the T-GTC model, which incorporates the temporal dimension into the training and representation generation. Since this model needs a different kind of data and preprocessing, we do an additional evaluation separate from the main evaluation. A brief description of the evaluation setup is followed by the results of the evaluation.

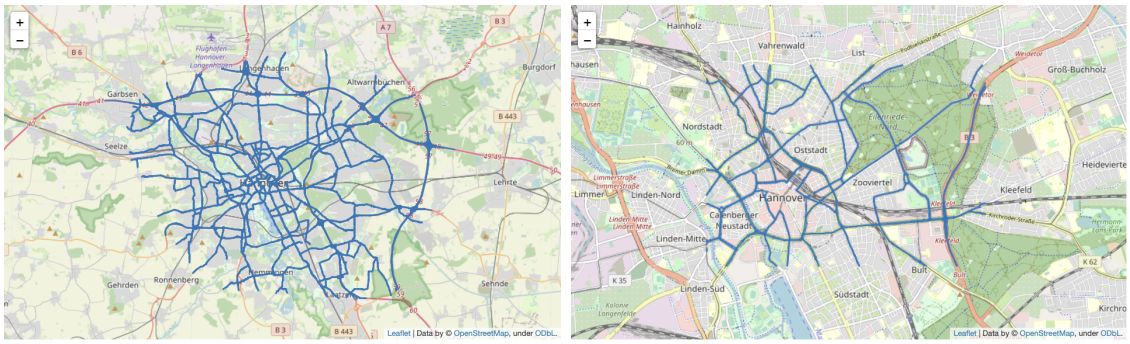
7.1 Evaluation Setup

As in the main evaluation, this section describes the setup for the experiments regarding the T-GTC model. First, an overview of the used datasets and their properties is given. This is followed by a condensed explanation of which baselines, metrics, tasks, and hyperparameters are used in the evaluation.

7.1.1 Datasets

To train and evaluate the T-GTC model, we use three different data sources. Specifically, data from the road network of Hanover, corresponding speed measurement data from sensors, and cab trajectory data are used. All datasets are provided by the L3S research center [86] and are not publicly available.

The road network dataset contains only high-order road segments, i.e., primary, secondary, trunk, tertiary, and corresponding linking segments (see table C.5 for an explanation of the types). The feature matrix consists of the segment's length, speed limit, and highway type. The feature columns are complete and no further preprocessing except for normalization is applied. Furthermore, we downscale the road network to a manageable size, since the weight matrices in our model scale with the number of road segments and the considered timeframe. For each additional road segment, the input, and weight matrices are extended by $|T| \times |F|$ rows, where $|T|$ is the number of time steps and $|F|$ is the number of features. This makes it computationally infeasible with too many road segments. To make the network smaller and more feasible, we reduce the rectangle area of the original network by 80%. All road segments outside the down-scaled rectangle are removed, and the intersecting outgoing segments are truncated after the intersection point.



(a) Original road network with 8620 segments (b) Reduced road network with 1366 segments

Figure 7.1: Original (a) and reduced (b) road network of Hanover. The reduced network is used in the training and evaluation process.

Using this method, the road segments inside the considered road network could be reduced from 8620 to 1366. Figure 7.1 shows the original and the reduced network. The speed measurements are collected over a timeframe of around four months from 15/08/2019 to 01/12/2019. Each measurement is already mapped to a corresponding road segment in the road network. The measurements have a time interval of 15 minutes and are filled with zeros on missing entries. The trajectory dataset is the same as described in section 6.1.1, but the trajectories are mapped (for details on the mapping process, see section 5.1.2) to the reduced road network. The mapping resulted in 22409 trajectories for the evaluation. It should be noted that the speed measurement data correlates with the trajectory data regarding the observation period.

7.1.2 Baselines and Tasks

The goal is to evaluate if the T-GTC model learns structural and temporal properties simultaneously and if decoder models can utilize this extended knowledge to reach better performance on various tasks. Regarding the tasks, we evaluate on road label classification, travel time estimation, next location prediction, and destination prediction. The road label classification, next location, and destination prediction tasks can measure how accurately the embeddings incorporate the static structure of the road network, while the travel time prediction task measure structural as well as temporal properties. The travel time for a route can be extremely volatile regarding the temporal dependence, because of dynamic events in the road network like congestions. The hyperparameter setup and architecture of the task-dependent decoder models are the same as described in section 6.1.2. Regarding the temporal models, the road segment embeddings for the trajectory-based task decoder models are generated dynamically. Specifically, given a trajectory, the road segment embeddings are generated for a road network sequence in the same timeframe as the considered trajectory. For the road label classification task, the corresponding feature column is removed in the training process of the models to prevent leakage. The competitor baselines are chosen based on the underlying technique they use. We compare the T-GTC model against convolution, walk, and feature-based methods. Specifically, we compare against a GAE model with a GCN encoder and as a second variant with a GAT encoder, Node2Vec, DeepWalk, and a PCA model.

Furthermore, we compare against a SOTA temporal baseline, namely Temporal Graph Convolutional Network (T-GCN) [78], which combines a Gated Recurrent Unit (GRU) with a GCN model. In our evaluation, the T-GCN model has the same training objective as the T-GTC model, namely graph and sequence reconstruction. The hyperparameter setup for the baseline models is the same as described in section 6.1.3, except for the reduced output dimension k of the PCA model, which is set to $k = 2$ in this evaluation. This value is selected because the road network under consideration has only three static features.

7.1.3 Hyperparameter Setup

This section gives a brief introduction to the hyperparameter setup for the T-GTC model and its variants. As mentioned before, the hyperparameters for the baseline models are the same as described in section 6.1.3. However, for our models and the T-GCN baseline, we tuned the learning rate using a learning rate finder [114], which does small runs and increases the learning rate while logging the loss. Afterwards, the finder picks the optimal learning rate regarding the loss. Table 7.1 shows the used learning rates for each variant. As an optimizer, we use Adam [102]. The embedding dimension is the same as for the baseline models, namely 128. For the T-GTC model and its variant, the underlying GTC model uses the bidirectional variant with $k = 2$ as a hyperparameter for the trajectory aggregation length. In this setting, it is furthermore possible to train the temporal models using early stopping to counteract overfitting [115] since the driving speed measurements can be divided into a train and validation set. We use the last 10% of the driving speed measurement data as the validation set and the remaining 90% as the training set. The resulting training epochs can be seen in table 7.1. Because of computational limitations, each variant was trained with a batch size of 16.

Model	learning_rate	trained epochs
T-GCN	0.001096	21
T-GTC	0.000912	19
T-GTC (Attention)	0.00871	9

Table 7.1: Learning rates and trained epochs for the temporal models.

7.2 Experimental Results

In the following, we will present and describe the results of the temporal evaluation. Table 7.2 shows the evaluation results of our proposed models against the competitor baselines on all tasks under consideration. On the road label classification task, our T-GTC attention model has the best overall performance regarding the F1 macro score, followed by convolutional-based non-temporal models, namely GAE with its different encoders. Our base T-GTC variant performs worse than the T-GCN baseline and the convolutional models. The worst performers are walk-based methods, namely Node2Vec and DeepWalk, followed by the PCA model.

Interesting results can be recognized on the travel time task. Regarding the performance, our attention-based T-GTC model performed superior, followed by the standard T-GTC

model and the T-GCN model. Models that do not incorporate the temporal dimension into the learning and inference process performed a lot worse. Specifically, the best non-temporal model is Node2Vec with around 7.3% and 6.9% performance decrease in MAE and RMSE, respectively, in contrast to our T-GTC attention model. The Node2Vec model is followed by the PCA model, which has another noticeable decrease in performance. The DeepWalk model and convolutional methods performed the worst on this task.

In the next location task, which evaluates the structural learning power of the models, a walk-based method, namely Node2Vec, performed best. However, the performance of this model is closely followed by our base T-GTC model, with a decrease of 1.5% in accuracy performance. The T-GTC is closely followed by the DeepWalk model. Our proposed attention variant did not perform as well as the base variant on this task, but also not much worse. The T-GCN and our attention variant performed relatively similar, followed by convolutional methods and the PCA model. From the models without a temporal consideration, the PCA model performed worst, followed by convolutional methods.

	Road Label Classification		**Travel Time Estimation**	
	F1 Macro	F1 Weighted	MAE	RMSE
GAE (GCN)	0.322	0.479	86.684	135.768
GAE (GAT)	0.351	0.534	93.444	138.160
Node2Vec	0.217	0.521	81.545	138.029
DeepWalk	0.239	0.539	87.365	153.315
PCA	0.146	0.458	84.417	136.899
T-GCN	0.337	**0.653**	79.359	136.963
T-GTC	0.298	0.572	79.22	130.07
T-GTC (Attention)	**0.405**	0.573	**75.579**	**128.480**
	Next Location Prediction		**Destination Prediction**	
	Accuracy		Accuracy	
GAE (GCN)	0.606		0.353	
GAE (GAT)	0.551		0.355	
Node2Vec	**0.665**		0.400	
DeepWalk	0.650		0.398	
PCA	0.139		0.243	
T-GCN	0.621		0.393	
T-GTC	0.655		**0.405**	
T-GTC (Attention)	0.617		0.387	

Table 7.2: Evaluation results for the T-GTC model, its variants, and the competitor baselines on all tasks under consideration.

Similar observations can be made on the destination prediction task. On this task, our base T-GTC model performed slightly better than the Node2Vec model, with around 1.2% performance difference. This is followed by DeepWalk and the T-GCN model. Our attention variant follows closely after the T-GCN model. As in the next location task, the PCA model performed worst, followed by convolutional methods.

As in the main evaluation, we calculated the generalization scores following formula 6.9 to get a better overview of the general performance difference between the evaluated models. Table 7.3 shows the resulting generalization scores. The overall best performance on this metric is achieved by our attention-enhanced T-GTC model. This is followed by the other temporal models, namely T-GCN and the T-GTC base variant, with a relatively large performance gap. In general, the walk-based methods perform better than convolutional methods, while the PCA model performs worst by far.

	Generalization Score				
	Road	Travel Time	Next Location	Destination	**Final Score**
GAE (GCN)	0.679	0.378	0.888	0.677	0.655
GAE (GAT)	0.791	0.000	0.784	0.693	0.567
Node2Vec	0.275	0.666	**1.000**	0.970	0.728
DeepWalk	0.360	0.340	0.972	0.955	0.657
PCA	0.000	0.505	0.000	0.000	0.126
T-GCN	0.735	0.788	0.918	0.928	0.842
T-GTC	0.588	0.796	0.981	**1.000**	0.841
T-GTC (Attention)	**1.000**	**1.000**	0.909	0.887	**0.949**

Table 7.3: Generalization score for our proposed temporal models and all baseline models. The best score for each task is marked in bold. The last column on the right shows the final score calculated as the average of all task-specific scores for the corresponding model.

Overall, the evaluation of the temporal models showed that incorporating the temporal dynamics of a road network can significantly improve the general power and performance on various ITS tasks. Specifically, on the travel time task, which is the only task that heavily depends on temporal properties, the temporal models outperformed static models significantly. We could further outperform a current SOTA model, namely T-GCN, with our proposed attention-enhanced T-GTC model. The attention mechanism in combination with a LSTM model seems to better capture temporal dependencies of the road network in contrast to only using a LSTM or GRU model.

Chapter 8

Discussion

In this chapter, we discuss the results of the evaluations regarding the performance of our proposed models in detail and address possible limitations of this thesis.

Starting with the main evaluation in chapter 6.3, we were able to develop a model that outperformed both standard graph procedures and SOTA road network representation learning models over five different ITS tasks. In particular, on tasks, such as road label and mean speed prediction, that predicts static road segment properties, our model has shown superior performance relative to all other models across three different datasets. This can be attributed to the fact that we combined the learned representations of the GTC and TSD models. We were able to show in section 6.4.2 of the ablation that the combination of the two embeddings produced a significant performance boost in contrast to the embeddings of the corresponding single models on all tasks, but especially on road label and mean speed prediction. This performance boost can be attributed to the fact that the respective decoder model can decide for itself which information from the embedding is relevant for the downstream task. The walk-based embedding of the TSD model provides information about structural properties, both locally and globally. Furthermore, the embedding of the GTC model contains, in addition to structural properties, also feature-based information. By aggregating both embeddings, we could create a representation that captures structural properties, enriched with feature-based properties. We visualized this idea in section 6.7 in addition to our GTN model for a walk-based and a convolution-based method, namely GAE with a GCN encoder and Node2Vec. It was visible that the combined representation of the GTN model captures properties of both underlying embeddings, such as road type clusters from the convolutional method, and structural properties in the form of line clusters from the walk-based method. However, it must be noted that the visualized walk and convolution-based embeddings are not directly from the TSD and GTC models, but from similar base variants which use the same underlying technique to learn the road segment representations. Besides our proposed model, for static downstream tasks, we could determine that in general convolution-based methods work better. They directly integrate features of road segments into the embeddings, where some of them, such as the speed limit, correlate strongly with the target variables. For example, a road segment with a speed limit of 50 km/h will most likely have a corresponding average driving speed.

Regarding the trajectory-based task, such as travel time, next location, and destination prediction, our model achieved superior performance on the destination and travel time tasks over all datasets. That can be attributed to the direct integration of trajectory data. Our model can integrate underlying traveling semantics and travel flows directly into the road segment representations, which is expressed by the superior performance on this task compared to other models. We were able to confirm this claim visually in figure 6.18, where it is visible that the embedding generated by the GTN model captures the spatial properties of the sample trajectory. This mostly does not hold for every single trajectory, since the embedding holds multi-facet properties of the road network, but it shows that our model incorporates trajectory semantics directly into the road segment representations. Another fundamental aspect that is shown through the results on these tasks is that the incorporation of trajectories into the learning process of road segment embeddings has a highly positive impact on performance. This is furthermore evident in the ablation study, specifically in section 6.4.2, where our trajectory-enhanced models, namely TSD and GTC, performed overall better on the task than similar standard methods.

The only task where our model did not outperform the competitor models is the next location task. However, it did not underperform, since it was the second-best model on the Porto and San Francisco dataset. On these datasets, our model performed better than standard graph models, such as GAE, Node2Vec, and DeepWalk, and furthermore outperformed most of the competitor SOTA models. We argue that this task can be solved in two different ways. Specifically, if the embedding captures the local structure of the network or if the embedding learned the local trajectory flows. We further argue that the best models on this task, such as RFN and HRNR have a high focus on learning the local structural properties of the road network. Since the average node degree is extremely low in road networks (see table 6.1 for reference values) and therefore the corresponding choice of possible next locations, this task is easier to solve when the embedding captures local structures.

However, we were able to show using the generalization score that our model has superior generalization over the tasks and datasets compared to the competitor models, which most of the time only work well on a subset of the tasks. Additionally, we determined in section 6.4.1 that the combination of using TSD and GTC as pre-initialization embeddings for the GTN transformer resulted in the best performance in average over all tasks in comparison to other possible variants, which further empowers our claim to incorporate traveling semantics through trajectories into the learning process.

Besides the main model, we were able to show in section 6.6 that simple features derived from trajectory data, such as absolute utilization or mean driving speed on the road segments, can significantly boost the performance of convolutional-based methods on all evaluated task. This further shows that trajectory data is a crucial part of learning robust representations of road segments and can be used in multiple ways to learn better representations.

Additionally, it should be noted that the results from the main evaluation are not obtained by cherry-picking lucky seeds [116]. The results are averaged over five different seeds corresponding to five different train-test splits of the trajectory datasets.

Hence, the shown results are very robust and reliable. Furthermore, we provide a reproducibility guide in B, which comprehensively explains how the results from the evaluation can be reproduced inside our codebase.

Regarding our proposed temporal models, namely T-GTC and its attention-enhanced variant, we showed that the incorporation of temporal dynamic properties into the learning process significantly improves the performance on temporal-dependent tasks, while keeping great performance on highly structural dependent tasks. Our attention-enhanced T-GTC outperformed static models and the SOTA T-GCN temporal model. Additionally, it reached a superior generalization score over the evaluated tasks. We argue that incorporating the temporal dependencies is crucial to learn representative embeddings and that this approach holds a lot of potential for further investigations. Specifically, the design of temporal aware loss functions regarding trajectories could further improve the learned representations of temporal models. We provide a concrete idea in the future work part of the conclusion in the next chapter.

Limitations

This work has been done to the best ability with the resources available. Nevertheless, there are possible limitations in this work, which are explained in the following.

A possible limitation of this thesis is the underlying trajectory data. As described in section 5.1.2, we invested much effort into preprocessing the trajectory data. However, we could not check every single trajectory for reasonableness and correctness. Hence, it is indeed possible that there are noisy samples inside the generated dataset. This possible noise is further propagated into the label and train data generation of the mean speed and the three trajectory-based tasks, which could have an impact on the results. However, since all models are evaluated on the same datasets, it has no relevant impact on the comparability of the models performances. We further argue that it makes sense to have noisy samples inside the datasets, since this is the most common situation when working with data captured by noisy sensors, such as GPS trackers in our case.

Another limitation is the lack of model tuning in this thesis. Since we evaluated against many competitor models, it was insufficient to tune every model for the optimal hyperparameters. Specifically, for our own proposed model, namely the GTN model, the training is extremely time-intense. Therefore, we decided to use hyperparameter setups from related papers for standard models and the proposed hyperparameter setups by the authors for the road representation learning models. The disadvantage of this strategy is obviously that the performance of the models could be better under certain circumstances.

At last, it should be mentioned that we evaluated the impact of the trajectory features on the performance of the GTC model, but have not used the features in the main evaluation. Using the features in our model could potentially give a performance boost on the tasks, but we argue that it would not be fair in comparison to the other models, which have not used these features. Hence, all models that used road segment features in the training process were provided with the same set of features to make the results more comparable.

Chapter 9

Conclusion

This thesis addressed the problem of learning robust and generic representations of road segments, which are usable for various ITS downstream tasks. We investigated the potential behind integrating travel semantics, induced by trajectory data, directly into the representations. Specifically, we proposed three novel models, namely GTC, TSD, and GTN, which directly incorporate trajectory data into the learning process. We provided a comprehensive evaluation of the proposed models against SOTA models and common baselines from the literature on three different datasets and over five different ITS downstream tasks. Through the evaluation, we showed that our GTN model successfully incorporated travel semantics into the learned road segment representations, which resulted in superior performance and generalization against the competitor models. Specifically, our GTN model had a 57% higher generalization score over the ITS downstream tasks than the second-best-performing model. We conducted extensive ablation studies, which showed that our GTN model outperformed different variations of itself with various components. Additionally, we evaluated the underlying base models, namely GTC and TSD against models with similar underlying techniques, and showed that our proposed models outperform them on the ITS downstream tasks.

Furthermore, we showed that additionally incorporating temporal dynamics into the representation learning of road segments can improve the performance on ITS downstream tasks that are prone to temporal aspects, such as travel time prediction, without decreasing the performance on tasks that are mostly focused on structural properties of road networks, such as road label prediction. To enable the induction of temporal dynamics into the road segment representations, we proposed an additional model based on the GTC architecture, namely T-GTC. Additionally, we proposed an attention-enhanced variant of the T-GTC model. The evaluation showed that the attention-enhanced variant had the overall best performance in comparison to all other competitor models, with a generalization performance improvement of 30.4% compared to the best non-temporal performing model.

In summary, we were able to create different models, which incorporate underlying road network data, such as trajectory or temporal data, and showed that the incorporating of these data can significantly improve the performance on various ITS downstream tasks.

The field of representation learning of road networks still holds a lot of research potential, especially when it comes to the incorporation of related data, such as trajectory or temporal data. This thesis revealed many aspects that can be further developed. In future work, additionally to road segments, it could be beneficial to include road intersections into the learning process. Since they hold information about road elements, such as traffic lights, it could further improve the performance on ITS downstream tasks, especially on tasks that have a temporal dependence.

Furthermore, our GTN model can theoretically generate route representations through the transformer module. There could be potential in using these representations for different route-based downstream tasks, such as destination or travel time prediction.

The thesis showed that combining walk-based and convolution-based embeddings can improve the general performance on a variety of downstream tasks. It could be possible to design a model that simultaneously uses walk and convolution-based methods while learning road segment representations. Theoretically, this could generate an embedding that directly combines the advantages of both techniques.

Another possible improvement could be in the design of the loss functions. In this thesis, we mostly used reconstruction losses. The general performance could be further improved with more tailored loss functions. For example, in the case of the T-GTC model, it could improve the performance by introducing an additional self-supervised loss similar to the one proposed for the GTN model. Specifically, the decoder could reconstruct trajectories to learn the respective structure and induce it into the representations. The training trajectories could be mapped to the current training graph sequence regarding the observation period. This would enable the model to learn trajectory behavior dependent on time. Another approach, in this direction, could be to design and incorporate road network specific contrastive loss functions [117].

Further inspiration for future work can be found in chapter A, where we briefly describe important approaches that did work in the case of this thesis.

Appendices

Appendix A

Remarks on Efforts that did not Work

This chapter explains some efforts, which we evaluated during the creation of this thesis that did not work as excepted, but hold potential for further investigations.

Task: Route Reconstruction

Initially, we wanted to evaluate our models on six different tasks. The sixth task was the route reconstruction, where the first and last road segments are used as input and the part that connects the first and last road segment of the trajectory with r segments is the prediction target. This task is the most demanding since a variable set of road segments has to be predicted. However, the performance on this task was destitute regarding all models. Because of the poor performance, we decided to remove this task from the evaluation. We argue that this task is too complex without enough personalized trajectory data since the possible sequences of road segments between a start and end road segment is too large. The implementation can still be found inside the source code.

Model: GTN Dynamic Embeddings

Additionally, to static embedding learning, our developed GTN model can dynamically generate context-aware route representations through the transformer module. Doing a forward pass, given a sequence of road segments, generates an embedding for the road segment sequence. We tested the performance of these embeddings on different trajectory tasks but did not make considerable changes to the decoder models. Hence, the performance was not better than using the static embeddings, but even worse. It could be beneficial to evaluate the dynamic embeddings with better decoder models to eventually increase the downstream task performance. However, the goal of this thesis was to generate static road segment embeddings, and therefore we did not follow up on this possibility.

Model: T-GTC Architecture Variants

In the developing process of the T-GTC model, we evaluated numerous model architectures. In the following, we give a brief description of what we tried out.

We tried a lot of attention mechanisms after the first LSTM layer to obtain a better global aware representation of the sequence. Specifically, we implemented the mechanisms from [118, 119, 120] and tested them in various settings. This included different numbers of LSTM layers, norm layers, and dense layers. Furthermore, we implemented a similar architecture as proposed in [121] with gated fusion layers that combine the TSD embedding with the temporal embedding. However, these approaches could not outperform the base model with a single LSTM layer. This could be the case because we learn in an unsupervised setting, while the proposed architectures all worked in a supervised environment.

Appendix B

Reproducibility

This chapter explains how the results obtained in this thesis can be reproduced. Since the project got relatively big regarding files and directories, we begin by explaining the overall project structure. The following visualizes the simplified project structure as a directory tree.

```
root
├── datasets
│   ├── hanover
│   ├── porto
│   └── sf
├── evaluation
│   ├── tasks
│   │   └── ...definition of tasks
│   ├── evaluate.sh
│   ├── evaluate_temporal.sh
│   └── explanatory_data_analysis.ipynb
├── experiments
├── generator
├── models
│   ├── training
│   │   └── ...training scripts
│   ├── model_states
│   └── ...definition of models
└── environment.yml
```

The evaluation directory contains all source code related to evaluating the models from chapter 6.3 and chapter 7.2. This includes, among other things, the definition of the different tasks, which can be found in the *tasks* directory, and the evaluation scripts. We will explain the usage of the scripts later in this chapter. The *experiments* directory contains notebooks, which show how to preprocess the trajectory datasets and how to generate the corresponding road network graphs. The *generator* directory contains source code for the underlying preprocessing steps and transformations. The last important directory is the *models* folder, where all the source code related to the models is placed.

This includes model definitions, original model code from papers, and the corresponding training scripts.

We provide a *conda* environment [122] for a simple recreation of our code environment. The environment can be installed with the following bash commands:

```
conda config --env --set channel_priority strict
conda env create -f environment.yml
conda activate road-emb
```

To reproduce the results, the reproduction scripts in the evaluation directory can be used. The *evaluate.sh* script reproduces the results from main evaluation from chapter 6.3 and can be used as follows:

```
sh evaluate.sh -m [model_type] -t [task] -p [output_path] -d [gpu_id]
-e [epochs] -b [batch_size] -l [learning_rate] -s [seeds] -c [city]
```

The script for the temporal evaluation works the same way, with slightly different arguments:

```
sh evaluate_temporal.sh -m [model_type] -t [task] -p [output_path]
-d [gpu_id] -e [epochs] -b [batch_size] -l [learning_rate]
```

The output of the scripts is a CSV file for each task, containing the results for each evaluated model. The file is saved in the directory path given by the *-p* argument. For the main evaluation, we used the seeds 69,88,42,420, and 123. The corresponding train-test splits can be found in the *datasets* directory inside the corresponding city directory. All models were trained and evaluated using a single Nvidia GTX 1080 TI with 11 GB memory. The used hardware provided 128 GB of memory ram, and we recommend having 100 GB of ram when reproducing the results.

Appendix C

Additional Materials

Formatting of Trajectory Data

For our evaluation, several trajectory datasets were used. The raw trajectory data comes in different formats and has to be preprocessed to a unified standard to fit into the pipeline described in section 5.1.2. In the following, we show what the different formats look like and how they are unified to a single standard.

The Porto trajectory dataset is a single file containing one trajectory per row, which is further identified by a unique number called *trip_id*. Each row has a starting Unix timestamp, a field that indicates missing data, and a polyline column consisting of a GPS sequence describing the driven trajectory. Furthermore, it is known that there are 15 seconds between each GPS measurement. Table C.1 shows a sample row from the trajectory dataset.

trip_id	timestamp	missing_data	polyline
1372636858620000589	1372636858	False	[[-8.618643,41.141412],[-8.618499,41.141376],[...

Table C.1: Trajectory sample from the Porto dataset. Columns that are not relevant have been removed for conciseness.

Fundamentally different is the San Francisco dataset, which consists of several files. Each file describes the trajectories of a cab over the entire period under consideration. Each row in a file includes latitude, longitude, id, and a timestamp entry. Furthermore, the row includes an entry that indicates if a cab was occupied at the time of recording. Table C.2 shows a sample row of a single file from the trajectory dataset.

tax_id	timestamp	occupied	latitude	longitude
abdremlu	1213032899	1	37.78554	-122.42929

Table C.2: Trajectory sample from the San Francisco dataset. Columns that are not relevant have been removed for conciseness.

The Hanover trajectory dataset is a single file containing all measurements for each vehicle. Each row is labeled with a trajectory id, an ascending sequence number, a

timestamp, latitude, and longitude entry. Specifically, the file contains all trajectories, while a single trajectory is defined over multiple rows. Table C.3 shows two sample rows of a single file from the trajectory dataset.

id	seq	time	latitude	longitude
0000eb4fcec1fcf7af18cd8f6b4642ff	0	2019-08-16T16:53:23.023Z	52.332600	9.828800
0000eb4fcec1fcf7af18cd8f6b4642ff	1	2019-08-16T16:54:43.043Z	52.331356	9.819705

Table C.3: Trajectory sample from the Hanover dataset. Columns that are not relevant have been removed for conciseness.

As can be seen, the formats of the trajectory datasets are unique. Hence, we decided to map them to a single format. The format describes each trajectory in a single row, containing a unique id, a sequence of timestamps, and the corresponding sequence of GPS points as a polyline [123] element. The timestamps start at zero and measure the time difference between successive points. For the temporal evaluation using the Hanover dataset, start and end Unix timestamps are furthermore included. Table C.4 shows a sample row of the unified trajectory format.

id	timestamps	polyline	start_stamp	end_stamps
0	0, 80, 424, ...	LINESTRING (9.8288 52.3326, ...	1565974403	1565982018

Table C.4: Trajectory sample of the unified trajectory format. The columns start_stamp and end_stamp are only necessary for the temporal evaluation.

To transform the Porto dataset, we first remove all rows that are marked with missing data. Afterwards, the GPS sequence is extracted and converted to a polyline. Since it is known that between each measurement are 15 seconds time difference, a sequence of corresponding timestamps can be calculated as follows:

$$T = V * 15 \tag{C.1}$$

where T is the timestamp sequence and V is a vector containing ascending numbers starting at 0. The vector V has the same length as the GPS sequence.

The San Francisco dataset is transformed by extracting all rows of each file into a single table and grouping the rows by the tax_id. Afterwards, each group is further grouped by successive occupied trajectory measurement points, which results in a set of groups, where each group contains a single occupied trajectory. Furthermore, the timestamp, latitude, and longitude columns of each group are aggregated into sequences. Finally, the first timestamp is subtracted from each element in the timestamp sequence, and the GPS sequence is transformed into a polyline.

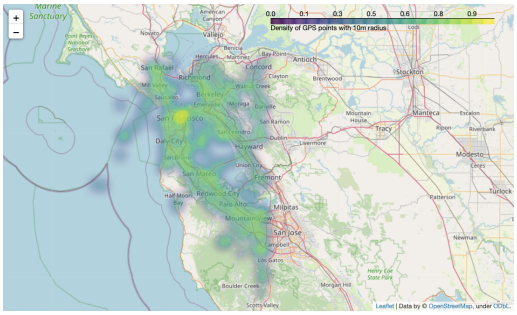
A similar procedure is applied to the Hanover dataset, where the rows are initially grouped by the trajectory id. Each group is sorted by the sequence column in ascending order. Afterwards, the latitude and longitude columns are aggregated into a sequence of coordinate tuples and converted to a polyline. The timestamps are also aggregated to an ordered sequence, and the first timestamp is subtracted from all elements in the sequence.

OpenStreetMap Highway Type Description

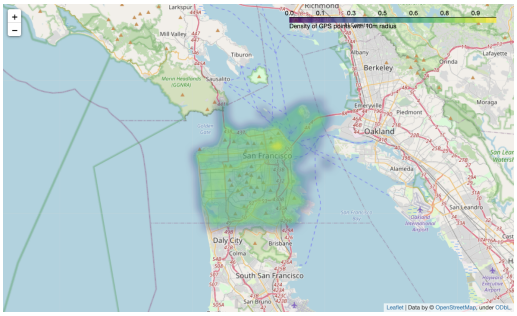
Highway Tag	Description
motorway	A restricted access major divided highway, normally with 2 or more running lanes plus emergency hard shoulder. Equivalent to the Freeway, Autobahn, etc..
trunk	The most important roads in a country's system that aren't motorways. (Need not necessarily be a divided highway.)
primary	The next most important roads in a country's system. (Often link larger towns.)
secondary	The next most important roads in a country's system. (Often link towns.)
tertiary	The next most important roads in a country's system. (Often link smaller towns and villages)
unclassified	The least important through roads in a country's system i.e. minor roads of a lower classification than tertiary, but which serve a purpose other than access to properties. (Often link villages and hamlets.)
residental	Roads which serve as an access to housing, without function of connecting settlements. Often lined with housing.
motorway_link	The link roads (sliproads/ramps) leading to/from a motorway from/to a motorway or lower class highway.
trunk_link	The link roads (sliproads/ramps) leading to/from a trunk road from/to a trunk road or lower class highway.
primary_link	The link roads (sliproads/ramps) leading to/from a primary road from/to a primary road or lower class highway.
secondary_link	The link roads (sliproads/ramps) leading to/from a secondary road from/to a secondary road or lower class highway.
teritiary_Link	The link roads (sliproads/ramps) leading to/from a tertiary road from/to a tertiary road or lower class highway.
living_street	For living streets, which are residential streets where pedestrians have legal priority over cars, speeds are kept very low and where children are allowed to play on the street.
road	A road/way/street/motorway/etc. of unknown type. It can stand for anything ranging from a footpath to a motorway
busway	A dedicated roadway for bus rapid transit systems

Table C.5: Description of the OSM highway tags [124].

GPS Clipping Figures for San Francisco and Hanover

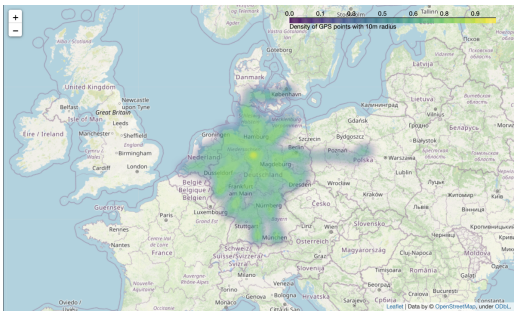


(a) GPS points before clipping

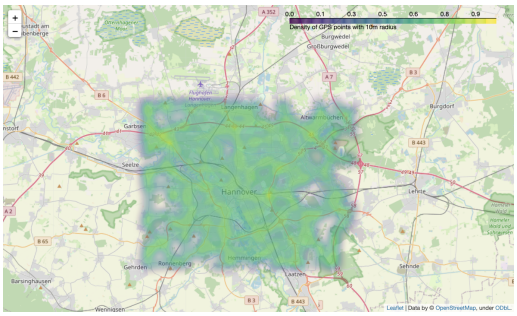


(b) GPS points after clipping

Figure C.1: Example of trajectory GPS points clipping on the San Francisco trajectory dataset. The heatmap describes the density of GPS points within 10 meters, where bluish areas are fewer GPS points and yellow in proportion many (20% of the GPS points are used for the plots). The bright yellow area in (a) and (b) is the center of San Francisco.



(a) GPS points before clipping



(b) GPS points after clipping

Figure C.2: Example of trajectory GPS points clipping on the Hanover trajectory dataset. The heatmap describes the density of GPS points within 10 meters, where bluish areas are fewer GPS points and yellow in proportion many (10% of the GPS points are used for the plots). The yellow circle in part (a) is the city of Hanover.

References

[1] L. Figueiredo, I. Jesus, J. Tenreiro Machado, R. Ferreira, and J. Carvalho, "Towards the development of intelligent transportation systems," pp. 1206 – 1211, 02 2001.

[2] Y. Jing, H. Wang, K. Shao, X. Huo, and Y. Zhang, "Unsupervised graph representation learning with variable heat kernel," *IEEE Access*, vol. PP, pp. 1–1, 01 2020.

[3] K. Oono and T. Suzuki, "Graph neural networks exponentially lose expressive power for node classification," 2019.

[4] Y. Chen, X. Li, G. Cong, Z. Bao, C. Long, Y. Liu, A. K. Chandran, and R. Ellison, *Robust Road Network Representation Learning: When Traffic Patterns Meet Traveling Semantics*, pp. 211–220. New York, NY, USA: Association for Computing Machinery, 2021.

[5] M.-X. Wang, W.-C. Lee, T.-Y. Fu, and G. Yu, "On representation learning for road networks," *ACM Trans. Intell. Syst. Technol.*, vol. 12, dec 2020.

[6] N. Wu, X. W. Zhao, J. Wang, and D. Pan, "Learning effective road network representation with hierarchical graph neural networks," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '20, (New York, NY, USA), pp. 6–14, Association for Computing Machinery, 2020.

[7] T. S. Jepsen, C. S. Jensen, and T. D. Nielsen, "Relational fusion networks: Graph convolutional networks for road networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, pp. 418–429, jan 2022.

[8] F. Harary and R. Z. Norman, "Some properties of line digraphs," *Rendiconti del Circolo Matematico di Palermo*, vol. 9, pp. 161–168, 1960.

[9] R. L. Hemminger, "Line digraphs," in *Graph Theory and Applications* (Y. Alavi, D. R. Lick, and A. T. White, eds.), (Berlin, Heidelberg), pp. 149–163, Springer Berlin Heidelberg, 1972.

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

[11] P. Chao, Y. Xu, W. Hua, and X. Zhou, "A survey on map-matching algorithms," 2019.

[12] C. Yang and G. Gidofalvi, "Fast map matching, an algorithm integrating hidden markov model with precomputation," *International Journal of Geographical Information Science*, vol. 32, no. 3, pp. 547 – 570, 2018.

[13] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[14] M. Nixon and A. Aguado, *Feature extraction and image processing for computer vision*. Academic press, 2019.

[15] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 2, pp. 604–624, 2020.

[16] P. Cui, X. Wang, J. Pei, and W. Zhu, "A survey on network embedding," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 5, pp. 833–852, 2018.

[17] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: A survey," *IEEE transactions on Big Data*, vol. 6, no. 1, pp. 3–28, 2018.

[18] P. D. Hoff, A. E. Raftery, and M. S. Handcock, "Latent space approaches to social network analysis," *Journal of the American Statistical Association*, vol. 97, no. 460, pp. 1090–1098, 2002.

[19] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," 2016.

[20] J. Li, J. Zhu, and B. Zhang, "Discriminative deep random walk for network classification," pp. 1004–1013, 01 2016.

[21] B. Perozzi, R. Al-Rfou, and S. Skiena, "DeepWalk," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, aug 2014.

[22] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, aug 2017.

[23] C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao, "Scalable graph embedding for asymmetric proximity," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, pp. 2942–2948, AAAI Press, 2017.

[24] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[25] Z. Li, W. Yang, S. Peng, and F. Liu, "A survey of convolutional neural networks: Analysis, applications, and prospects," 2020.

[26] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016.

[27] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," 2017.

[28] S. Brody, U. Alon, and E. Yahav, "How attentive are graph attention networks?," 2021.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[30] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2017.

[31] K. Liu, S. Gao, P. Qiu, X. Liu, B. Yan, and F. Lu, "Road2vec: Measuring traffic interactions in urban road system from massive travel routes," *ISPRS International Journal of Geo-Information*, vol. 6, no. 11, 2017.

[32] J. Hu and L. Chen, "Multi-attention based spatial-temporal graph convolution networks for traffic flow forecasting," in *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–7, 2021.

[33] OpenStreetMap, "Openstreetmap Nodes," 2022. [Online; accessed 26-May-2022].

[34] OpenStreetMap, "Openstreetmap Ways," 2022. [Online; accessed 26-May-2022].

[35] OpenStreetMap, "Openstreetmap Tags," 2022. [Online; accessed 26-May-2022].

[36] OpenStreetMap, "Openstreetmap Wiki," 2022. [Online; accessed 26-May-2022].

[37] A. Narayanan, M. Chandramohan, R. Venkatesan, L. Chen, Y. Liu, and S. Jaiswal, "graph2vec: Learning distributed representations of graphs," 2017.

[38] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[39] McCormick, "Word2vec tutorial - the skip-gram model," Apr 2016.

[40] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," 2013.

[41] L. Wåhlin, "Towards machine learning enabled automatic design of it-network architectures," 2019.

[42] S. Even and G. Even, *Graph Algorithms*. Cambridge University Press, 2011.

[43] Y. Ma and J. Tang, *Deep Learning on Graphs*. Cambridge University Press, 2021.

[44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, pp. 84–90, may 2017.

[45] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2013.

[46] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," 2015.

[47] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, pp. 4–24, jan 2021.

[48] A. F. Agarap, "Deep learning using rectified linear units (relu)," 2018.

[49] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

[50] T. N. Kipf and M. Welling, "Variational graph auto-encoders," 2016.

[51] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning* (I. Guyon, G. Dror, V. Lemaire, G. Taylor, and D. Silver, eds.), vol. 27 of *Proceedings of Machine Learning Research*, (Bellevue, Washington, USA), pp. 37–49, PMLR, 02 Jul 2012.

[52] J. Zhai, S. Zhang, J. Chen, and Q. He, "Autoencoder and its various variants," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 415–419, 2018.

[53] H. Pishro-Nik, *Introduction to Probability, Statistics, and Random Processes*. Kappa Research, LLC, 2014.

[54] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," 2020.

[55] S. Pan, R. Hu, G. Long, J. Jiang, L. Yao, and C. Zhang, "Adversarially regularized graph autoencoder for graph embedding," 2018.

[56] W. Yu, C. Zheng, W. Cheng, C. C. Aggarwal, D. Song, B. Zong, H. Chen, and W. Wang, "Learning deep network representations with adversarially regularized autoencoders," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '18, (New York, NY, USA), pp. 2663–2671, Association for Computing Machinery, 2018.

[57] A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, mar 2020.

[58] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[59] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.

[60] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," 2016.

[61] G. Boeing, "Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks," *Computers, Environment and Urban Systems*, vol. 65, pp. 126–139, 2017.

[62] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," tech. rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[63] E. Fitkov-Norris, S. Vahid, and C. Hand, "Evaluating the impact of categorical data encoding and scaling on neural network classification performance: The case of repeat consumption of identical cultural goods," vol. 311, pp. 343–352, 09 2012.

[64] 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*, vol. 12, pp. 2825–2830, 2011.

[65] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman, "Missing value estimation methods for DNA microarrays ," *Bioinformatics*, vol. 17, pp. 520–525, 06 2001.

[66] S. G. K. Patro and K. K. Sahu, "Normalization: A preprocessing stage," 2015.

[67] L. Rabiner and B. Juang, "An introduction to hidden markov models," *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[68] M. Li, P. Tong, M. Li, Z. Jin, J. Huang, and X.-S. Hua, "Traffic flow prediction with vehicle trajectories," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 294–302, May 2021.

[69] B. Hui, D. Yan, H. Chen, and W.-S. Ku, "Trajnet: A trajectory-based deep learning model for traffic prediction," in *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '21, (New York, NY, USA), pp. 716–724, Association for Computing Machinery, 2021.

[70] H. Hong, Y. Lin, X. Yang, Z. Li, K. Fu, Z. Wang, X. Qie, and J. Ye, "Heteta: Heterogeneous information network embedding for estimating time of arrival," pp. 2444–2454, 08 2020.

[71] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," 2019.

[72] D. Schultes, "Route planning in road networks," 01 2008.

[73] D. Eppstein and S. Gupta, "Crossing patterns in nonplanar road networks," 2017.

[74] E. Seneta, "Markov and the birth of chain dependence theory," *International Statistical Review / Revue Internationale de Statistique*, vol. 64, no. 3, pp. 255–263, 1996.

[75] V. Salnikov, M. T. Schaub, and R. Lambiotte, "Using higher-order markov models to reveal flow-based communities in networks," *Scientific Reports*, vol. 6, mar 2016.

[76] F. Wu, T. Zhang, A. H. d. Souza, C. Fifty, T. Yu, and K. Q. Weinberger, "Simplifying graph convolutional networks," 2019.

[77] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016.

[78] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-GCN: A temporal graph convolutional network for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, pp. 3848–3858, sep 2020.

[79] S. Guo, Y. Lin, H. Wan, X. Li, and G. Cong, "Learning dynamics and heterogeneity of spatial-temporal graph data for traffic forecasting," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2021.

[80] B. Yu, H. Yin, and Z. 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*, International Joint Conferences on Artificial Intelligence Organization, jul 2018.

[81] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[82] J. Zhu, Y. Song, L. Zhao, and H. Li, "A3t-gcn: Attention temporal graph convolutional network for traffic forecasting," 2020.

[83] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[84] L. Moreira-Matias, J. Gama, M. Ferreira, J. Mendes-Moreira, and L. Damas, "Predicting taxi-passenger demand using streaming data," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1393–1402, 2013.

[85] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "CRAWDAD dataset epfl/mobility (v. 2009-02-24)." Downloaded from `https://crawdad.org/epfl/mobility/20090224`, Feb. 2009.

[86] L3S Research Center, "Homepage," 2022. [Online; accessed 14-Okt-2022].

[87] C.-Y. J. Peng, K. Lee, and G. M. Ingersoll, "An introduction to logistic regression analysis and reporting," *The Journal of Educational Research*, vol. 96, pp. 14–3, 2002.

[88] K. Gurney, *An Introduction to Neural Networks*. USA: Taylor & Francis, Inc., 1997.

[89] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen, *Classification and Regression Trees*. Chapman and Hall/CRC, 1984.

[90] Y. Duan, Y. L.V., and F.-Y. Wang, "Travel time prediction with lstm neural network," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1053–1058, 2016.

[91] D. Wang, J. Zhang, W. Cao, J. Li, and Y. Zheng, "When will you arrive? estimating travel time based on deep neural networks," in *AAAI*, 2018.

[92] H. Wu, Z. Chen, W. Sun, B. Zheng, and W. Wang, "Modeling trajectories with recurrent neural networks," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pp. 3083–3090, 2017.

[93] C. Liang, J. Berant, Q. Le, K. D. Forbus, and N. Lao, "Neural symbolic machines: Learning semantic parsers on freebase with weak supervision," 2016.

[94] A. Y. Xue, R. Zhang, Y. Zheng, X. Xie, J. Huang, and Z. Xu, "Destination prediction by sub-trajectory synthesis and privacy protection against such prediction," in *Proceedings of The 29th IEEE International Conference on Data Engineering*, April 2013.

[95] A. Xue, J. Qi, X. Xie, R. Zhang, J. Huang, and Y. Li, "Solving the data sparsity problem in destination prediction," *The VLDB Journal*, vol. 24, 04 2014.

[96] M. Ojala and G. C. Garriga, "Permutation tests for studying classifier performance," in *2009 Ninth IEEE International Conference on Data Mining*, pp. 908–913, 2009.

[97] I. Jolliffe and Springer-Verlag, *Principal Component Analysis*. Springer Series in Statistics, Springer, 2002.

[98] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[99] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018.

[100] S. Law, M. Berghauser Pont, Y. Shen, and A. Penn, "Identifying street-character-weighted local area using locally weighted community detection methods. the case study of london and amsterdam," 07 2019.

[101] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, (Cambridge, MA, USA), pp. 849–856, MIT Press, 2001.

[102] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[103] M. Grandini, E. Bagli, and G. Visani, "Metrics for multi-class classification: an overview," 2020.

[104] G. Tsoumakas, I. Katakis, and I. Vlahavas, *Mining Multi-label Data*, pp. 667–685. 07 2010.

[105] A. Botchkarev, "A new typology design of performance metrics to measure errors in machine learning regression algorithms," *Interdisciplinary Journal of Information, Knowledge, and Management*, vol. 14, pp. 045–076, 2019.

[106] *Spearman Rank Correlation Coefficient*, pp. 502–505. New York, NY: Springer New York, 2008.

[107] G. Rousselet and C. Pernet, "Improving standards in brain-behavior correlation analyses," *Frontiers in Human Neuroscience*, vol. 6, 2012.

[108] D. Picard, "Torch.manual_seed(3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision," 2021.

[109] R. Meyes, M. Lu, C. W. de Puiseau, and T. Meisen, "Ablation studies in artificial neural networks," 2019.

[110] A. Kok, E. Hans, and M. Schutten, "Vehicle routing under time-dependent travel times: The impact of congestion avoidance," *Computers & OR*, vol. 39, pp. 910–918, 05 2012.

[111] A. Genender-Feltheimer, "Visualizing high dimensional and big data," *Procedia Computer Science*, vol. 140, pp. 112–121, 2018. Cyber Physical Systems and Deep Learning Chicago, Illinois November 5-7, 2018.

[112] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," 2018.

[113] L. van der Maaten and G. Hinton, "Viualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 11 2008.

[114] L. N. Smith, "Cyclical learning rates for training neural networks," 2015.

[115] L. Prechelt, *Early Stopping — But When?*, pp. 53–67. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[116] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," 2017.

[117] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan, "Supervised contrastive learning," 2020.

[118] D. Chen, C. Xiong, and M. Zhong, "Improved lstm based on attention mechanism for short-term traffic flow prediction," in *2020 10th International Conference on Information Science and Technology (ICIST)*, pp. 71–76, 2020.

[119] X. Ran, Z. Shan, Y. Fang, and C. Lin, "An lstm-based method with attention mechanism for travel time prediction," *Sensors*, vol. 19, p. 861, 02 2019.

[120] J. Zhu, Y. Song, L. Zhao, and H. Li, "A3t-gcn: Attention temporal graph convolutional network for traffic forecasting," 2020.

[121] B. Lim, S. O. Arik, N. Loeff, and T. Pfister, "Temporal fusion transformers for interpretable multi-horizon time series forecasting," 2019.

[122] "Anaconda software distribution," 2020.

[123] S. Gillies *et al.*, "Shapely: manipulation and analysis of geometric objects," 2007–.

[124] OpenStreetMap, "Openstreetmap Highway," 2022. [Online; accessed 30-Aug-2022].

Acronyms

BERT Bidirectional Encoder Representations from Transformers. 27, 36, 47, 104

BFS breadth-first search. 21, 104

CNN Convolutional Neural Network. 22, 104

DFS depth-first search. 21, 104

FMM Fast Map Matching. 32

GAE Graph Auto Encoder. 24, 25, 46, 57, 58, 62, 64–66, 71, 72, 75, 76, 79, 80, 104, 107

GAT Graph Attention Network. 15, 23, 29, 47, 57, 64, 75

GCN Graph Convolutional Network. 10, 15, 22, 23, 25, 29, 47, 48, 57, 62, 64–67, 71, 72, 75, 76, 79, 104

GNN Graph Neural Network. 10, 11, 15, 22

GPS Global Positioning System. 13, 14, 29–32, 42, 81, 90, 91, 93, 105, 107

GRU Gated Recurrent Unit. 76, 78

GTC Graph Trajectory Convolution. 6, 11, 12, 33, 35–37, 39, 41, 50, 62, 64–69, 76, 79–82, 107

GTN Graph Trajectory Network. 6, 11, 12, 36, 41, 42, 50, 57, 59–64, 71–73, 79–83, 86, 105–107

HRNR Hierarchical Road Network Representation. 49, 58, 59, 80

ITS intelligent transportation systems. 6, 10–12, 14, 16, 37, 64, 67–70, 78, 79, 82, 83, 107

KNN K-Nearest Neighbor. 29

LSTM Long-Short Term Memory. 11, 37–40, 44, 45, 57, 63, 65, 67, 78, 87, 106, 108

MAE mean absolute error. 6, 51, 58, 60, 68, 77

MLM masked language modeling. 27, 36, 104

MSE mean squared error. 24, 40

NLP natural language processing. 15, 19, 27

NSP next sentence prediction. 27, 36, 104

OSM OpenStreetMap. 12, 13, 17–19, 28, 29, 42, 92, 104, 105, 109

PCA Principle Component Analysis. 46, 58, 59, 75–78

RFN Relational Fusion Network. 48, 58, 59, 80

RMSE root mean squared error. 51, 58, 77

RNN Recurrent Neural Network. 26

SGC Simple Graph Convolution. 34

SOTA state-of-the-art. 10–12, 22, 42, 46, 57, 61, 76, 78–82

T-GCN Temporal Graph Convolutional Network. 76–78, 81

T-GTC Temporal Graph Trajectory Convolution. 6, 11, 12, 37, 39, 40, 74–78, 81–83, 87, 105, 108

T-SNE T-Distributed Stochastic Neighbor Embedding. 71

TSD Trajectory Smoothed DeepWalk. 6, 11, 12, 35, 36, 38, 50, 62, 64–66, 79, 80, 82, 87, 107

List of Figures

List of Tables