Department of Informatics
Technical University of Munich

Master's Thesis in Informatics

Predictive Representations for Traffic Scenes Using Graph Neural Networks

Prädiktive Repräsentationen von Verkehrssituationen mittels Graph Neural Networks

Supervisor Prof. Dr.-Ing. Matthias Althoff

Advisor Eivind Meyer, M.Sc.

Author Max Schickert

Date July 15, 2022 in Garching

Disclaimer

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, July 15, 2022

(Max Schickert)

Acknowledgments

I would like to thank Eivind Meyer for his advice, helpful discussions and constructive feedback.

Abstract

In this master's thesis, we study the applicability of graphs and graph neural networks for computing predictive vehicle representations from traffic scenarios. Based on a set of assumptions and constraints we systematically design a spatio-temporal heterogeneous graph as a structured representation of a traffic scenario. The graph models vehicles, past vehicle states and the road geometry and topology. Vehicle interactions as well as spatial and temporal relationships are modeled as edges of the graph. We subsequently leverage the relational structure by proposing a heterogeneous graph neural network that can learn direct and higher-order interactions between entities of the heterogeneous graph and uses relational reasoning to compute vehicle representations. For training the state representation framework we adopt an end-to-end trainable encoder-decoder architecture. The graph neural network constitutes the encoder, while the decoder is used to formulate learning objectives in the form of domain-specific prediction tasks and loss functions that incorporate prior knowledge and direct the model to learn predictive representations. We collect and preprocess a traffic scenario dataset comprised of real-world urban road networks with simulated vehicle traffic. The dataset is used to train and evaluate our model. The results show that the proposed system is able to extract relevant and predictive features from traffic scenarios and can generalize to new scenarios.

Zusammenfassung

In dieser Masterarbeit untersuchen wir die Eignung von Graphen und Graph Neural Networks für die Berechnung von prädiktiven Fahrzeugrepräsentationen aus Verkehrsszenarien. Basierend auf einer Reihe von Annahmen und Bedingungen entwerfen wir systematisch einen räumlich-zeitlichen heterogenen Graphen als strukturierte Darstellung eines Verkehrsszenarios. Der Graph modelliert Fahrzeuge, frühere Fahrzeugzustände sowie die Straßengeometrie und -topologie. Fahrzeuginteraktionen sowie räumliche und zeitliche Beziehungen werden als Kanten des Graphen modelliert. Anschließend nutzen wir die relationale Struktur, indem wir ein heterogenes Graph Neural Network vorschlagen, das direkte und übergeordnete Interaktionen zwischen Objekten des heterogenen Graphen lernen kann und relationale Schlussfolgerungen zur Berechnung von Fahrzeugrepräsentationen verwendet. Für das Training der Zustandsdarstellung verwenden wir eine ganzheitlich trainierbare Kodierer-Dekodierer-Architektur. Das Graph Neural Network bildet den Kodierer, während der Dekodierer dazu dient, Lernzielen in Form von domänenspezifischen Vorhersageaufgaben und Verlustfunktionen zu formulieren, die Vorwissen einbeziehen und das Modell leiten, prädiktive Repräsentationen zu lernen. Wir sammeln und verarbeiten einen Datensatz von Verkehrsszenarien, der aus realen städtischen Straßennetzen mit simuliertem Fahrzeugverkehr besteht. Der Datensatz wird zum Training und zur Evaluierung unseres Modells verwendet. Die Ergebnisse zeigen, dass das vorgeschlagene System in der Lage ist, relevante und prädiktive Merkmale aus Verkehrsszenarien zu extrahieren und auf neue Verkehrsszenarien zu verallgemeinern.

Contents

1 Introduction **3**
1.1 Contributions . . . 4
1.2 Structure . . . 4

2 Background **7**
2.1 Deep Representation Learning . . . 7
2.1.1 Inductive Bias . . . 7
2.2 Graphs . . . 8
2.3 Graph Neural Networks . . . 9
2.3.1 Graph Convolutions . . . 9
2.3.2 Graph Convolutional Network . . . 10
2.4 Variational Autoencoders . . . 11
2.4.1 Conditional Variational Autoencoders . . . 12
2.5 CommonRoad . . . 12
2.5.1 Lanelets . . . 12

3 Related Work **15**
3.1 Deep Representation Learning . . . 15
3.2 Graph Neural Networks . . . 16
3.3 Trajectory Prediction . . . 17

4 Methodology **19**
4.1 Graph Representation . . . 21
4.1.1 Transformation Invariance . . . 22
4.1.2 Lanelets . . . 23
4.1.3 Vehicles . . . 25
4.1.4 Vehicle-Lanelet Edges . . . 27
4.2 Model Architecture . . . 28
4.3 Encoder Component . . . 28
4.3.1 Lanelet Bound Encoder . . . 29
4.3.2 Lanelet Edge Type Encoder . . . 29
4.3.3 Temporal Edge Encoder . . . 29
4.3.4 Graph Neural Network Model . . . 30
4.4 Decoder Component . . . 33
4.4.1 Drivable Area . . . 33
4.4.2 Trajectory Prediction . . . 36
4.4.3 Prior Losses . . . 39
4.4.4 Hybrid Loss . . . 40

5 Evaluation **41**
5.1 Benchmark Dataset and Preprocessing . . . 41

5.2 Training 43
5.3 Results 45
5.3.1 Vehicle Representation 45
5.3.2 Drivable Area Prediction 45
5.3.3 Trajectory Prediction 46
5.3.4 Ablation Study 47

6 Future Work **51**

7 Conclusion **53**

A Appendix **55**
A.1 Decoder Output Samples 55
A.2 Dataset and Model Parameters 55

Bibliography **59**

Notation

$\mathcal{V}$	Set of nodes of a graph.
$\mathcal{E}$	Set of edges of a graph.
$\mathcal{N}(v)$	Set of nodes which are connected to node v via an edge (node neighborhood).
$\mathcal{E}(s,t)$	Set of directed edges from node s to node t.
$\boldsymbol{x}_v$	Node attribute vector of node $v \in \mathcal{V}$.
$\boldsymbol{x}_{(s,r,t)}$	Edge attribute vector of edge $(s,r,t) \in \mathcal{E}$.
$\sigma : \mathbb{R}^d \to \mathbb{R}^d$	Activation function (elementwise non-linearity).
$\boldsymbol{z} = [\boldsymbol{x} \| \boldsymbol{y}]$	Concatenation of vectors $\boldsymbol{x} \in \mathbb{R}^{d_x}$ and $\boldsymbol{y} \in \mathbb{R}^{d_y}$ results in vector $\boldsymbol{z} \in \mathbb{R}^{d_x+d_y}$.
$\boldsymbol{z} = \Big\Vert_{k=1}^{K} \boldsymbol{x}_k$	Concatenation of vectors $\boldsymbol{x}_1 \in \mathbb{R}^{d_{x_1}}, \ldots, \boldsymbol{x}_K \in \mathbb{R}^{d_{x_K}}$ results in vector $\boldsymbol{z} \in \mathbb{R}^{\sum_k d_{x_k}}$.
$f'(x), f''(x)$	First and second derivative of f w.r.t. its parameter: $\frac{\mathrm{d}f}{\mathrm{d}x}$, $\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}$.
$A = \mathrm{diag}(\boldsymbol{a})$	Diagonal matrix $A \in \mathbb{R}^{d \times d}$ with values $a_1, \ldots, a_d$ on the diagonal.
${}^{L}\boldsymbol{x}$	Vector $\boldsymbol{x}$ in coordinate frame L.

Chapter 1

Introduction

Fueled by advances in the field of artificial intelligence, especially machine learning, autonomous driving and advanced driver assistance systems have experienced a surge of interest and research activity in recent years. Though not yet widely deployed, autonomous vehicles can already be seen driving and transporting passengers on public roads. Fully automated vehicles promise a major transformation of the mobility sector in the coming years. Many approaches for developing learning-based planners and control algorithms for intelligent automated vehicles exist. In order to make safe and effective decisions, they require an accurate understanding of the current state of the environment [LVL14] that includes other traffic participants and information about the road network. Planning into the future additionally necessitates prediction of future behavior of other traffic participants that takes interactions between them into account. The basis for the successful use of learning-based control algorithms are informative representations that extract task-relevant features from observations [BCV13].

In this thesis we systematically design a framework for extracting predictive vehicle state representations from traffic scenarios, based on a graph neural network (GNN). Graphs are a natural fit for modeling complex system with interacting heterogeneous entities. We represent traffic scenarios as spatio-temporal heterogeneous graphs which encode vehicles as well as the road network and integrate temporal and spatial relationships between entities. Thus, as opposed to related works, all information pertaining to the traffic scenario is unified in a single graph. Several benefits are associated with this. The structured representation provided by a graph enables us to explicitly model pair-wise vehicle interactions, dependencies on the geometry and topology of the road network and model vehicle dynamics and driver intent by incorporating past vehicle states. Graph neural networks have the capacity to learn representations of discrete entities and relations between them [Bat+18] to allow for structured computation on the graph. Thus, relationships between entities can be learned automatically. We use an attention-based graph neural network architecture to allow the model to learn to assess the relevance of related information. By extending the Heterogeneous Graph Transformer [Hu+20] we base our work on a proven GNN architecture but adapt it to ensure that all features encoded in our graph representation are exploited in the message passing computation of the GNN.

For training the state representation framework we adopt an encoder-decoder architecture. The vehicle state representation is produced by a graph neural network-based encoder. We formulate learning objectives in the form of decoders which constitute domain-specific prediction tasks and loss functions which incorporate prior knowledge. Two decoders are proposed that base their predictions exclusively on the vehicle representation. While this increases the difficulty of the learning task as the decoder do not have access to the scenario graph, it is essential for learning as it trains the encoder network to aggregate all relevant context for the prediction tasks in the vehicle representations. A drivable area decoder pre-

dicts a two-dimensional binary drivable area mask around the target vehicle which requires knowledge of nearby road geometry and traffic participants. A multi-modal trajectory predictor using the Conditional Variational Autoencoder framework generates a diverse set of trajectory hypotheses from a learned distribution over future trajectories. The trajectory decoder utilizes a kinematic vehicle model to produce realistic trajectories. Once the system is trained only the encoder component is retained for computing vehicle embeddings.

To allow for flexibility in defining features of our traffic scenario graph, we collect and preprocess our own traffic scenario dataset. The dataset comprises urban traffic scenarios from European cities. We evaluate the proposed system using our dataset. Quantitative and qualitative evaluations of vehicle state representations and prediction tasks performed by the decoder heads are conducted. The results show that the system computes representations that are a sufficient basis for the prediction tasks and hence contain relevant and predictive features.

1.1 Contributions

The main contributions of this thesis are summarized as:

- We propose a set of requirements for predictive vehicle representations. Based on these requirements we design a **spatio-temporal graph representation of traffic scenarios** (Section 4.1) that includes not only vehicles but also the road network and previous vehicle states. Edges between vehicles and vehicles and the road network enable relational reasoning.
- We propose an end-to-end trainable **deep representation learning architecture based on a graph neural network model** for extracting vehicle representations from traffic scenario graphs (Section 4.2).
- We propose and implement several adaptations to a **heterogeneous graph neural network model** (Section 4.3.4), which we subsequently use in our architecture.
- Furthermore, we collect and preprocess a large **traffic scenario dataset** comprised of real-world urban road networks with simulated vehicle traffic (Section 5.1).
- The dataset is used to **evaluate the performance of our proposed neural network architecture** (Section 5.3) by analyzing the learned representation, evaluating drivable area prediction results and comparing trajectory prediction results with a baseline model. We also perform ablation studies to determine the impact of our modeling decisions.

1.2 Structure

This thesis is structured as follows. In Chapter 2 we introduce and explain concepts that are used throughout this document. Chapter 3 reviews related work pertaining to deep representation learning, graph neural networks and traffic trajectory prediction. Next, Chapter 4 presents our approach to learning predictive state representations of traffic participants from structured traffic scenario data and describes our proposed architecture in detail. In Chapter 5 we introduce our benchmark dataset, describe our training procedure and evaluate the performance of the trained model. Chapter 6 provides ideas for direction of future research.

Lastly, Chapter 7 summarizes our work and results and concludes this thesis. In Appendix A we present examples of predictions produced by our model as well as details about dataset and model that are referenced throughout the thesis.

Chapter 2

Background

2.1 Deep Representation Learning

Representation learning is concerned with extracting features from high-dimensional data which describe underlying explanatory factors of the data that are useful to the task at hand. [BCV13]. Deep representation learning specifically uses deep neural networks as feature extractors, which are typically trained by optimizing a training objective function via local gradient descent. Learned representations can then be used as the basis for other machine learning tasks, for example as a state representation for Reinforcement Learning policies [De +18]. The performance of these downstream tasks depends on the suitability of the data representation in use.

An advantage of learning representations from data is that the various implicit factors which influence the representation can be learned automatically as long as they can be expressed via training criteria. This is opposed to the more traditional hand-crafting of features, which is often a time-intensive process that requires extensive domain knowledge.

We can formulate objectives for the learned representation by incorporating prior knowledge as assumptions or constraints into the learning process. These are referred to as inductive biases. General inductive biases express constraints that are applicable to a wide range of problems, for example sparsity, smoothness or temporal coherence of the representation [BCV13]. In the context of deep neural networks we can also incorporate inductive biases into the network architecture, for example by adapting it to the topological structure of the problem. Convolutional neural networks, for example, impose translation equivariance when working with grid-structured data. Graph neural networks impose node permutation equivariance. Additionally, prior domain knowledge can also be used to help guide the learning process. Prior domain knowledge can, for example, be expressed in the form of an auxiliary objective function which represents a domain-specific downstream task [Les+18]. By incorporating the objective into the learning process, additional constraints are imposed on the solution space.

2.1.1 Inductive Bias

Inductive biases in the context of deep learning are assumptions or constraints that are imposed either on the space of solutions or encoded in the network architecture which prioritize some solutions over others a priori, i.e. independently of observed data [Bat+18; San+21]. Many problems allow for multiple, equally good solutions. These often stem from regularities or symmetries that are produced by underlying structure in the data [Bro+21]. Similarly, deep neural networks represent a parametric function with numerous parameters which influence the output. But there can exist multiple parameter constellations which do not alter

the output of the function. Feed-forward neural networks, for example, are invariant to certain permutations of the network weights. By explicitly modeling these invariances, inductive biases can reduce the number of parameters and improve generalization. This allows the model to "[...] remove sensitivity of the representation to directions of variance in the data that are uninformative to the task at hand" [BCV13, p. 24], i.e. to trade expressivity for better sample complexity and generalization. It can constrain an otherwise ambiguous, multi-modal solution space to have a unique solution.

2.2 Graphs

A graph is a structured but flexible data representations which comprises entities and their relationship with each other. Graphs can represent data from a variety of sources, for example social network friendship data, citation networks, proteins and traffic data. To accommodate for the graph representation used in this thesis we formulate a custom definition of a graph, adapted from [Bat+18].

Definition 2.2.1 (Graph)**.** A directed heterogeneous multi-graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{R})$ is defined as a tuple of a set of nodes (entities) $\mathcal{V}$, edges (relationships between entities) $\mathcal{E}$, node types $\mathcal{A}$ and edge types $\mathcal{R}$. Edges are defined as a 3-tuple of source node, edge type and target node $\mathcal{E} = \mathcal{V} \times \mathcal{R} \times \mathcal{V}$. Each node $v \in \mathcal{V}$ is assigned a node type via the node type mapping function $\tau_{\mathcal{V}}(v) : \mathcal{V} \to \mathcal{A}$. Edges $e \in \mathcal{E}$ are analogously associated with an edge type $\tau_{\mathcal{E}}(e) : \mathcal{E} \to \mathcal{R}$. In addition, each node and edge have associated attribute vectors which we denote as $\mathbf{x_v} \in \mathbb{R}^{D_{\mathcal{V}}}$ for node attributes and $\mathbf{x_e} \in \mathbb{R}^{D_{\mathcal{E}}}$ for edge attributes.

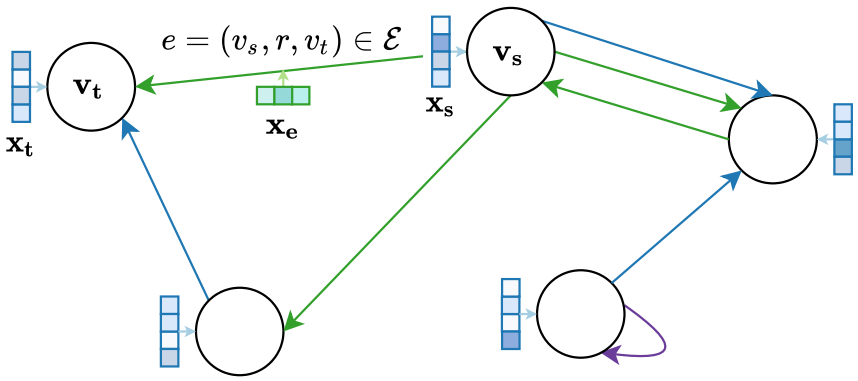

Figure 2.1: Example of a graph as defined by Definition 2.2.1. Edge types are encoded as different colors. Colored squares represent node and edge attribute vectors. Not all edge attribute vectors are visualized.

Figure 2.1 shows an example of a graph that can be expressed with our definition of a graph. Note that this definition allows for self-edges and multiple edges of different types between a pair of nodes.

Graph-global information can be represented as node attribute data of a *global node* which is connected to all other nodes via bidirectional edges of a special type [Gil+17].

Definition 2.2.2 (Spatio-temporal Graph)**.** Spatio-temporal graphs encode spatial relationships as well as a time dimension in the graph structure. For example, nodes can be associated with a time step and edges can encode relative positions of connected nodes. As opposed to dynamic graphs, temporal graphs are static, i.e. their structure and attributes are not time dependent.

2.3 Graph Neural Networks

Graph neural networks (GNNs) [GMS05; Sca+08] are a class of neural network models which perform computations on graph-structured data via graph convolutions. Like other neural networks, they can learn high-level representations, but can do so from explicitly structured data comprised of discrete entities with complex pairwise relationships between them that is represented in the form of a graph. GNNs bring relational inductive biases that are encoded in their architecture which enable these networks to exploit structure when computing representations of entities [Bat+18].

A graph neural network architecture is composed of multiple stacked graph computation layers [KW16]. The input of each layer is the output of the previous layer, or the input to the neural network for the first layer. A GNN can update node, edge and graph-global features. The node connectivity (i.e. existence of edges) is generally not modified by the computation.

We can distinguish between three general categories of tasks for which GNNs are commonly utilized:

- **Node-level** tasks compute an output for each node in the graph (e.g. node representation learning, node classification, node clustering).
- **Edge-level** tasks compute an output either for each actual edge in the graph or for each possible edge in the graph. The latter problem is called link prediction.
- **Graph-level** tasks compute features for the whole graph, thereby essentially predicting a graph-global property (e.g. graph classification).

A challenge when working with graph data from a deep learning perspective is encoding the graph structure into a representation that is suitable for processing with neural networks. More specifically, the variably-sized sets of nodes and edges have to be represented such that they are permutation invariant since they have no canonical order. This is addressed by graph convolutions.

2.3.1 Graph Convolutions

Convolutions on regular grid data as is used in Convolutional Neural Networks (CNNs) [LB+95] can be generalized to more irregular graph data. Two approaches exist in the literature. Spectral graph convolution [HVG11; Bru+13] is based on spectral graph theory and utilizes spectral decomposition of the graph Laplacian to define a graph convolution. Spatial graph convolution utilizes a node's spatial neighbors to define a convolution operation that propagates information along edges to neighboring nodes (message passing), thereby only aggregating information from the local neighborhood. For an in-depth background regarding both approaches we refer the interested reader to [Wu+20]. We will focus on spatial graph convolution in this thesis as it is less computationally intensive and thus can be applied to larger graphs. In addition, it allows us to integrate edge attribute information into the message passing operation and can process heterogeneous graphs.

We introduce the general Message Passing Neural Network (MPNN) framework for spatial GNNs that was proposed by [Gil+17]. Let $\boldsymbol{h}_t^{(l)} \in \mathbb{R}^{D_\mathcal{V}}$ be the hidden representation of node t at layer l. For each graph neural network layer we update the hidden node representation of each node $t \in \mathcal{V}$:

$$\boldsymbol{m}_t^{(l+1)} = \bigsqcup_{\substack{\forall s \in \mathcal{N}(t) \\ \forall e \in \mathcal{E}(s,t)}} M(\boldsymbol{h}_s^{(l)}, \boldsymbol{h}_t^{(l)}, e^{(l)}) \tag{2.1}$$

$$\boldsymbol{h}_t^{(l+1)} = U\left(\boldsymbol{h}_t^{(l)}, \boldsymbol{m}_t^{(l+1)}\right) \tag{2.2}$$

The message passing operation is performed in three steps. Updated node representations are computed based on aggregated messages from neighboring nodes:

1. **Message:** A message is computed for all neighboring nodes $s \in \mathcal{N}(t)$ using message function M. The message is computed based on the source and target node representations as well as edge information from the previous layer.

2. **Aggregation:** Messages from neighboring nodes are aggregated (gathered) using an aggregation function. The aggregation function is permutation invariant. This is key for allowing message passing to model node permutation invariant operations. Equation (2.1) denotes the aggregation with the $\square$ symbol. Examples for aggregation functions are sum, mean or max.

3. **Update:** Aggregated messages and the target node are inputs to the update function U which computes the updated node representation.

Equation (2.1) combines step 1 and 2 while Equation (2.2) shows the update step. Crucially, the message and update functions are shared for all message computations of a single GNN layer. Similar to convolution kernels, this helps keep the number of learnable weights low.

Each layer of a spatial GNN, because it performs one message passing step, aggregates information from the 1-hop neighborhood. Therefore, a K-layer graph neural network aggregates information from all nodes which are at most K hops away. We refer to K as the depth of the GNN, the furthest distance that information is propagated along the edges of the graph.

2.3.2 Graph Convolutional Network

To give an example of the message passing step of a GNN model we express the Graph Convolutional Network (GCN) [KW16] in the MPNN framework. The GCN model assumes a directed homogeneous (a single node and edge type) graph without self-edges. Edges are weighted with scalar edge weights.

Let $e_{s,t} \in \mathbb{R}$ be the edge weight for an edge from node s to node t with $e_{t,t} = 1$ for self-edges. Then $d_t = 1 + \sum_{s \in \mathcal{N}(t)} e_{s,t}$ is the sum of incoming edge weights for node t. It is used as a normalization to avoid exploding gradients.

Updated hidden representations are computed as follows:

$$\boldsymbol{h}_t^{(l+1)} = \sigma\left(\Theta^\top \sum_{\forall s \in \mathcal{N}(t) \cup \{t\}} \frac{e_{s,t}}{\sqrt{d_s d_t}} \boldsymbol{h}_s^{(l)}\right) \tag{2.3}$$

where $\Theta \in \mathbb{R}^{D_\mathcal{V} \times D_\mathcal{V}}$ is a learnable weight matrix.

By defining the following message and update equations we can represent the GCN model in the MPNN framework:

$$M(\boldsymbol{h}_s^{(l)}, t) = \frac{e_{s,t}}{\sqrt{d_s d_t}} \boldsymbol{h}_s^{(l)} \tag{2.4}$$

$$U(\boldsymbol{m}_t^{(l+1)}) = \sigma\left(\Theta^\top \boldsymbol{m}_t^{(l+1)}\right) \tag{2.5}$$

2.4 Variational Autoencoders

The Variational Autoencoder (VAE) [KW13; Kin17] models a latent variable generative process. High-dimensional data from real-world observations can often be described using much lower-dimensional explanatory factors [BCV13]. However, often these factors are latent, i.e. not directly observed. This underlying latent structure is modeled in the probabilistic generative model ($\boldsymbol{\theta} \in \mathbb{R}^d$ are parameters of the distributions):

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) = p_{\boldsymbol{\theta}}(\boldsymbol{z})\, p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) \tag{2.6}$$

Equation (2.6) defines a two-step generative process that is assumed by the Variational Autoencoder model: 1. Sample a set of latent variables $\boldsymbol{z}$ from the prior distribution: $\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\boldsymbol{z})$. 2. Sample data $\boldsymbol{x}$ from the generative distribution conditioned on $\boldsymbol{z}$: $\boldsymbol{x} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$.

The parameters of the generative model can be learned from i.i.d. (independent and identically distributed) data samples using maximum likelihood estimation. However, in many cases the marginal log-likelihood over observed data $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$ is intractable to compute. Instead of optimizing the likelihood directly we can apply the stochastic gradient variational Bayes (SGVB) [KW13; RMW14] framework and optimize the variational lower bound $\mathcal{L}_{VAE}(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\phi})$ of the log-likelihood [KW13, p. 3]:

$$\begin{aligned} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) &= D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \| p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[-\log q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) + \log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})\right] && (2.7) \\ &\geqslant -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \| p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\right] \quad =: \mathcal{L}_{VAE}(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\phi}) && (2.8) \end{aligned}$$

where $D_{KL}(q(z) \| p(z)) = \mathbb{E}_{z \sim q(z)}\left[\log q(z) - \log p(z)\right] \geqslant 0$ is the Kullback-Leibler divergence between distributions q and p. It is a non-negative measure of the similarity of two distributions which is 0 iff the two distributions are equal [KL51].

Equation (2.7) introduces a distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ which approximates the true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. $q_{\boldsymbol{\phi}}$ can be an arbitrary distribution, but the lower bound will be closer to the (intractable) true posterior the better $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ approximates $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ [KW13]. $q_{\boldsymbol{\phi}}$ is known as the recognition model.

A Gaussian prior on the latent variables, i.e. $p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z}\,|\,\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is a popular choice because it allows a differentiable closed-form solution for the Kullback-Leibler divergence term of the variational lower bound [Doe16]. The second term of the lower bound can be approximated using L Monte Carlo estimates of the expectation. It is made differentiable w.r.t. both $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ by applying the "reparameterization trick" [KW13; RMW14]. The lower bound becomes:

$$\tilde{\mathcal{L}}_{VAE}(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{\phi}) = -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \| p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \frac{1}{L}\sum_{l=1}^{L} \log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}^{(l)}) \tag{2.9}$$

where $\boldsymbol{z}^{(l)} = g_{\boldsymbol{\phi}}(\boldsymbol{\epsilon}^{(l)}, \boldsymbol{x})$ and $\boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(\boldsymbol{\epsilon}\,|\,\mathbf{0}, \mathbf{Id}_n)$. Due to the "reparameterization trick" the $\boldsymbol{z}^{(l)}$ are no longer sampled directly from the prior distribution but instead the result of a deterministic function $g_{\boldsymbol{\phi}}$ that is fully differentiable with respect to $\boldsymbol{x}$.

Neural networks can be used to model $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}^{(l)})$. Since the final objective function $\tilde{\mathcal{L}}_{VAE}$ is differentiable with respect to both of its parameters, the whole model can be optimized using gradient descent.

2.4.1 Conditional Variational Autoencoders

The Conditional Variational Autoencoder (CVAE) [SLY15; Wal+16] slightly modifies the generative model of the VAE by differentiating between input variables $\boldsymbol{x}$ and output variables $\boldsymbol{y}$.

$$p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{z}) \tag{2.10}$$

The conditional generative process under this model is: 1. Sample a set of latent variables $\boldsymbol{z}$ from the conditional prior distribution conditioned on the given input $\boldsymbol{x}$: $\boldsymbol{z} \sim p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. 2. Sample data $\boldsymbol{y}$ from the generative distribution conditioned on $\boldsymbol{x}$ and $\boldsymbol{z}$: $\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{z})$.

Example

When we use a standard multivariate normal distribution as the prior distribution over latent variables $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \mathbf{Id}_n)$ and model the recognition model as a normal distribution with parameters from a neural network (known as the encoder) $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x},\boldsymbol{y}) = \mathcal{N}(\mu_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y}), \Sigma_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{y}))$ we can compute the Kullback-Leibler divergence term analytically [Doe16]. The generative distribution $p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{z})$ is also modeled with a neural network known as the decoder. The computational graph used during training- and test-time is shown in Figure 2.2. The encoder is only used during training.

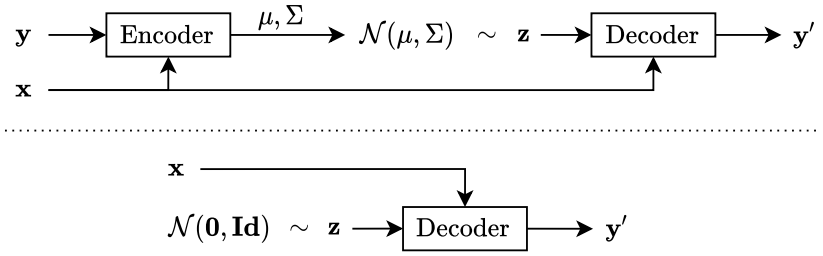

Figure 2.2: CVAE model during training-time (top) and test-time (bottom). The reparameterization trick, which is used for sampling the latent variable, is not shown.

2.5 CommonRoad

CommonRoad (composable benchmarks for motion planning on roads) [AKM17] is a collection of traffic scenarios for benchmarking motion planning on roads. A key objective of CommonRoad is to ensure that benchmarks are comparable and reproducible by providing a comprehensive, high-level description of the driving environment. A traffic scenario is composed of a road network graph, static obstacles as well as dynamic obstacles and their movement over time. The project defines an XML-based file format for serializing traffic scenarios.

2.5.1 Lanelets

The road network is represented by a set of lanelets [BZS14]; drivable road segments which are interconnected to form a directed road network graph. Each lanelet consists of a left and right bound represented as a list of vertices, which determines the lane geometry. The driving direction is always in the direction of the lanelet. A lanelet can additionally be linked to a

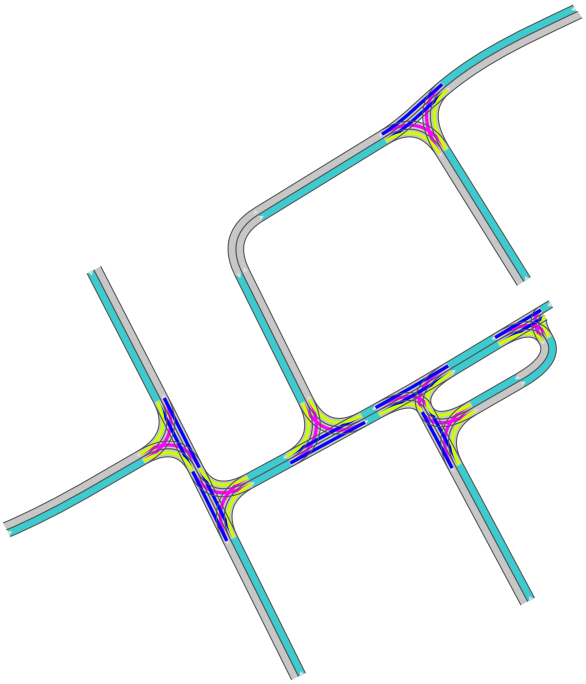

Figure 2.3: Example of a CommonRoad scenario rendered using the CommonRoad Python library. Only the lanelet network is shown. Scenario downloaded from *https://commonroad.in.tum.de/scenarios*, benchmark ID `DEU_Moelln-4_1_T-1`.

set of regulatory elements, which are used to describe properties of the lanelet such as speed limit. Lanelets are atomic in that all regulatory elements and adjacency connections are valid for the full length of the lanelet. The road topology is captured by adjacency connections between lanelets. CommonRoad defines successor, predecessor, adjacent left and adjacent right connection types.

Chapter 3

Related Work

Various approaches to deep state representation learning, relational reasoning using neural networks and traffic prediction have been proposed over the years. This section reviews works from the literature that motivate and relate to our approach. It summarizes works pertaining to representation learning, graph neural networks, as well as multi-modal trajectory prediction and traffic trajectory prediction using graph neural networks. The connection and differences to our work are stated in each section.

3.1 Deep Representation Learning

The domain of representation learning is concerned with researching methods to automatically extract meaningful features from data [JB15]. Deep representation learning focuses on designing feature extractors using deep neural networks, which learn representation from data, often in an unsupervised setting. The performance of downstream machine learning tasks heavily depends on data representation and whether the representation captures underlying (latent) explanatory factors in the data [BCV13]. Recent advances in fields like natural language processing (NLP) were enabled by advances in representation learning [Vas+17; Dev+18; Bro+20]. Large language models learn representations for text on which task-specific learning algorithms are based.

A challenge during the design of deep representation learning architectures is formulating learning objectives which encode the intention of learning a ”useful” representation. Among other things, this can be achieved by constructing learning objectives and constraints from prior knowledge, which are called *priors* in the literature. Bengio et al. [BCV13], in their comprehensive review of the field of representation learning, list general-purpose priors which we can impose as constraints on the learned representations. Jonschkowski and Brock [JB15] propose robotic priors which can be applied for learning representations which capture the state of the environment for system which operate in and interact with this environment. They formulate robotic priors as loss functions, which, when minimized, ensure that the representation encodes said priors.

In the domain of Reinforcement Learning (RL), separating state representation learning from policy learning was shown to benefit learning efficiency and computational complexity [BSG11; MKB16; Les+18]. Raw observations are often high-dimensional and contain redundant or irrelevant information with respect to the learning objective. Low-dimensional learned representation, on the other hand, extract only discriminative latent features which explain essential properties of the environment more directly.

In this work we learn a predictive state representation from a graph representation, using both general as well as domain-specific prior knowledge. Representation learning is guided

by downstream prediction tasks.

3.2 Graph Neural Networks

Graph neural networks are a neural network architecture for learning node and edge representations for graph-structured data. GNNs were first introduced in [GMS05] and [Sca+08]. We distinguish between spectral [HVG11; Bru+13; KW16; DBV16] an spatial (non-spectral) [HYL17; Vel+17; DB20] approaches. Spectral approaches use the spectrum of the normalized graph Laplacian in their definition of spectral graph convolution, which is expensive to compute for large graphs [KW16]. Spatial graph convolution (also referred to as message passing) aggregates information from the one-hop neighborhood of each node, which can be performed more efficiently. Depending on their scope, spatial GNNs are well characterized by the message passing neural network (MPNN) and graph network (GN) frameworks proposed by Gilmer et al. [Gil+17] and Battaglia et al. [Bat+18], respectively.

The Graph Attention Network (GAT) [Vel+17] adopts an attention mechanism for weighting incoming graph edges depending on node features of connected nodes. It assumes a homogeneous graph and does not consider edge attributes. An extension which processes edge attributes has been proposed in [CC21], but the authors assume an undirected homogeneous graph. GAT defines the calculation of the attention coefficient as a dot product of a learnable weight vectors and a linear transformation of node attributes. This definition has been shown to be of limited expressivity [BAY21]. Other attention-based GNN models define a more expressive dot-product attention mechanism based on query and key vectors [Vas+17] and integrate edge attributes, but do not consider heterogeneous graphs [DB20; Yin+21; CC21].

A variety of heterogeneous attention-based GNNs have been proposed. In these architectures, features are first projected into a common space before they are combined during the attention or message computation. This is because heterogeneous nodes may have node-type-specific feature distributions. Projecting to a common space ensures that features are compatible. The Heterogeneous Graph Attention Network (HAN) [Wan+19] uses the limited attention computation proposed by GAT and does not consider edge attributes. In [MXL21] the authors propose the HEAT graph neural network layer which considers heterogeneous graphs and incorporates edge attributes but employs the limited attention mechanism from GAT. Hu et al. [Hu+20] propose the Heterogeneous Graph Transformer (HGT) model. Inspired by the attention mechanism used in the Transformer [Vas+17] architecture, they propose a node- and edge-type-specific attention mechanism which is computed from a separate query and key vectors. Message and aggregation functions also incorporate node and edge type into the computation. However, node attributes are not considered in their model. The authors also propose a relative temporal encoding (RTE) to represent relative time in the graph structure.

To fully utilize all features of a traffic scenario graph we require a GNN which processes heterogeneous graphs and incorporates both node and edge attributes. We adapt the HGT model proposed by Hu et al. [Hu+20] by extending it with additional graph processing capabilities in order to fit our requirements.

3.3 Trajectory Prediction

Trajectory prediction is concerned with predicting future states, given the current state or a sequence of previous states. In past years, there has been a strong interest in this area. Various approaches for predicting trajectories of pedestrians [Kit+12; Ala+16], motion from image sequences [Wal+16; VPT16] and vehicles have been proposed. Future states are dependent on various latent factors and often these are not observed. This leads to uncertainty in the predictions, which can be modeled by allowing for multi-modal trajectory prediction, i.e. predicting multiple trajectory hypotheses. This approach is followed in multiple related works [Wal+16; Lee+17; HSP19; Li+20a; Li+20b] which employ a Conditional Variational Autoencoder (CVAE) [SLY15] to generate a diverse set of trajectory samples from a distribution over future trajectories.

More recently, graph neural networks have become popular for processing traffic graphs before performing trajectory prediction. Graphs are a popular representation for traffic as they naturally represent the discrete set of vehicles and can encode vehicle interactions as edges of the graph. Considering interactions between multiple entities in a scene is crucial since future behavior does not exclusively depend on past states but is also influenced by the behavior of other entities.

Li et al. [LYC19] propose GRIP, an architecture which employs one-dimensional convolutions along the temporal dimension for capturing temporal features, followed by a spectral graph convolution to propagate vehicle features in an undirected homogeneous vehicle graph. A single trajectory hypothesis is predicted. They evaluate their model on two highway traffic datasets. Diehl et al. [Die+19] conduct a performance comparison of variants of GCN and GAT models, which are also evaluated on highway traffic datasets. In addition, the authors compare different strategies for constructing vehicle edges. They find that the GAT model, which uses an attention mechanism for computing weights for message aggregation, performs best. Li et al. [Li+20a] propose Social-WaGDAT, a comprehensive multi-modal trajectory prediction architecture using an attention-based graph neural network which also incorporates temporal information into the computation. They integrate kinematic constraints for future trajectory prediction. The authors, however, constrain themselves to fixed scenes. Context information is processed by convolutional neural networks, their graph neural network only deals with homogeneous vehicle graphs. Their model is evaluated on diverse datasets. Mo et al. [MXL20] propose ReCoG, which represent the traffic scenario as a heterogeneous graph of vehicle and local map nodes, thereby modeling interactions between vehicles as well as vehicle-road interactions. However, the road topology is not encoded in the graph structure as each map node is only connected to a single vehicle and edges between map nodes are not included in the graph. Edge attributes cannot be processed by their graph neural network. SCALE-Net is proposed in [JCK20] and represents the vehicle scenario as an edge-attributes homogeneous graph where edge attributes encode the spatial relationship between vehicle nodes. The road network or context information is not considered in their architecture. The performance is evaluated on highway datasets. Mo et al. [MXL21] propose a heterogeneous edge-enhanced graph attention network (HEAT) which considers historical states of traffic participants and the road network. They only use their GNN model to represent vehicle interactions. Historical states are processed by a recurrent history encoder, which ignores interactions and road network features. Information about the road network is only used for future trajectory prediction. The ReCoG, SCALE-Net and HEAT architectures all predict only a single trajectory.

Our trajectory prediction model differs from related work in that we predict multi-modal trajectories from a single vehicle representation vector, without direct access to previous trajectory data. While this constitutes a more difficult learning problem, our main objective

is learning a single representation vector for each vehicle which should encode the current and previous vehicle states and interactions with the environment. Our model incorporates kinematic constraints for predicting future vehicle trajectories. Related work does not encode temporal information and information about the environment in the graph structure. Some related works evaluate their proposed model on highway traffic datasets, which we consider less challenging than urban traffic scenarios as the road network is simple, and vehicle motion is mostly uniform.

Chapter 4

Methodology

The goal of this work is to model a graph neural network-based architecture for extracting predictive vehicle state representations from traffic scenarios, represented as spatio-temporal heterogeneous graphs. More specifically, the graph neural network computes a representation vector, also called embedding, for each vehicle in a traffic scenario. The vehicle representation shall encode useful information about vehicle interactions, the road network as well as previous states of the environment such that it can be used as the foundation for predicting future states of the traffic scenario. We focus on the task of short-term prediction, that is we do not aim to model long-term influences such as high-level route planning or distant obstacles.

Representing a traffic scenario as a graph has several advantages. Graphs offer a structured representation, enabling us to represent a variable number of discrete entities and explicitly model pair-wise relationships or interactions between them as edges. By employing a graph neural network we can then leverage this structure in the computation of vehicle representations. Pair-wise vehicle and vehicle-road interactions are expressed in the graph structure such that the GNN model can learn to account for dependencies on other vehicles and the road network. Higher-order interaction can also be learned by the model, by applying several graph neural network layers sequentially.

The graph representation of traffic scenarios is capable of describing a diverse set of road networks and traffic situations using a consistent set of entities and relations between them. Thus, once a neural graph network has learned to account for the various relationships in computing vehicle representations, it can be expected to generalize to a variety of traffic scenarios.

In order to learn useful representation we propose learning objectives and accompanying loss functions which encode desirable properties of the representation. They act as proxies to the ambiguous objective of learning to extract a *predictive* per-vehicle representation. The loss functions are modeled in such a way that minimizing them jointly results in what we define as a predictive representation.

Since we do not impose an explicit structure on the learned embedding vector, it is not directly interpretable. For this reason, we again use a learning-based approach utilizing neural networks for prediction tasks on the basis of the learned vehicle representation. We can thus learn representations in an end-to-end fashion.

In the following we formulate a list of assumptions and constraints that apply to the learning problem, which we collectively refer to as inductive biases. We are then able to perform *principled* modeling of a deep representation learning system by encoding them into the graph structure, neural network architecture and loss function. The basis and motivation for the inductive biases is the latent structure and underlying explanatory factors that determine the evolution of a traffic scenario over time. The movement of a vehicle, for example, depends on factors such as the presence and movement of other traffic participants, previous states

of the environment, vehicle dynamics and driver intent. Two broad categories of inductive biases are distinguished: general inductive biases and domain-specific ones. We begin with general-purpose inductive biases which are applicable to representation learning in general.

Sparsity, simplicity of the representation: We assume that a representation is sparse as not every traffic scenario will have all possible relevant factors [BCV13; JB15]. This can also be phrased as a bias toward simple, i.e. low-dimensional representations. We encode this inductive bias in the loss function of our model as well as by limiting the dimensionality of the representation.

Temporal coherence: Limited by the laws of physics, many properties of real-world scenes tend to change relatively slowly and smoothly over time [JB15]. The temporal coherence inductive bias encodes this fact by penalizing the change of a representation over time [BCV13]. We encode this in the loss function of our model.

Uncertainty: Unobserved factors lead to stochasticity in the environment that cannot be reliably captured by the representation. The abstract description of traffic scenarios, which is the basis for our dataset, does not capture all details of the environment. For example, we do not model high-level strategies of traffic participants such as their final destination and as such are unable to reliably predict the routing decisions traffic participants make at road junctions. Our assumption is that by explicitly accounting for uncertainty in the model, independent of the representation, the representation becomes less susceptible to random variations in the environment because these are clearly separated from deterministic factors. We capture uncertainty by modeling the trajectory prediction task as a latent variable generative process.

Domain-specific inductive biases encode domain-specific prior knowledge to constraint the learning problem.

Localized interactions: Representations for vehicles in a traffic scenario should include information from the local environment. This is because short-term planning is primarily influenced by the local road network and other traffic participants in proximity to the vehicle. This constraint is modeled in the traffic scenario graph and subsequently leveraged by the graph neural network.

Causality: Future states are the result of past actions. This fact should be encoded explicitly as the dependency of a representation on past states of the environment. Conversely, the representation of past states must not be influenced by information from future ones. We model this in the structure of the traffic scenario graph.

Transformation invariance: A traffic scenario contains spatial information, for example the lanelet geometry or the position of vehicles. CommonRoad scenarios define a global coordinate system into which the lanelet network and vehicles are placed. To allow for generalization, we require our model to be invariant to arbitrary translations and rotations of this global coordinate system.

Two domain-specific prediction tasks are proposed as proxy objectives that encode previously formulated inductive biases and direct the learning system to produce predictive features. They base their predictions exclusively on the vehicle representation. Accurate predictions from these prediction tasks imply that the vehicle representation can be considered to be predictive. We propose the following two tasks:

Reconstruction of the local environment: The local environment encompasses all entities within a defined distance to the vehicle. By demonstrating that it is possible to reconstruct entities in the local neighborhood from the learned representation alone, we can

establish that information about these entities is encoded in the representation. The local environment includes both the geometry of the relevant local road network as well as the position and orientation of close-by vehicles. We can combine both into a unified representation by modeling the *drivable area* around the vehicle. The drivable area is composed of non-obstructed road surface, i.e. the road surface minus the space occupied by other vehicles.

Multi-modal trajectory prediction: The representation should be sufficient to predict future states of the vehicle it represents. Trajectory prediction requires knowledge of factors like vehicle dynamics and driver intent. Since these factors cannot be reliably determined from the current time step along, the trajectory prediction task also ensures that the vehicle representation encodes information from past time steps. Uncertainty in the future is accounted for by modeling a distribution over future trajectories from which diverse trajectories can be sampled.

The following sections detail how heterogeneous traffic scenario graphs are constructed and how the graph neural network-based state representation and prediction task are composed into an end-to-end trainable system.

4.1 Graph Representation

We construct traffic scenario graphs from CommonRoad scenarios as they comprise our training dataset. A CommonRoad scenario S includes a lanelet network graph G_L as well as vehicles V and their states $X_{1:T}$ over $T \in \mathbb{N}$ discrete time steps. Time steps are spaced equally in time, we denote the time difference between adjacent time steps as Δt.

$$S = (G_L, V, X_{1:T}) \tag{4.1}$$

Vehicles are not guaranteed to exist over the lifetime of the scenario.

$$X_t = \{x_{t,v} | v \in V\} \tag{4.2}$$

We construct a graph which contains all vehicles and lanelets in the scenario and models their relationships. CommonRoad traffic scenarios can naturally be represented as heterogeneous graphs by modeling the set of vehicles and road segments as two types of nodes and relations between them as heterogeneous edges. Figure 4.1 shows an example of the conversion.

CommonRoad scenarios discretize the time dimension into a set of time steps. We adopt this premise and construct separate graphs for each time step of the traffic scenario. The heterogeneous scenario graph for time step t is denoted as $G^{(t)}$.

$$G^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)}, \mathcal{A}, \mathcal{R}) \tag{4.3}$$

We define two node types, vehicle nodes V and lanelet nodes L.

$$\mathcal{A} = \{V, L\} \tag{4.4}$$

Edges between all combinations of node types are modeled. In addition, a VTV edge type is defined, which connects vehicles nodes from different time steps.

$$\mathcal{R} = \{V2V, L2L, V2L, L2V, VTV\} \tag{4.5}$$

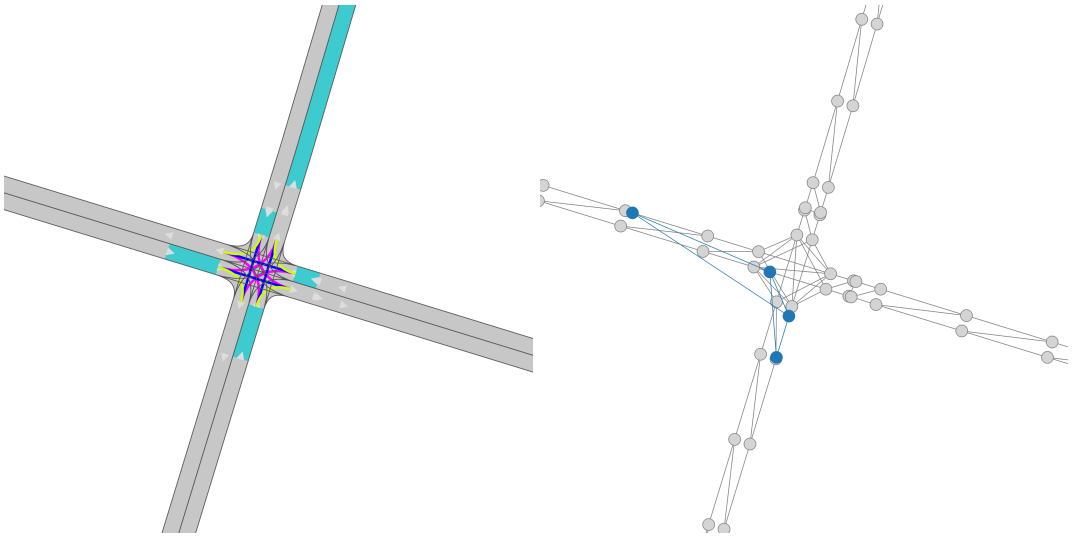

Figure 4.1: Example of the conversion from a CommonRoad scenario (left) to the equivalent traffic scenario graph (right). The graphic of the CommonRoad scenario only shows the lanelet network. In the traffic scenario graphs vehicle nodes and edges are colored blue, lanelet nodes and edges are colored gray. Only a single time step is shown.

The node and edge sets of $G^{(t)}$ are defined as:

$$\begin{aligned} \mathcal{V}^{(t)} &= \mathcal{V}_V^{(t)} \cup \mathcal{V}_L^{(t)} \\ \mathcal{E}^{(t)} &= \mathcal{E}_{V2V}^{(t)} \cup \mathcal{E}_{L2L}^{(t)} \cup \mathcal{E}_{V2L}^{(t)} \cup \mathcal{E}_{L2V}^{(t)} \end{aligned} \tag{4.6}$$

Ultimately, the spatio-temporal heterogeneous traffic scenario graph is defined as the union of all $G^{(t)}$, joined by temporal vehicle edges. It contains all time steps of a traffic scenario. While the lanelet graph is constant over time, we assume the learned representation of lanelet nodes to be conditioned on the time step of connected vehicle nodes. For this reason the lanelet graph is duplicated for each time step. The complete traffic scenario graph is denoted as G.

$$\begin{aligned} & G = (\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{R}) \\ \text{where} \quad & \mathcal{V} = \bigcup_t \mathcal{V}^{(t)} \\ & \mathcal{E} = \left(\bigcup_t \mathcal{E}^{(t)} \right) \cup \mathcal{E}_{VTV} \end{aligned} \tag{4.7}$$

4.1.1 Transformation Invariance

A traffic scenario defines the geometry of the road network as well as the position and orientation of vehicles over time. In CommonRoad scenarios, all spatial quantities are expressed in a shared global coordinate system. This simplifies relating entities to each other in space but poses a problem for machine learning models as it impedes generalization. An arbitrary rotation and translation of the global coordinate system should not affect the vehicle representations that are extracted by our model. We therefore design the graph representation to be invariant to an any transformation of the global coordinate system from the special Euclidean group in two dimensions $T \in SE(2)$ (an orthogonal linear transformation followed

by a translation):

$$T(\boldsymbol{x}) = R\boldsymbol{x} + \boldsymbol{t} \tag{4.8}$$
$$\text{where} \quad R \in \mathbb{R}^{2\times 2},\ R^\top = R^{-1},\ \det(R) = 1$$
$$\boldsymbol{t} \in \mathbb{R}^2$$

The transformation invariance is achieved by placing every lanelet and vehicle into its own exclusive, local coordinate system, thereby making it independent of the global coordinate system [JCK20; MXL21]. Any spatial information that pertains to the lanelet or vehicle is expressed in the local coordinate system. Spatial relations are preserved by storing relative quantities (e.g. relative position and orientations) in edge attributes of the graph. Since a transformation of the global coordinate system affects local coordinate frames uniformly, the relative coordinates of spatial relations are insensitive to such a transformation. The exclusive coordinate systems are aligned such that the x axis always points in the driving direction. We denote the coordinate system that a vector is expressed in via a superscript prefix. For example, vector $\boldsymbol{x}$ expressed in coordinate system V is denoted as ${}^{V}\boldsymbol{x}$.

The following sections detail how the graph is constructed, which node and edge features are provided as well as how we incorporate general-purpose prior knowledge as well as prior domain knowledge in the graph structure.

4.1.2 Lanelets

Lanelet nodes together with edges between them represent the road network of a traffic scenario. Each lanelet from a traffic scenario is represented as a corresponding lanelet node in the traffic scenario graph. The following sections describe lanelet node and edge attributes. Refer to Table 4.1 for a summary of lanelet node attributes and to Table 4.2 for a summary of lanelet edge attributes.

Lanelet Segmentation

The length of a lanelet is not bounded [BZS14] whereas lanelet node representation vectors are of fixed size and as such can only capture a limited amount of information. To resolve this disparity we define an upper bound on the length of each lanelet and segment all lanelet which exceed the maximum length into a set of shorter lanelets before creating lanelet nodes. Adjacency relations between lanelets apply to the entire length of the lanelet. As such, segmenting a lanelet in isolation would violate adjacency relations of adjacent lanelets. For this reason adjacent lanelets are segmented collectively such that segmentation points of adjacent lanelets align and we can maintain valid adjacency relations between them.

Lanelet Nodes ($\mathcal{V}_L$)

The road segment represented by a lanelet is made up of the space between the left and right lanelet bound polylines. Both polylines have the same number of vertices, denoted as $N \in \mathbb{N}$. We denote the left and right lanelet polylines as V_l and V_r respectively. Each polyline is made up of a sequence of vertex positions in the global coordinate system: $V_l = \langle \boldsymbol{v}_{l,i} \rangle_{i\in 1,\dots,N}$ and $V_r = \langle \boldsymbol{v}_{r,i} \rangle_{i\in 1,\dots,N}$.

Since we only include relative positions in the graph representation of the traffic scenario, the vertex positions in the left and right polylines are transformed to the lanelet-local coordinate system L. We denote them as ${}^{L}V_l$ and ${}^{L}V_r$.

Feature	Definition	Unit
Vertex positions of the lanelet left and right bound	$({}^{L}V_l, {}^{L}V_r)$	m
Lanelet length	$\|L\|_2$	m
Lanelet curvature	κ	$\mathrm{rad\,m^{-1}}$

Table 4.1: Lanelet node attributes.

By including the left and right lanelet boundary in lanelet node attributes we implicitly encode the lanelet width, local curvature as well as the center polyline, which can be interpreted as the ideal driving line.

The position $\boldsymbol{x}_L$ of lanelet L is the origin of the lanelet-local coordinate system which is defined as the midpoint between the first vertex of the left and right lanelet bound. The lanelet orientation θ_L is the angle between the ray from $\boldsymbol{x}_L$ to the midpoint between the second vertex of the left and right lanelet bound and the x axis.

$$\boldsymbol{x}_L = \frac{\boldsymbol{v}_{l,1} + \boldsymbol{v}_{r,1}}{2} \tag{4.9}$$

$$\theta_L = \operatorname{atan2}\left(\frac{(\boldsymbol{v}_{l,2} + \boldsymbol{v}_{r,2}) - (\boldsymbol{v}_{l,1} + \boldsymbol{v}_{r,1})}{2}\right) \tag{4.10}$$

where $\operatorname{atan2}(\boldsymbol{v})$ computes the angle between the x axis and the ray from the origin to $\boldsymbol{v} = \begin{pmatrix} x & y \end{pmatrix}^\top$.

The lanelet length $\|L\|_2$ is the Euclidean length of the center polyline (average of left and right polyline vertices) of the lanelet.

$$\|L\|_2 = \sum_{i=1}^{N-1} \left\| \frac{\boldsymbol{v}_{l,i+1} + \boldsymbol{v}_{r,i+1}}{2} - \frac{\boldsymbol{v}_{l,i} + \boldsymbol{v}_{r,i}}{2} \right\|_2 \tag{4.11}$$

We also compute the lanelet curvature. For a two-dimensional parametric curve $\boldsymbol{f}(s) = (x(s), y(s))$ we can compute the unit tangent vector at a point $s \in [a, b]$ along the curve as $\boldsymbol{T}(s) = \frac{\boldsymbol{f}'(s)}{\|\boldsymbol{f}'(s)\|_2}$. The curvature κ is defined as the change in direction of the tangent vector, i.e. the magnitude of the first derivative of the tangent vector [Kre13, pp. 34–35].

$$\kappa = \left\| \frac{\mathrm{d}\boldsymbol{T}(s)}{\mathrm{d}s} \right\|_2 = \frac{|x'(s)y''(s) - x''(s)y'(s)|}{(x'(s)^2 + y'(s)^2)^{\frac{3}{2}}} \tag{4.12}$$

Since lanelet bounds are described by polylines which are not continuous curves we cannot directly compute the first and second derivatives of x and y. Instead, we use finite difference approximation of the derivatives to compute them numerically.

Lanelet Edges ($\mathcal{E}_{L2L}$)

Lanelet edges connect to other lanelets which are adjacent to the current lanelet in the lanelet network of the CommonRoad scenario. The edges thus define lanelet topology and encode the spatial relationship between lanelets. Specifically, we model edges from the current lanelet L to predecessor and successor lanelets as well as adjacent lanelets, denoted as L'. Successor lanelets continue the current driving corridor. Their first vertices of the left and right lanelet bound are constrained to be at the same position as the last vertices of the current lanelet bound. A predecessor relation is the inverse of a successor relation, i.e. the current lanelet is a successor of its predecessor lanelet. Lanelets can have multiple predecessors and successors. We also define "sibling" relations to lanelets which connect to the

Feature	Definition	Unit
Edge type	$\tau_{\mathcal{E}}(e) \in \{$ `predecessor`, `successor`, `adjacent left same direction`, `adjacent left opposite direction`, `adjacent right same direction`, `adjacent right opposite direction`, `predecessor of successor`, `successor of predecessor` $\}$	
Relative orientation	$\theta_{L'} - \theta_L$	rad
Relative position	${}^{L}\boldsymbol{x}_{L'}$	m
Distance	$\|\boldsymbol{x}_{L'} - \boldsymbol{x}_L\|_2 = \|{}^{L}\boldsymbol{x}_{L'}\|_2$	m

Table 4.2: Lanelet edge attributes.

successor of the current lanelet in a predecessor relationship (edge type `predecessor of successor`) and those which connect to the predecessor in a successor relationship (edge type `successor of predecessor`). For adjacent lanelet we distinguish between the relative position to the current lanelet (left or right) and the traffic direction (same direction, opposite direction). All edge types are listed in Table 4.2. To allow for more efficient parameter sharing in the graph neural network we encode the edge type in the edge attributes instead of by modeling distinct relations for each edge type.

For an edge from lanelet L to L' we also compute the relative position and orientation of lanelets in the coordinate frame of the source lanelet L.

4.1.3 Vehicles

Vehicle nodes represent traffic participants at a specific time while vehicle edges $\mathcal{E}_{V2V}$ represent interaction between vehicles, allowing the model to learn interaction-aware representations.

Vehicle Nodes ($\mathcal{V}_V$)

A vehicle node $i \in \mathcal{V}_V^{(t)}$ represents vehicle i from the traffic scenario at time t. Multiple nodes representing the same vehicle can exist in the traffic graph as long as they are associated with distinct time steps.

We denote the position of vehicle i as $\boldsymbol{x}_i$ and the time derivatives as $\dot{\boldsymbol{x}}_i$ (velocity) and $\ddot{\boldsymbol{x}}_i$ (acceleration). The shape of vehicle i is approximated by a rectangle with lateral size w_i (width) and longitudinal size l_i (length). The vehicle position expresses the center of the vehicle shape. It is also the origin of a vehicle-local coordinate frame V that is used to express values relative to the vehicle. The x axis of the coordinate frame is aligned with the forward driving direction of the vehicle. Since our graph should be transformation-invariant, we do not include the absolute position and orientation of a vehicle in the node attributes. All vehicle node attributes are listed in Table 4.3.

Vehicle Edges ($\mathcal{E}_{V2V}$)

Vehicle edges connect vehicle nodes belonging to the same time step t.

$$\mathcal{E}_{V2V}^{(t)} = \bigcup_{i=1,\dots,|V|} \{e_{ij}\}_{j \in \mathcal{N}_{V^{(t)}}(i)} \tag{4.13}$$

Feature	Definition	Unit
Vehicle velocity	${}^{V}\dot{\boldsymbol{x}}_i$	$\mathrm{m\,s^{-1}}$
Vehicle acceleration	${}^{V}\ddot{\boldsymbol{x}}_i$	$\mathrm{m\,s^{-2}}$
Rectangular vehicle dimensions (width, length)	(w_i, l_i)	m

Table 4.3: Vehicle node attributes for vehicle i. All time-varying quantities are implicitly based on the current time step t.

Feature	Definition	Unit
Relative vehicle orientation	$\theta_j - \theta_i$	rad
Relative vehicle position	${}^{V}\boldsymbol{x}_j$	m
Vehicle distance	$\|\boldsymbol{x}_j - \boldsymbol{x}_i\|_2$	m
Relative vehicle velocity	${}^{V}\dot{\boldsymbol{x}}_j - {}^{V}\dot{\boldsymbol{x}}_i$	$\mathrm{m\,s^{-1}}$
Relative vehicle acceleration	${}^{V}\ddot{\boldsymbol{x}}_j - {}^{V}\ddot{\boldsymbol{x}}_i$	$\mathrm{m\,s^{-2}}$

Table 4.4: Vehicle edge attributes for edge from vehicle i to j. All time-varying quantities are implicitly based on the current time step t. Values are computed in the vehicle-centered coordinate frame V of the source vehicle.

Vehicle edges represent interactions between vehicles and also define the spatial relationship between them. Since we do not want to impose arbitrary constraints on the potential interactions between vehicles we construct edges to all vehicles within a large maximum distance D_{max}^{V2V}.

$$\mathcal{N}_{V^{(t)}}(i) = \{j \in \mathcal{V}_V^{(t)} \mid \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 \leqslant D_{max}^{V2V}\} \tag{4.14}$$

Relative position, velocity and acceleration edge attribute values are computed in the vehicle-centered coordinate frame V of the vehicle represented by source node i. Table 4.4 summarizes vehicle edge attributes.

Temporal Vehicle Edges ($\mathcal{E}_{VTV}$)

Lanelet and vehicle edges capture spatial dependencies and interactions between lanelet and vehicle nodes. We also model temporal vehicle edges which enable the graph neural network model to capture temporal dependencies in the traffic scenario graph.

There are different options for encoding the change in vehicle states over time. A vehicle trajectory can be encoded into a single feature vector which captures dynamics features of historic states [MXL21] such that each vehicle is represented as exactly one node in the graph. However, this discards information about vehicle interactions at previous time steps. For this reason we chose to explicitly express the temporal relationship of a vehicle to previous and subsequent vehicle states as temporal edges in the graph. Temporal edges connect vehicle nodes of the same vehicle at different time steps. We can encode causality in the evolution of vehicle states over time by constraining the direction of temporal edges to be forward in time., i.e. temporal vehicle edges relate vehicle nodes to future vehicle nodes but not to past ones. This effectively blocks information flow from future vehicle nodes to their past counterparts in GNNs since spatial graph convolutions only propagate information in the direction of edges. Temporal edges are constructed to all future time steps of the same vehicle up to T_{max}^{VTV} time steps into the future. Temporal edges allow each message passing step in the GNN to integrate information from T_{max}^{VTV} previous time steps.

Let v_t be a vehicle node at time t and $v_{t'}$ a vehicle node representing the same vehicle at time $t' < t$, connected by temporal edge $e \in \mathcal{E}_{VTV}$. Then $\Delta t_e = t - t' \geqslant \Delta t$ denotes the elapsed time. Relative quantities of the vehicle nodes connected by the temporal edge are specified in

Feature	Definition	Unit
Elapsed time	Δt_e	s
Relative vehicle orientation	$\theta_{v_t} - \theta_{v_{t'}}$	rad
Relative vehicle position	${}^{T'}\boldsymbol{x}_{v_t}$	m
Vehicle distance	$\|\boldsymbol{x}_{v_t} - \boldsymbol{x}_{v_{t'}}\|_2$	m
Relative vehicle velocity	${}^{T'}\dot{\boldsymbol{x}}_t - {}^{T'}\dot{\boldsymbol{x}}_{t'}$	$\mathrm{m\,s^{-1}}$
Relative vehicle acceleration	${}^{T'}\ddot{\boldsymbol{x}}_t - {}^{T'}\ddot{\boldsymbol{x}}_{t'}$	$\mathrm{m\,s^{-2}}$

Table 4.5: Temporal vehicle edge attributes for an edge e from $v_{t'}$ to v_t. Values are computed in the vehicle-centered coordinate frame T' of the source vehicle.

the vehicle-local coordinate frame T' of vehicle $v_{t'}$. Temporal vehicles edges (see Table 4.5) encode almost the same attributes as vehicle edges (see Table 4.4). Vehicle trajectories are implicitly encoded via temporal vehicle edges.

4.1.4 Vehicle-Lanelet Edges

Vehicle to lanelet ($\mathcal{E}_{V2L}$) and lanelet to vehicle ($\mathcal{E}_{L2V}$) edges relate vehicles and the road network. Each vehicle v is connected to all lanelets which contain the vehicle center (origin of the vehicle-centered coordinate frame $\boldsymbol{x}_v$) at the current time step via bidirectional vehicle-lanelet edges. Bidirectional edges are represented as two separate directed edges, each with identical edge attributes.

For an edge relating vehicle v to lanelet L the corresponding edge attributes specify the position of the vehicle on the lanelet. The vehicles' lanelet center position $\boldsymbol{x}_{L,c}$ is determined by laterally projecting the vehicle position onto the lanelet center polyline. $\boldsymbol{x}_{L,l}$ and $\boldsymbol{x}_{L,r}$ are the projected positions onto the left and right bound polylines of the lanelet. We define distance to left and right lanelet boundaries as d_l and d_r:

$$d_l = \|\boldsymbol{x}_{L,l} - \boldsymbol{x}_v\|_2 \tag{4.15}$$

$$d_r = \|\boldsymbol{x}_{L,r} - \boldsymbol{x}_v\|_2 \tag{4.16}$$

The signed offset from the lanelet center line d_e (lateral error) is defined as:

$$d_e = \frac{d_l - d_r}{2} \tag{4.17}$$

Let $l_v \in [0, \|L\|_2]$ denote the arc length from the start of the lanelet center polyline to $\boldsymbol{x}_{L,c}$. We also normalize the distance to be independent of the lanelet length by diving by $\|L\|_2$.

The orientation difference between vehicle and lanelet θ_e (heading error) is computed as the angle between the vehicle orientation θ_v and the lanelet orientation at $\boldsymbol{x}_{L,c}$:

$$\theta_e = \theta_L(\boldsymbol{x}_{L,c}) - \theta_v \tag{4.18}$$

Table 4.6 lists the previously defined edge attributes of vehicle-lanelet edges.

Feature	Definition	Unit
Distance to left lanelet boundary	d_l	
Distance to right lanelet boundary	d_r	
Signed offset from lanelet center line (lateral error)	d_e	
Arc length along lanelet center line	l_v	m
Normalized arc length along lanelet center line	$\frac{l_v}{\|L\|_2} \in [0,1]$	
Orientation difference between vehicle and lanelet (heading error)	θ_e	rad

Table 4.6: Vehicle-lanelet edge attributes. All time-varying quantities are implicitly based on the current time step t.

4.2 Model Architecture

In this section we present the components of our model architecture. We adopt a standard end-to-end trainable encoder-decoder architecture. The encoder employs a graph neural network to compute an embedding vector for each vehicle of the traffic scenario graph that it receives as input. The vehicle representations constitute the bottleneck of the architecture as they are the only input from which the decoder component computes the prediction tasks. The decoder is a combination of two decoder heads, one for each prediction task. Output from the decoder as well as the vehicle representations produced by the encoder component are used in multiple loss functions which we then combine into a hybrid loss such that the complete model can be trained by optimizing the hybrid loss function.

The decoder part of the architecture is only used for training and can be discarded after the model is trained. See Figure 4.2 for a visualization of this architecture.

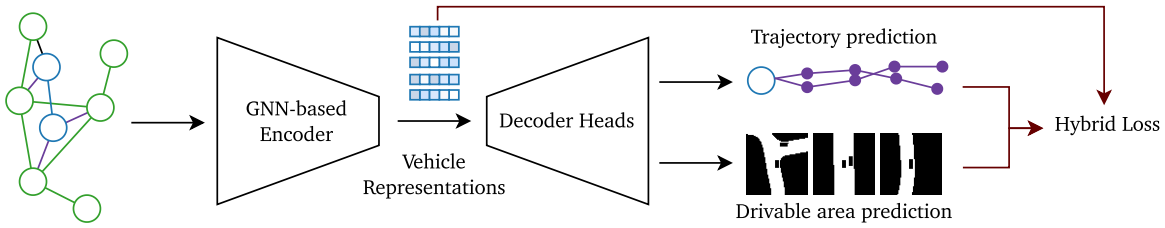

Figure 4.2: Overview of the encoder-decoder architecture. A heterogeneous traffic scenario graph (left) is passed to the GNN-based encoder which produces vehicle representations (middle). The representations are passed to decoder heads which produce task-specific outputs (right).

4.3 Encoder Component

The encoder component computes vehicle representations from a given traffic scenario graph. First, deep feature extractors prepare scenario graph attributes for processing by the graph neural network. Then L_{GNN} layers of our Edge-attributed Heterogeneous Graph Transformer (EHGT) model are applied sequentially to update node representations. The vehicle node representations of the final layer constitute the vehicle state representations.

4.3.1 Lanelet Bound Encoder

Part of lanelet node attributes are the variable-length sequence of vertex positions of the lanelets' left and right bound, which together define the lanelet geometry. Since the GNN cannot process variable-length sequence data in node attributes, we have to convert the sequence to a fixed-size embedding vector. A recurrent neural network is used for this task. The final hidden state of the recurrent network is used as the embedding vector and is concatenated with the remaining node attributes. We choose a gated recurrent unit (GRU) [Cho+14] as the recurrent neural network, which is a popular choice for sequence processing.

4.3.2 Lanelet Edge Type Encoder

A similar incompatibility exists with the categorical edge type value that is part of lanelet edge attributes. We convert the categorical attribute to a numerical vector by assigning each edge type a unique embedding vector. Like other weights of the network, these embedding vectors are updated via backpropagation, i.e. the embedding vectors are optimized during training. Each embedding vector is initialized with random values such that the model can learn to differentiate different edge types. Lanelet edge attributes are updated with the embedding vector, which replaces the categorical edge type value.

4.3.3 Temporal Edge Encoder

Temporal vehicle edges contain the relative time between time steps of the connected vehicle nodes as an edge attribute. This is necessary because we model temporal edges not only between vehicle nodes of consecutive time steps but additionally between more distant time steps. Relative time between vehicle states is crucial for computing time-dependent quantities like the velocity of a vehicle. We adopt Time2Vec [Kaz+19], a learnable vector representation of time that can capture periodic and non-periodic patterns. This is similar to the relative temporal encoding proposed in [Hu+20], however we capture the temporal relationship between nodes as temporal edges as opposed to augmenting the node representation. Time2Vec defines function $\mathrm{t2v}(\tau)$ for computing the $(k+1)$-dimensional time representation of time $\tau \in \mathbb{R}$, parameterized by learnable frequency $\boldsymbol{\omega} \in \mathbb{R}^{k+1}$ and phase shift $\boldsymbol{\varphi} \in \mathbb{R}^{k+1}$ weight vectors. The first dimension of the time representation is defined as a linear function of time, for capturing non-periodic patterns.

$$\mathrm{t2v}(\tau)_0 = \omega_0 \tau + \varphi_0 \tag{4.19}$$

The remaining dimension $i = 1, \ldots, k$ can be utilized to capture periodic patterns.

$$\mathrm{t2v}(\tau)_i = \sin(\omega_i \tau + \varphi_i) \tag{4.20}$$

We follow an initialization scheme similar to that proposed in [Vas+17]. We initialize $\omega_0 = 1$ and $\varphi_0 = 0$. By alternating the phase shift between 0 and $\frac{\pi}{2}$ for dimensions $i \geqslant 1$ we alternate between sine and cosine functions. The frequency weights for the remaining dimensions are initialized as follows:

$$\omega_{2i-1} = \omega_{2i} = \frac{2\pi}{C^{(2i-1)/k}} \tag{4.21}$$

where hyperparameter C defines the maximum period that the time representation should be able to capture. The time representation is added to the edge attributes of temporal vehicle edges.

4.3.4 Graph Neural Network Model

The graph neural network model is the key component of the encoder. It extracts a vehicle representation of each vehicle in the scenario from the traffic scenario graph by performing L_{GNN} sequential message passing steps over the graph, each represented by a GNN layer. Each message passing step successively updates the node representation of all nodes in the graph, including lanelet nodes. By composing multiple GNN layers, the network can learn higher-order interactions and relationships between nodes of the graph. Because all information pertaining to the traffic scenario is unified in the graph, the GNN model has access to all available context when computing vehicle representations.

Based on successful application in previous works [Vel+17; Die+19; Hu+20; JCK20; DB20; MXL21] we choose a heterogeneous GNN model which employs a masked self-attention mechanism to attend to incoming edges. The GNN defines node- and edge type-dependent message passing operations which allow it to, for example, learn specialized operations for vehicle interactions and incorporating temporal information.

In the following we first introduce the graph attention mechanism and heterogeneous GNN model which our graph neural network is based on. We then define the Edge-attributed Heterogeneous Graph Transformer (EHGT), which we use as the graph neural network model in the encoder.

Graph Attention Network

The use of an attention mechanism in GNNs was first introduced in [Vel+17] with the graph attention network (GAT) architecture. It works by assigning a scalar attention weight to each incoming message during the message passing step. An attention weight captures the relative importance of features from nodes in the neighborhood and enables the GNN to selectively aggregate information, based on the relevance to the target node. The attention weights are normalized to sum up to one using the softmax function.

Let $\alpha(s,t) \in \mathbb{R}$ be the attention weight for an edge from node $s \in \mathcal{N}(t)$ to $t \in \mathcal{V}$. The normalized attention weight $\bar{\alpha}(s,t)$ is:

$$\bar{\alpha}(s,t) = \frac{\exp(\alpha(s,t))}{\sum_{v \in \mathcal{N}(t)} \exp(\alpha(v,t))} \tag{4.22}$$

Messages from neighboring nodes are aggregated using a weighted sum with the normalized attention weights serving as the weights.

$$\boldsymbol{h}_t^{(l+1)} = \sigma\left(\sum_{\substack{\forall s \in \mathcal{N}(t) \\ \forall e \in \mathcal{E}(s,t)}} \bar{\alpha}^{(l+1)}(s,t)\, M^{(l+1)}\left(\boldsymbol{h}_t^{(l)}\right)\right) \tag{4.23}$$

Like the Transformer [Vas+17] model, the GAT architecture can be extended to use multiple attention heads. Instead of weighing all features of a message with the same scalar attention weight, multiple attention mechanisms can be used to allow the model to attend to different properties at different positions of the output. It also helps to stabilize the learning process [Vel+17]. Each attention head independently computes an updated node representation and the resulting vectors are concatenated to form the final updated representation. We denote the number of attention heads as $H \geqslant 1$.

$$\boldsymbol{h}_t^{(l+1)} = \Big\Vert_{k=1}^{H} \sigma\left(\sum_{\substack{\forall s \in \mathcal{N}(t) \\ \forall e \in \mathcal{E}(s,t)}} \bar{\alpha}_k^{(l+1)}(s,t)\, M^{(l+1,k)}\left(\boldsymbol{h}_t^{(l)}\right)\right) \tag{4.24}$$

GAT assumes a homogeneous graph and does not consider edge attributes during the update of hidden node representations $\boldsymbol{h}^{(l+1)}$.

Heterogeneous Graph Transformer

We base our graph neural network model on the Heterogeneous Graph Transformer (HGT) model proposed in [Hu+20]. Like the graph attention network architecture, HGT uses an attention mechanism to compute weights for averaging messages during the message passing step. It is explicitly designed to operate on heterogeneous graphs and accounts for node and edge types during the message passing computation. The HGT model only takes node attributes into account, edge attributes are not considered.

Depending on their type, node (and edge) attributes exhibit differing feature distributions. To still allow for meaningful combination of information from these diverse sources during message passing, the attribute vectors are projected into a common feature space using a linear transformation [Hu+20; Vas+17] before they are subsequently combined for attention or message computation and finally projected into the target node-type-specific feature space. This is achieved by conditioning both the computation of attention weights and of messages on the relation type triplet which consists of source node type $\tau_{\mathcal{V}}(s)$, edge type $\tau_{\mathcal{E}}(e)$ and target node type $\tau_{\mathcal{V}}(t)$ for an edge $e = (s, \tau_{\mathcal{E}}(e), t) \in \mathcal{E}$.

By projecting features from different distribution into a common feature space before projecting to the target node-type-specific feature space the authors of [Hu+20] reduce the number of linear transformations to $\mathcal{O}(2|\mathcal{A}| + |\mathcal{R}|)$ while still maintaining a type-specific transformation. A naive approach which projects all features from incoming edges to the target-specific feature space before further computations would require $\mathcal{O}\left(|\mathcal{A}|^2|\mathcal{R}|\right)$ distinct linear transformations.

Attention weights are computed using scaled dot-product attention [Vas+17]. The complete architecture uses multiple attention heads (see Equation (4.24)) however we will assume a single attention head in our equations to avoid cluttered notation. Each weight matrix is distinct per layer of the GNN architecture, but we keep this fact implicit in our notation. $W^{K}_{\tau_{\mathcal{V}}(s)}$ is the matrix for linear transformation of the key vector, $W^{Q}_{\tau_{\mathcal{V}}(t)}$ transforms the query vector. $W^{Att}_{\tau_{\mathcal{E}}e}$ is a matrix for conditioning the attention weight computation on the edge type. For edge $e = (s, \tau_{\mathcal{E}}(e), t)$ the attention weight is computed as follows:

$$\alpha^{(l+1)}(s, \tau_{\mathcal{E}}(e), t) = \left(W^{K}_{\tau_{\mathcal{V}}(s)}\boldsymbol{h}_s^{(l)}\right) W^{Att}_{\tau_{\mathcal{E}}(e)} \left(W^{Q}_{\tau_{\mathcal{V}}(t)}\boldsymbol{h}_t^{(l)}\right) \frac{\mu_{\tau_{\mathcal{V}}(s),\tau_{\mathcal{E}}(e),\tau_{\mathcal{V}}(t)}}{\sqrt{d}} \tag{4.25}$$

$\mu_{\tau_{\mathcal{V}}(s),\tau_{\mathcal{E}}(e),\tau_{\mathcal{V}}(t)} \in \mathbb{R}$ is a learnable prior weight meant to "denote the general significance of each [...] relation triplet" [Hu+20, p. 5].

The message function is parameterized by source node type and edge type but not by target node type. $W^{M,\mathcal{V}}_{\tau_{\mathcal{V}}(s)}$ represents the linear message transformation. $W^{M,\mathcal{E}}_{\tau_{\mathcal{E}}(e)}$ is an edge-type-specific matrix for conditioning the message on the edge type. The message for edge e is calculated as follows:

$$M^{(l+1)}(s, \tau_{\mathcal{E}}(e)) = W^{M,\mathcal{E}}_{\tau_{\mathcal{E}}(e)} W^{M,\mathcal{V}}_{\tau_{\mathcal{V}}(s)} h_s^{(l)} \tag{4.26}$$

Message aggregation is almost equivalent to the GAT architecture (see Equation (4.24)) with the addition of a target node-type-specific linear transformation and residual connection [He+16]. $W^{A,\mathcal{V}}$ transforms the aggregated message $\boldsymbol{h}_t^{(l+1)}$ to the node type-specific feature space. The final updated node representation $\tilde{\boldsymbol{h}}_t^{(l+1)}$ is calculated as follows:

$$\tilde{\boldsymbol{h}}_t^{(l+1)} = W^{A,\mathcal{V}}\boldsymbol{h}_t^{(l+1)} + \tilde{\boldsymbol{h}}_t^{(l)} \tag{4.27}$$

Edge-attributed Heterogeneous Graph Transformer (EHGT)

The Heterogeneous Graph Transformer model captures relation-type-specific interaction patterns when computing attention weights and messages from node attributes. However, to exploit all properties of our previously defined traffic scenario graph we introduced the Edge-attributed Heterogeneous Graph Transformer (EHGT) which extends the HGT model with new capabilities:

Edge attributes: The traffic scenario graph encodes, among other things, the spatial and temporal relationship of entities in edge attributes. These should, alongside node attributes, also be incorporated into the message passing operation. Therefore, we extend the HGT model to compute attention weights and messages from both node as well as edge attributes. Only node representations are updated in our model, edge attributes are static.

Global context: We propose to include a graph-global context vector in the update equations which captures globally relevant information. The global context vector is represented as an attribute vector of a virtual node with bidirectional edges to all nodes in the graph. Hence, it is included in the message aggregation alongside other nodes of the graph. Edges from and to the virtual global node do not have edge attributes.

Self-edges: We add self-edges to the graph as an alternative or supplement to the residual connection. Self-edges connect each node with itself, via an edge with unique edge type. They allow node attribute information from the node to be included in the neighborhood message aggregation as opposed to as part of the residual connection which does not use the proven self-attention mechanism.

Figure 4.3 visualizes the information flow during a message passing step of the model.

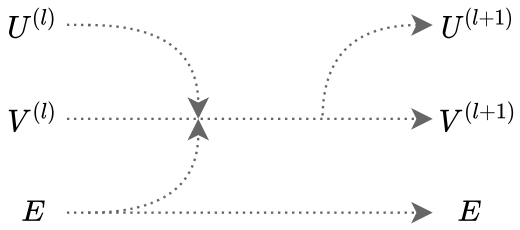

Figure 4.3: Information flow during a single message passing step to compute updated representations. U, V and E represent global attributes, the set of node attributes and the set of edge attributes, respectively. Global, node and edge attributes are used to compute updated node representations. Next, the updated global context vector is computed from updated node representations. Edge attributes are static, i.e. not updated.

Self-edges and edge to and from the global context node do not have edge attributes. For these edges we set edge attributes $\boldsymbol{x}_e = \mathbf{0}$.

Aggregation and update equations are similar to that of the HGT model.

$$\hat{\boldsymbol{h}}_t^{(l+1)} = W_{\tau(t)}^{Aggr} \, \sigma \left(\Big\Vert_{k=1}^{K} \sum_{\substack{s \in \mathcal{N}(t) \\ e \in \mathcal{E}(s,t)}} \alpha_k^{(l+1)}(s,e,t) \, M_k^{(l+1)}(s,e) \right) \tag{4.28}$$

$$\boldsymbol{h}_t^{(l+1)} = \hat{\boldsymbol{h}}_t^{(l+1)} + \boldsymbol{h}_t^{(l)} \tag{4.29}$$

The message function $M_k^{(l+1)}(s,e)$ uses both node and edge attributes. The node attribute vector is transformed using node type-specific weight matrix $W_{k,\tau_\mathcal{V}(s)}^{M,\mathcal{V}}$ before it is concatenated with the edge attributes and finally transformed using an edge type-specific linear transformation $W_{k,\tau_\mathcal{E}(e)}^{M,\mathcal{E}}$.

$$M_k^{(l+1)}(s,e) = W_{k,\tau_\mathcal{E}(e)}^{M,\mathcal{E}} \left[W_{k,\tau_\mathcal{V}(s)}^{M,\mathcal{V}} \boldsymbol{h}_s^{(l)} \| \boldsymbol{x}_e \right] \tag{4.30}$$

Edge attributes are not individually transformed as $W_{k,\tau_\mathcal{E}(e)}^{M,\mathcal{E}}$ already encodes the edge type-specific transformation. Notice that any additional node type-specific matrix transforming $\boldsymbol{x}_e$ could be absorbed into $W_{k,\tau_\mathcal{E}(e)}^{M,\mathcal{E}}$, therefore it does not add additional computational capability to the message function.

We use the scaled dot-product (i.e. multiplicative) attention [Vas+17] for computing content-based attention weights from node and edge attribute vectors.

$$\alpha_k^{(l+1)}(s,e,t) = \left(\left(\boldsymbol{k}_k^{(l+1)}(s,e) \right)^\top W_{k,\tau_\mathcal{E}(e)}^{Att} \boldsymbol{q}_k^{(l+1)}(t,e) \right) \frac{\mu_{k,(\tau_\mathcal{V}(s),\tau_\mathcal{E}(e),\tau_\mathcal{V}(t))}}{\sqrt{d}} \tag{4.31}$$

Key $\boldsymbol{k}_k^{(l+1)}(s,e)$ and query $\boldsymbol{q}_k^{(l+1)}(t,e)$ vectors are calculated from node and edge attributes.

$$\boldsymbol{k}_k^{(l+1)}(s,e) = W_{k,\tau_\mathcal{V}(s)}^{K,\mathcal{V}} \boldsymbol{h}_s^{(l)} + W_{k,\tau_\mathcal{E}(e)}^{K,\mathcal{E}} \boldsymbol{x}_e \tag{4.32}$$

$$\boldsymbol{q}_k^{(l+1)}(t,e) = W_{k,\tau_\mathcal{V}(t)}^{Q,\mathcal{V}} \boldsymbol{h}_t^{(l)} + W_{k,\tau_\mathcal{E}(e)}^{Q,\mathcal{E}} \boldsymbol{x}_e \tag{4.33}$$

The attention weights are normalized via the softmax function which is computed over all attention weights for the current target node, including those from other relation triplets.

$$\bar{\alpha}_k^{(l+1)}(s,e,t) = \underset{\substack{\forall s \in \mathcal{N}(t) \\ \forall e \in \mathcal{E}(s,t)}}{\text{softmax}} \left(\alpha_k^{(l+1)}(s,e,t) \right) \tag{4.34}$$

4.4 Decoder Component

The decoder component is designed to guide the encoder network towards learning to extract a *predictive* per-vehicle representation. It is composed of two decoder heads, each performing a separate prediction task, as well as additional loss functions which incorporate prior knowledge.

4.4.1 Drivable Area

We define one of the metrics for the predictiveness of a representation to be the ability to reconstruct the local environment around the vehicle. This is because the local environment encodes important context which informs driving decisions. Our traffic scenario graph is composed of two entities, the road network and vehicles, that make up the environment. We propose to model both in a fused representation of the vehicles' neighborhood: a two-dimensional binary drivable area mask, created from a top-view image of the area around the vehicle. Drivable area is defined as the area occupied by road surface which is not occupied by any vehicle. It therefore combines information about vehicles and the road network. Implicit in this representation are the geometry of the local road network, position and orientation of nearby vehicles as well as their size. Since we only wish to reconstruct the local environment, only the drivable area up to a limited distance around the vehicle is of interest.

We formalize the drivable area mask as function $M : \mathbb{R}^2 \to \{0,1\}$ which maps a position around the vehicle to a binary occupancy value, where 0 and 1 designate unoccupied and occupied space, respectively. As positions are expressed in a coordinate frame centered on the ego vehicle. The function is specific to a vehicle and time step (this is implicit in our notation). The drivable area is modeled up to an L_1 distance D^{DA} around the vehicle. For easier modeling we discretize the positions at which we sample the drivable area mask. We sample at equally-spaced positions represented by Cartesian product $P^2 \equiv P \times P$.

$$P = \left\{ \frac{2D^{DA}}{R^{DA}-1} k - D^{DA} \,|\, k \in \{0,\dots,R^{DA}\} \right\} \tag{4.35}$$
$$\text{where} \quad R^{DA} \in \mathbb{N}$$

These assumptions allow us to represent the drivable area as a binary image. R^{DA} is the resolution of the binary drivable area image. We denote the discretized drivable area mask function as $\bar{M} : P^2 \to \{0,1\}$. Figure 4.4 shows example images of drivable area.

We model the learning problem as binary classification of each position of the drivable area mask. Each position $k \in P^2$ is modeled as a Bernoulli distributed random variable $b_k = \bar{M}(k)$, which is conditioned on the vehicle representation $\boldsymbol{x}$. All positions are predicted independently of one another. The distribution over the complete drivable area mask is parameterized by parameters $\boldsymbol{\theta} \in [0,1]^{|P|^2}$

$$p_{\boldsymbol{\theta}}(\boldsymbol{b} \,|\, \boldsymbol{x}) = \prod_{k \in P^2} p_{\theta_k}(b_k \,|\, \boldsymbol{x}) \tag{4.36}$$
$$= \prod_{k \in P^2} \text{Bernoulli}(b_k \,|\, \boldsymbol{x}, \theta_k) \tag{4.37}$$

Before we present the network architecture which models parameters $\boldsymbol{\theta}$ and the loss function used for training, we first explain how the ground-truth drivable area images are produced from the dataset.

Drivable Area Rendering

Since the local drivable area can be represented by a binary image, we generate ground-truth image data by rendering lanelet and vehicle shapes using tools from computer graphics. Given a traffic scenario graph we render a drivable area image for each vehicle node. Black and white are used to represent non-drivable area and drivable area, respectively. The drivable area image is initialized by filling it with black color. The view is centered at the ego vehicle and the scene is scaled in such a way that the boundary pixels of the image are at L_1 distance D^{DA} to the vehicle.

Rendering is then performed in two steps. In the first step the lanelet geometry is rendered with white color. However, not all lanelets in the traffic scenario are rendered. We limit the set of rendered lanelets to the $(L_{GNN}-1)$-hop neighborhood of the lanelet node which the ego vehicle is located on. L_{GNN} is the number of GNN layers in the encoder network (c.f. Table A.2). Since each GNN layer propagates messages to all one-hop (i.e. direct) neighbors, L_{GNN} layers propagate information to all L_{GNN}-hop neighbors. This is why the vehicle representation can incorporate information from nodes which are at most L_{GNN} hops away (assuming there are no vehicle-to-vehicle edges). In particular, the representation can incorporate information from *lanelet* which are at most $L_{GNN}-1$ hops away because of the additional lanelet-vehicle edge. Lanelets which are more distant to the ego vehicle in the traffic scenario graph are therefore not rendered.

In the second step vehicle bounding boxes (represented by rectangles) are rendered with black color, thereby marking the area as occupied. The set of vehicles that should be rendered

Figure 4.4: Examples of binary drivable area in the local area around the ego vehicle. Unoccupied road surface is represented as white pixels. Here $D^{DA} = 32\,\text{m}$ and $R^{DA} = 64$.

is determined with the same algorithm as is used for lanelets. Figure 4.4 shows several examples of binary images representing drivable area around the ego vehicle.

Generator Network Architecture

We adopt a neural network architecture based on the transposed convolution (also called fractionally strided convolution) operation [RMC15] for predicting parameters $\boldsymbol{\theta}$ for the drivable area mask distribution. A Transposed convolution performs the reverse operation of a normal (discrete) convolution in that its output is of higher spatial dimensionality than its input (for more details see [RMC15; Shi+16]). If we assume two spatial dimensions, the transposed convolution operation essentially generates a higher-resolution image from a lower-resolution input image. By stacking multiple transposed convolutional layers we can generate images of arbitrary size.

The input to our drivable area generator network is the extracted vehicle representation. The data is first passed through two fully-connected layers. In preparation for the transposed convolutions it is reshaped to a two-dimensional image consisting of a single pixel with as many channels as the output dimensionality of the fully-connected layers. Next, a series of two-dimensional transposed convolution layers follow. Each layer doubles the spatial resolution and halves the number of channels. We adopt architecture guidelines from [RMC15] and apply Batch Normalization [IS15]. Since we require values in the range $[0,1]$, we apply the element wise sigmoid function $\sigma : \mathbb{R} \to (0,1)$ to the generator output.

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{4.38}$$

We denote the output of the generator network as $\hat{\boldsymbol{\theta}}$.

Drivable Area Loss

Since we are performing binary classification, we use the binary cross-entropy loss function, which computes the cross-entropy between the predicted and the ground-truth distributions. We compute the mean binary cross-entropy over all vehicles $i = 1,\ldots,N$ and all positions of

the drivable area mask $k \in |P|^2$.

$$\mathcal{L}_{\text{driv}} = -\frac{1}{N|P|^2} \sum_{i=1}^{N} \sum_{k=1}^{|P|^2} p_{\theta_k}(b_k \,|\, x) \log(p_{\hat{\theta}_k}(b_k \,|\, x)) + (1 - p_{\theta_k}(b_k \,|\, x)) \log(1 - p_{\hat{\theta}_k}(b_k \,|\, x)) \quad (4.39)$$

$$= -\frac{1}{N|P|^2} \sum_{i=1}^{N} \sum_{k=1}^{|P|^2} \theta_k \log(\hat{\theta}_k) + (1 - \theta_k) \log(1 - \hat{\theta}_k) \quad (4.40)$$

4.4.2 Trajectory Prediction

The trajectory prediction task is phrased as a (stochastic) sequence generation problem. Given a vehicle representation from the last observed time step T_{obs}, we design a trajectory predictor which can generate multiple plausible, short-term sequences of future vehicle states, i.e. trajectory hypotheses. Each trajectory is composed of a fixed number of T_{pred} future vehicle states. The vehicle states are predicted at regular time intervals Δt. As opposed to other trajectory prediction frameworks from the literature [JCK20; MXL21] we explicitly only use the representation from the most recent time step as input to the trajectory predictor. By doing so we encode the requirement that a vehicle representation vector should not only encode information about the current vehicle state but should additionally include information about past time steps.

The task of trajectory prediction is inherently stochastic as past observations of the traffic scenario do not yield enough information about the environment and traffic participants that it is possible to reliably predict the future when not given the capability of proposing multiple alternatives. This is because future states do not follow deterministically from past observations. Imagine a vehicle approaching an intersection. Without knowledge of the long-term route that the vehicle is following we cannot be certain whether the vehicle will continue straight or turn in either direction at the intersection, both alternatives are plausible trajectories. In addition, the model might not be able to reproduce path planning decisions of traffic participants for traffic scenarios with complex vehicle interactions. To account for these sources of uncertainty we make the stochastic nature of the task explicit in the design of our trajectory prediction model.

We adopt a probabilistic view by modeling the prediction task as a conditional latent variable generative process which defines a probability distribution over future trajectories. Latent variables $\boldsymbol{z}$ represent unobserved explanatory factors which influence the future trajectory. As explained in Section 2.4, they allow us to generate a diverse set of trajectory hypotheses while still ensuring that the individual samples are plausible. This encodes the uncertainty inductive bias into our model and should make the vehicle representations less susceptible to random variation in the environment as this is captured by the latent variable generative process.

Let $\boldsymbol{x}$ be a vehicle representation which includes information about the past trajectory of the vehicle from T_{obs} time steps. The future trajectory $\boldsymbol{Y}_{1:T_{pred}} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{T_{pred}}\}$ is made up of a sequence of future positions where each $\boldsymbol{y}_i$ represents the future vehicle state at time step $T_{obs} + i$ in the traffic scenario. Then the generative process which defines the distribution over future trajectories is modeled by the conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{Y}_{1:T_{pred}} \,|\, \boldsymbol{x})$:

$$p_{\boldsymbol{\theta}}(\boldsymbol{Y}_{1:T_{pred}} \,|\, \boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z} \,|\, \boldsymbol{x})\, p_{\boldsymbol{\theta}}(\boldsymbol{Y}_{1:T_{pred}} \,|\, \boldsymbol{x}, \boldsymbol{z})\, d\boldsymbol{z} \quad (4.41)$$

CVAE for Trajectory Generation

We employ a deep generative model in the form of a Conditional Variational Autoencoder (CVAE) architecture to model the latent variable generative process [Lee+17]. This allows us to incorporate the diverse sample generation into our neural network architecture in a differentiable way, meaning we can still train our model from data using gradient descent. The CVAE is composed of recognition model $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{Y})$, prior model $p_{\boldsymbol{\psi}}(\boldsymbol{z} \mid \boldsymbol{x})$ as well as generation model $p_{\boldsymbol{\theta}}(\boldsymbol{Y} \mid \boldsymbol{x}, \boldsymbol{z})$ where $\{\boldsymbol{\phi}, \boldsymbol{\psi}, \boldsymbol{\theta}\}$ are the parameters of the models. We use a multivariate normal distribution as the distribution for latent variables and relax the assumption that the prior distribution over latent variables is conditioned on the vehicle representation [Kin+14; SLY15]. The prior distribution is a standard multivariate normal distribution.

$$p_{\psi}(\boldsymbol{z} \mid \boldsymbol{x}) = p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \mathbf{Id}) \tag{4.42}$$

Both the recognition and the generation models are parameterized using neural networks. The recognition network (also called encoder) produces parameters for the multivariate normal distribution: mean vector $\boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{Y})$ and covariance matrix $\boldsymbol{\Sigma}(\boldsymbol{x}, \boldsymbol{Y})$. To ensure that the covariance matrix is positive definite and to keep the number of parameters linear in the number of dimensions of the latent space, the covariance matrix is constrained to be a diagonal matrix with positive values on the diagonal $\boldsymbol{\Sigma}(\boldsymbol{x}, \boldsymbol{Y}) = \text{diag}(\boldsymbol{\sigma}^2(\boldsymbol{x}, \boldsymbol{Y}))$.

$$q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{Y}) = \mathcal{N}\left(\boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{Y}), \boldsymbol{\Sigma}(\boldsymbol{x}, \boldsymbol{Y})\right) \tag{4.43}$$

By modeling both the prior and the recognition models with multivariate normal distributions we can find a closed-form solution to the Kullback-Leibler divergence of the two that is used in the loss function [Duc07; KW13]. The generation network (also called decoder) directly produces a trajectory from latent sample $\boldsymbol{z}$ and the vehicle representation $\boldsymbol{x}$. A reconstruction loss function evaluates the closeness between generated trajectory $\hat{\boldsymbol{Y}}_{1:T_{pred}}$ and target (ground truth) trajectory $\boldsymbol{Y}_{1:T_{pred}}$ (see Section 4.4.2). During training, given vehicle representation and target trajectory, the recognition network learns to predict mean and covariance parameters of the recognition model in such a way that the decoder is likely to generate a trajectory that matches the target trajectory, given $\boldsymbol{z}$ and $\boldsymbol{x}$ [Lee+17]. During testing we sample from the prior distribution $p(\boldsymbol{z})$. We can generate a diverse set of trajectories by sampling from the prior multiple times.

The encoder network uses a multilayer perceptron (MLP) composed of multiple feedforward layers with rectified linear unit (ReLU) activation functions. As input, it receives the concatenated vehicle representation vector and target trajectory vector. It produces mean and variance vector which parameterize the latent variable distribution. Then $\boldsymbol{z}_i$ is sampled $\boldsymbol{z}_i \sim \mathcal{N}\left(\boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{Y}), \boldsymbol{\Sigma}(\boldsymbol{x}, \boldsymbol{Y})\right)$. To allow for backpropagation through the sampling step we make use of the reparameterization trick presented in [KW13].

Since the target trajectory is composed of a fixed number of vehicle states we can model the decoder network using an MLP. The decoder does not directly produce a trajectory. Rather, the output from the decoder is used in a differentiable vehicle model which yields the final trajectory.

Differentiable Vehicle Model

We make use of the fact that vehicle motion is constrained by vehicle dynamics, i.e. not every possible motion can be performed by the vehicle because of non-holonomic constraints. If we were to naively predict trajectories by generating position updates for each time step, the generator network would have to learn motion constraints from data. While this is possible, it is not as sample efficient as encoding prior knowledge about vehicle dynamics into the

trajectory prediction network. We therefore interpret the output from the decoder network as a sequence of inputs to a vehicle model, which imposes kinematic constraints [Ma+19; Li+20a]. The vehicle model transforms the sequence of inputs into a sequence of resulting vehicle states. It is fully differentiable such that we can backpropagate gradients through it during training.

We use a relaxed version of the kinematic single-track vehicle model [AKM17] as our vehicle model. The model does not consider roll dynamics or tire slip, but since our dataset consists of urban traffic scenarios with a relatively low speed limit, these assumptions are reasonable. Instead of using the steering angle velocity $\dot{\delta}$ we use the steering angle δ as input to the model. Furthermore, steering and acceleration constraints are relaxed from parameters found in "CommonRoad: Vehicle Models"[1]. This is done to ensure that we do not excessively constrain the feasible vehicle motions as all vehicle motions found in our dataset must be reproducible using the vehicle model.

The decoder network produces a sequence of vehicle model inputs $\hat{\boldsymbol{H}}_{1:T_{pred}} = \{\hat{\boldsymbol{h}}_1, \dots, \hat{\boldsymbol{h}}_{T_{pred}}\}$, each specifying steering angle input $\gamma_i \in \mathbb{R}$ and longitudinal acceleration input $a_i \in \mathbb{R}$ for time step i.

$$\hat{\boldsymbol{h}}_i = \begin{pmatrix} \gamma_i & a_i \end{pmatrix}^\top,\ i \in 1, \dots, T_{pred} \tag{4.44}$$

The vehicle model maintains the transformation invariance inductive bias as inputs to the model are expressed in a vehicle-centered coordinate system. Because we are predicting vehicle states for a number of discrete time steps, we calculate a vehicle state delta for each time step. Let δ_i be the vehicle orientation, δ_{max} the maximum absolute steering angle, v_i the longitudinal velocity and a_{max} the maximum longitudinal acceleration. Δt denotes the fixed amount of time between two consecutive time steps. The v_i is lower-bounded by $v_{min} < 0$ and upper-bounded by $v_{max} > 0$. Refer to Table A.2 for the values of parameters used here.

$$\delta_i = \delta_{max} \tanh(\gamma_i) \tag{4.45}$$
$$\Delta v_i = a_{max} \tanh(a_i) \tag{4.46}$$
$$\tilde{v}_i = v_{i-1} + \Delta t \cdot \Delta v_i \tag{4.47}$$
$$v_i = \max\{v_{min}, \min\{\tilde{v}_i, v_{max}\}\} \tag{4.48}$$
$$\Delta\hat{\theta}_i = \frac{v_i}{l_{wb}} \tan(\delta_i) \tag{4.49}$$
$$\hat{\theta}_i = ((\hat{\theta}_{i-1} + \Delta t \cdot \Delta\hat{\theta}_i + \pi) \bmod 2\pi) - \pi \tag{4.50}$$
$$\Delta\hat{\boldsymbol{x}}_i = \begin{pmatrix} v_i \cos(\hat{\theta}_i) \\ v_i \sin(\hat{\theta}_i) \end{pmatrix} \tag{4.51}$$
$$\hat{\boldsymbol{x}}_i = \hat{\boldsymbol{x}}_{i-1} + \Delta t \cdot \Delta\hat{\boldsymbol{x}}_i \tag{4.52}$$

The output of the vehicle model is a sequence of vehicle states $\hat{\boldsymbol{y}}_i = \begin{pmatrix} \hat{\boldsymbol{x}}_i & \hat{\theta}_i \end{pmatrix}^\top$ where $\hat{\boldsymbol{x}}_i \in \mathbb{R}^2$ and $\hat{\theta}_i \in [-\pi, \pi)$ are the position and orientation of the vehicle at state i.

Trajectory Prediction Loss

The CVAE loss function is composed of two terms. The negative Kullback-Leibler divergence loss $\mathcal{L}_{\text{KL}}$ between the recognition and prior distributions:

$$\mathcal{L}_{\text{KL}} = -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})) = -D_{KL}\left(\mathcal{N}\left(\boldsymbol{\mu}(\boldsymbol{x}, \boldsymbol{Y}), \boldsymbol{\Sigma}(\boldsymbol{x}, \boldsymbol{Y})\right) \| \mathcal{N}(\mathbf{0}, \mathbf{Id})\right) \tag{4.53}$$

[1] Matthias Althoff and Gerald Würsching. "CommonRoad: Vehicle Models (Version 2020a)". https://gitlab.lrz.de/tum-cps/commonroad-vehicle-models. Visited June 16th 2022.

And the reconstruction loss $\mathcal{L}_{\text{recon}}$, which we define as the mean squared error (MSE) between predicted positions $\hat{\boldsymbol{x}}_i$ and true positions $\boldsymbol{x}_i$ as well as between predicted orientations $\hat{\theta}_i$ and true orientations θ_i:

$$\mathcal{L}_{\text{recon}} = \frac{1}{T}\sum_{i=1}^{T} \|\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i\|_2^2 + (\theta_i - \hat{\theta}_i)^2 \tag{4.54}$$

The complete trajectory prediction loss $\mathcal{L}_{\text{traj}}$ is the sum of the Kullback-Leibler divergence loss and the reconstruction loss.

$$\mathcal{L}_{\text{traj}} = \mathcal{L}_{\text{KL}} + \mathcal{L}_{\text{recon}} \tag{4.55}$$

See Figure 4.5 for an overview over the complete trajectory prediction network, including losses.

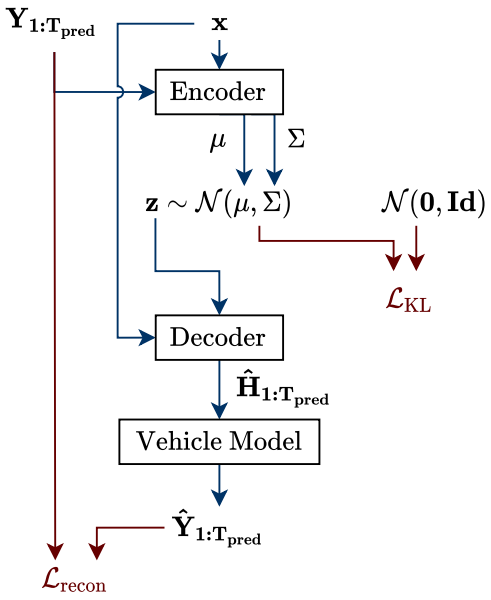

Figure 4.5: Architecture of the trajectory prediction network. The diagram shows the operation during training time. During test time the encoder is not used and the latent variable $\boldsymbol{z}$ is sampled from the prior standard multivariate normal distribution. Vehicle states $\boldsymbol{x}$ and the true trajectory $\boldsymbol{Y}_{1:T_{pred}}$ are inputs to the model. The model outputs a generated vehicle trajectory $\boldsymbol{Y}_{1:T_{pred}}$. Loss terms are colored dark red.

4.4.3 Prior Losses

The sparsity and temporal coherence inductive biases are encoded as loss functions over the computed representations. They are called prior losses because they encode constraints on the representation space derived from general-purpose prior knowledge [BCV13]. The sparsity prior is implemented in two ways: 1. by limiting the dimensionality of the representations. Low-dimensional representations have less capacity for storing information. 2. by penalizing the absolute value of each feature of the representation (L_1 loss). Not every factor that the vehicle representation can encode will be relevant in all traffic scenarios. By imposing a loss on non-zero features in the representation we ensure that only sufficiently relevant features are encoded. Relevancy in this context is the features' positive influence on task-specific losses such that they balance out the penalty imposed by the sparsity loss. The L_1 loss can also be interpreted as a regularizing term which allows us to moderately increase the

dimensionality without risking overfitting. In the following, $\boldsymbol{x}_i^{(t)}$ is representation of vehicle $i = 1, \ldots, N$ at time step $t = 1, \ldots, T$.

$$\mathcal{L}_{\text{sparse}} = \frac{1}{NT} \sum_{i=1}^{N} \sum_{t=1}^{T} \|\boldsymbol{x}_i^{(t)}\|_1 \tag{4.56}$$

The temporal coherency prior (also called slowness principle [Les+18]) encodes the requirement that vehicle representations of the same vehicle at consecutive time step should be similar. This encodes that fact that many properties of real-world scenes change gradually over time.

$$\mathcal{L}_{\text{tc}} = \frac{1}{N(T-1)} \sum_{i=1}^{N} \sum_{t=1}^{T-1} \|\boldsymbol{x}_i^{(t)} - \boldsymbol{x}_i^{(t+1)}\|_2 \tag{4.57}$$

4.4.4 Hybrid Loss

The loss function for the entire model is the composition of the prior, drivable area and trajectory prediction losses.

$$\mathcal{L} = \omega_1 \mathcal{L}_{\text{sparse}} + \omega_2 \mathcal{L}_{\text{tc}} + \omega_3 \mathcal{L}_{\text{driv}} + \omega_4 \mathcal{L}_{\text{traj}} \tag{4.58}$$

Loss weights $\omega_1, \ldots, \omega_4 \in \mathbb{R}$ are used to re-scale the individual losses to a similar range for the combined loss. They balance the relative importance of learning objectives [JB15]. Table A.2 lists values used during training.

Chapter 5

Evaluation

In this chapter we evaluate the previously defined model by training it on a large dataset and assessing various properties of the trained model. Since we created a custom traffic scenario dataset to fit our requirements, the first part of this chapter present our approach for collecting raw data and explains the various preprocessing steps we applied to produce our final benchmark dataset. Next, we briefly discuss the implementation of the model as well as the training process. Finally, the representations produced by the model are analyzed, and we evaluate the performance of our decoder heads.

5.1 Benchmark Dataset and Preprocessing

As part of this thesis we have worked on and contributed various parts of the *commonroad-geometric*[1] Python software framework. *commonroad-geometric* aims to simplify geometric deep learning research in the autonomous driving domain. It integrates CommonRoad scenario data with the PyTorch [Pas+19] deep learning and PyTorch Geometric [FL19] geometric deep learning frameworks. Additionally, it offers an interface (CR-SUMO [Kli+19]) to the SUMO traffic simulator [Lop+18] which is used for generating realistic traffic simulations for a CommonRoad scenario. In this work we utilize *commonroad-geometric* for collecting and preprocessing our benchmark dataset.

We collect a benchmark dataset of traffic scenarios consisting of road networks from urban areas in Europe with simulated vehicle traffic. In the following we present the individual steps we took when producing the dataset.

1. **Collecting map data:** In order to capture realistic road networks in the dataset we download map sections from OpenStreetMap (OSM) [HW08] using code that is based on the Globetrotter tool [Kli+20]. We wish to collect a diverse set of road geometries which allow for complex vehicle interactions. For this reason we collect map data from urban areas as urban roads are offer frequent opportunities for vehicle interactions. We select a number of European cities and randomly sample $200\,\mathrm{m} \times 200\,\mathrm{m}$ map sections from a circular region enclosing the city. Table A.3 lists the location and size of sample regions used for our benchmark dataset.

2. **Conversion to CommonRoad scenarios:** The downloaded OSM map sections are converted to CommonRoad scenarios by using the *OSM2CR* converter that is part of the CommonRoad scenario designer [MKA21] software package.

[1] Hosted at https://gitlab.lrz.de/cps/commonroad-geometric.

3. **Filtering out scenarios:** We find that the conversion from the OSM map format to the CommonRoad scenario format is not always error-free. In addition, some randomly sampled map sections only contain a very short or simple (i.e. straight) road segment. For these reasons we filter scenarios based on the following features:
   - number of lanelets in the scenario
   - total lanelet length
   - lanelet curvature
   - lanelet width
   - intersection count
   - number of parallel lanes

   This filtering step results in approximately 25% of scenarios being removed. Figure 5.1 shows example scenarios.

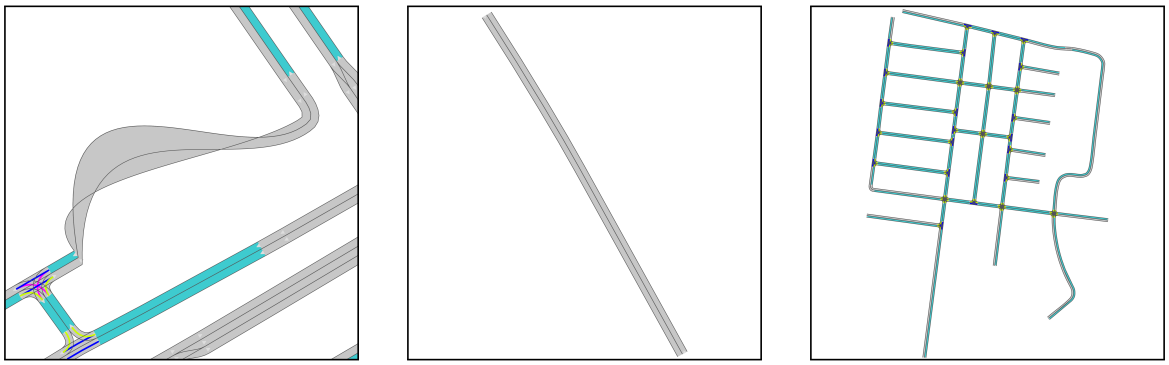

Figure 5.1: Examples of CommonRoad scenarios produced by *OSM2CR*. **Left:** Error in conversion step produces an invalid lanelet geometry. This scenario is discarded in the filtering step. **Middle:** Scenario consisting of a single straight road. This scenario is also discarded during the filtering step as it does not provide an interesting road geometry or opportunities for complex vehicle interactions. **Right:** Example of a scenario that is accepted by the filtering step.

4. **Construction of the lanelet graph:** The collected scenarios do not contain vehicle traffic, yet. However, the lanelet graph can already be constructed from the lanelet network. We present this process in detail in Section 4.1.2.

5. **Traffic simulation:** Next, the *commonroad-geometric* framework is used to simulate vehicle traffic for each scenario. *commonroad-geometric* uses the CR-SUMO bridge to perform the simulation with the SUMO simulator. Vehicles are spawned on random lanelets over the lifetime of the simulation. Data collection is started only after the scenario is sufficiently populated. Our heuristic for determining when data collection should begin is the first time that a vehicle exits the scenario through one of the outbound lanelets. Data collection then runs for $T = T_{obs} + T_{pred}$ time steps (see Table A.1).

6. **Construction of the temporal traffic scenario graph:** We now fuse simulated vehicle data with the previously generated lanelet graph to construct the temporal traffic scenario graph. Please refer to Section 4.1.3 for an in-depth description of this step. This step also includes rendering the drivable area masks.

7. **Feature normalization:** Machine learning algorithms in general and neural networks in particular can benefit from feature normalization [IS15], i.e. the transformation of values such that they have zero mean and unit variance over the dataset. For this reason we perform a feature normalization step which normalizes numerical node and edge

attributes. This step utilizes `StandardScaler` from the *scikit-learn* machine learning library [Ped+11].

Figure 5.2: Average drivable area (left) and closest sample to average (right) over the entire dataset. We observe a strong bias towards the drivable area mask of a straight two-lane road.

8. **Addressing imbalance in dataset:** In our experiments we observe overfitting of the drivable area decoder, which manifests itself in a mode collapse. This is caused by a severe imbalance in training data. Specifically, a significant portion of the ground-truth drivable area masks in the dataset show a straight two-lane road. This is to be expected as most roads exhibit a low local curvature, but still poses a problem for training. We can visualize the imbalance by plotting mean and median drivable area over the entire dataset (Figure 5.2) and by plotting a histogram of absolute difference (L_1 distance) of drivable area mask samples to the average drivable area mask (Figure 5.3). The imbalance is especially visible in the histogram.

   An option to rectify this imbalance is to eliminate duplicates from the dataset. However, this would result in the drivable area decoder and the corresponding loss function begin inactive for a significant portion of the dataset. The decoder component of our representation learning architecture is designed to guide the training of the encoder component by encoding desirable properties of the representation as prediction tasks. With an inactive drivable area prediction decoder, the training of the encoder would be biased towards only the trajectory prediction task.

   To avoid this problem we instead opt to use all available drivable area masks as training data but down weight the drivable area decoder loss for masks which are overrepresented in the dataset. With this approach we ensure that the drivable area decoder will always contribute to the training of the encoder component. This approach is implemented by assigned each drivable area mask sample a scalar weight $w_i \leqslant 1$. The weight value is computed based on the histogram (Figure 5.3). Samples in histogram bins which exceed a predefined size $n_i > \bar{n}_{hist}$, i.e. contribute a disproportionate number of similar samples, are assigned a weight $w_i = \min\left\{\frac{\bar{n}_{hist}}{n_i}, 1\right\}$ in order to assure $w_i n_i \leqslant \bar{n}_{hist}$. We empirically observe an improvement in prediction performance when applying this measure.

The complete dataset contains 42346 traffic scenarios.

5.2 Training

We utilize the PyTorch and PyTorch Geometric deep learning frameworks to define, train and evaluate our model. Refer to Table A.2 for details on the hyperparameters used for training.

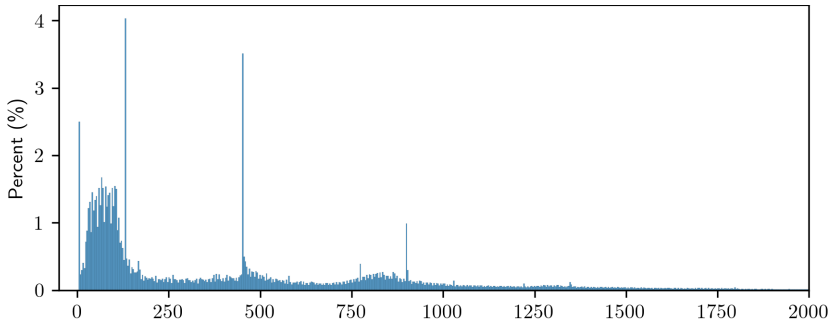

Figure 5.3: Histogram of drivable area masks in the dataset. The x axis represents L_1 distance to the mask that is most similar to the average drivable area mask (Figure 5.2, right). We find that a significant portion of the drivable area masks are very similar to the average sample, i.e. show a straight two-lane road. Additionally, we observe other peaks in the histogram which represent straight roads with a different number of lanes (not visualized).

Statistic	Training & Validation	Test
Temporal traffic scenario graphs	38549	3797
Average node count per graph	4620.6	4539.6
Average edge count per graph	15104.6	14820.7
Average lanelet length	16.6	16.6 m
Average total lanelet length per scenario	3581.9 m	3524.0 m
Average number of vehicles per time step	16.7	16.4

Table 5.1: Statistics on training and validation as well as test datasets.

Our benchmark dataset is partitioned into training, validation and test datasets. The training dataset contains 34694 samples, the validation dataset 3855 samples and the test dataset contains 3797 samples. Table 5.1 presents statistics on the datasets. We use the Optuna framework [Aki+19] for comparing different model architectures and optimizing hyperparameters based on the performance on the validation dataset. Finally, we train until convergence on the validation dataset and subsequently evaluate the performance of the trained model on the held-out test dataset. The progression of losses computed on the validation dataset over the course of training is plotted in Figure 5.4. Refer to Table A.1 for parameter values of the dataset used for training.

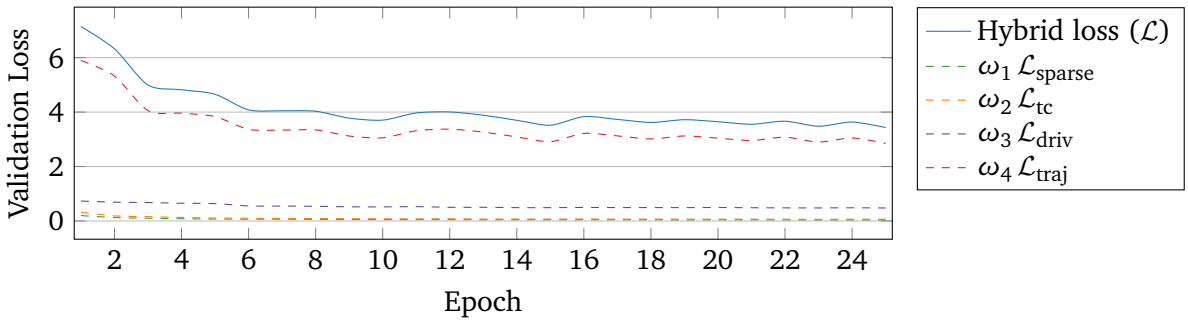

Figure 5.4: Loss values computed on the validation dataset, plotted for each epoch during training.

5.3 Results

In the following we analyze the vehicle representations produced by the encoder component of our trained model and also evaluate the performance of the decoders on their specific tasks. We perform both quantitative and qualitative evaluations. All results are produced with the test dataset, which has no overlap with the training dataset and can thus show ability of the architecture to generalize to new traffic scenarios.

5.3.1 Vehicle Representation

The predictiveness of the vehicle representation is determined by the performance of downstream prediction tasks. However, we can qualitatively evaluate the vehicle representation space by visually inspecting the distribution of embeddings. We make use Principal Component Analysis (PCA), a dimensionality reduction technique which defines an algorithm for determining an orthogonal linear transformation of observations such that the dimensions of observations in the transformed coordinate space have maximum variance and are pairwise linearly uncorrelated [JC16]. Dimensionality reduction is then performed in the transformed coordinate space, by retaining only the first k dimensions (principal components) which explain most of the variation present in the data.

For visualization purposes we reduce the dimensionality of the vehicle embedding vectors to two dimensions. The PCA-transformed vehicle embeddings of all vehicles in the test dataset are plotted in Figure 5.5. The distribution of embeddings in the two-dimensional space exhibits structure. We randomly sample from select regions in the lower-dimensional space and visualize ground truth drivable area masks corresponding to these samples in Figure 5.6. The regions are numbered and outlined in red in Figure 5.5. Drivable area masks of samples from region 1 (first row in the figure) seem to be biased towards left-turning lanelets, while samples from region 2 (second row) seem to show right-turning lanelets. We hypothesize that the 2nd PCA dimension correlates with the turning direction of the lanelet. Samples from region 3 seem to correspond to vehicles which are driving on a straight road (third row).

The last row of Figure 5.6 shows drivable area masks corresponding to vehicle embeddings sampled at regular intervals along the purple line connecting region 1 to 3, visualized in Figure 5.5. We can observe a relatively smooth progression from left- to right-turning lanelets.

5.3.2 Drivable Area Prediction

We have formulated drivable area prediction as a binary classification task (see Section 4.4.1). To evaluate the performance of the decoder head which performs the prediction we first determine statistics about the ground truth and predictions on the test dataset. True positive, false negative, false positive and true negative ratios are arranged in a confusion matrix (Table 5.2). From these values we can then derive summary statistic to measure the performance of the classifier. The classifier has an accuracy of 95.96 %, a precision of 93.28 % and a recall of 89.68 %. The F_1 score is 0.9144. This shows that the drivable area decoder is able to predict drivable area with good accuracy from the vehicle representation alone.

Complex road geometry, as for example found at complex intersections, still poses a challenge for the model. This can, on one hand, be attributed to the fact that vehicle embedding vectors are of limited size and thus have only limited capacity for encoding complex geometry. Increasing the dimensionality incurs a trade-off; particularly complex road geometries

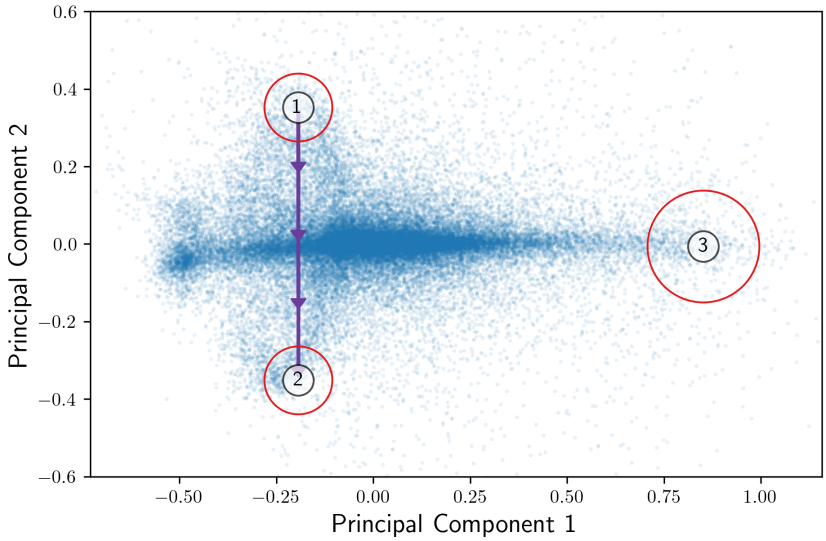

Figure 5.5: Visualization of the first two principal components of PCA-transformed vehicle representations.

	Predicted drivable	**Predicted non-drivable**
True drivable	21.57 %	2.48 %
True non-drivable	1.56 %	74.39 %

Table 5.2: Confusion matrix

might be reconstructed more accurately, but the overall efficiency decreases because more computations are necessary. On the other hand, we suspect that the training dataset might be a limiting factor because most drivable area masks depict a straight road and only a small set show more complex geometries. While the produced drivable area mask is not accurate enough to be used for path planning, it does illustrate that the vehicle embedding vector encodes significant information about the local road network. Multiple examples of predictions from our model are provided in Figure A.1 in the appendix.

5.3.3 Trajectory Prediction

We evaluate the trajectory prediction decoder of our proposed model quantitatively by comparing with a baseline vehicle model.

Baseline Vehicle Model

We compare our trajectory prediction performance with a parameter-free baseline vehicle model, the Constant Velocity (CV) model. The CV model assumes that each vehicle keeps moving with the same velocity as in the last observed time step.

Metrics

Trajectory prediction results are reported in terms of displacement error (DE), average displacement error (ADE) and final displacement error (FDE), as is done in previous works [Ala+16; MXL20; Li+20a; MXL21]. The displacement error for time step t is the Euclidean distance between predicted $\hat{\boldsymbol{x}}_t$ and ground truth $\boldsymbol{x}_t$ vehicle positions.

$$DE(\boldsymbol{x}_t, \hat{\boldsymbol{x}}_t) = \|\boldsymbol{x}_t - \hat{\boldsymbol{x}}_t\|_2 \tag{5.1}$$

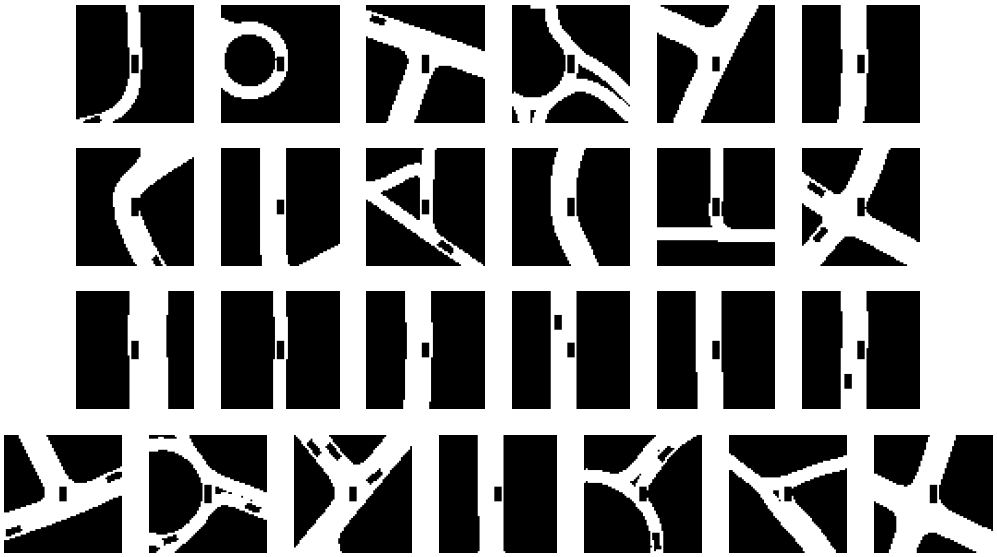

Figure 5.6: The first, second and third rows show ground truth drivable area masks corresponding to samples from region 1, 2 and 3, respectively. The bottom row shows drivable area masks corresponding to samples taken along the purple line in Figure 5.5.

ADE and FDE are defined in terms of the displacement error. Given predicted and true trajectories $\hat{\boldsymbol{x}}_{1:T_{pred}}$ and $\boldsymbol{x}_{1:T_{pred}}$, the ADE is the average displacement error over all time steps while the FDE is the displacement error at the final predicted time step ($t = 3\,\mathrm{s}$):

$$ADE(\boldsymbol{x}_{1:T_{pred}}, \hat{\boldsymbol{x}}_{1:T_{pred}}) = \frac{1}{T_{pred}} \sum_{t=1}^{T_{pred}} DE(\boldsymbol{x}_t, \hat{\boldsymbol{x}}_t) \tag{5.2}$$

$$FDE(\boldsymbol{x}_{1:T_{pred}}, \hat{\boldsymbol{x}}_{1:T_{pred}}) = DE(\boldsymbol{x}_{T_{pred}}, \hat{\boldsymbol{x}}_{T_{pred}}) \tag{5.3}$$

The overall DE, ADE and FDE metrics are computed as the average over all trajectory predictions.

Comparison to Baseline

The performance of our model is compared to the baseline model on the test dataset. Since our model generates multiple trajectory hypotheses, we average the displacement error over 20 different predictions. The results are plotted in Figure 5.7. The Constant Velocity model achieves an ADE of 4.07 m and FDE of 10.33 m while our model achieves an ADE of 1.23 m and FDE of 2.97 m. This constitutes a 69.8 % improvement over the baseline ADE and a 71.3 % improvement over the baseline FDE. Thus, our model performs significantly better than the baseline. Multiple predicted trajectories are provided in Figure A.2 in the appendix.

5.3.4 Ablation Study

We perform ablation studies to evaluate how the information from the traffic scenario graphs and select parts of our encoder architecture contribute to the performance of the system. To reduce overall training time, all model variants are trained on a smaller dataset (15419 traffic scenario graphs). Each variant is trained for 15 epochs. The model checkpoint from the epoch which produces the lowest loss on the validation dataset is used to compute metrics. Reported metrics are the best epoch the loss on the validation dataset as well as the F_1 score of

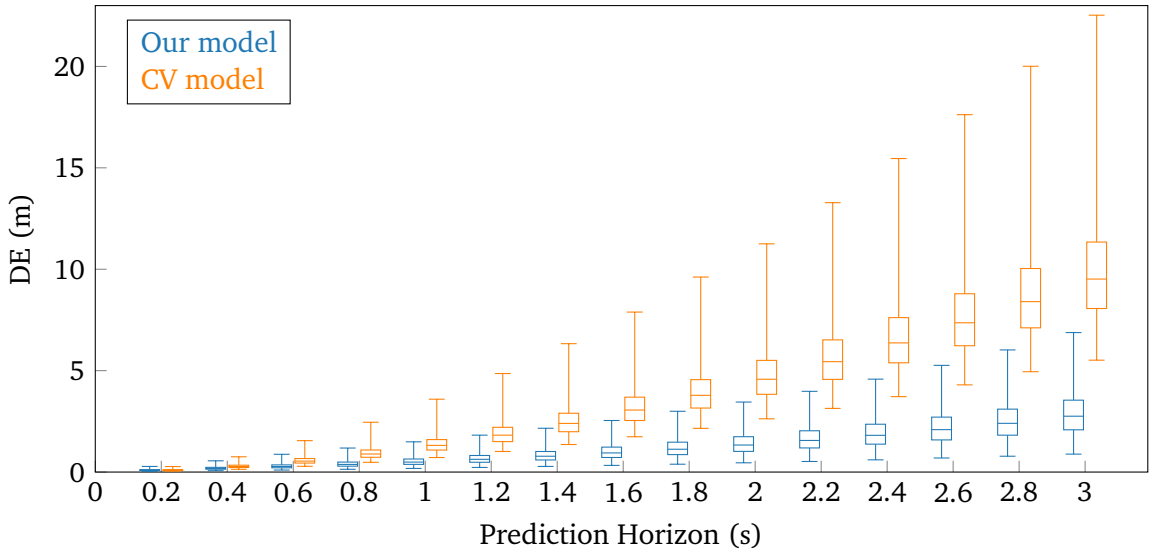

Figure 5.7: Box plot of displacement error of predicted trajectories over the prediction horizon. Boxes are slightly offset on the x axis such that they do not overlap. Lower and upper whisker illustrate the 2nd and 98th percentile, outliers are not shown.

the drivable area binary classification and the average ADE and FDE of predicted trajectories for each of the model variants. The following variants are compared:

- *Baseline*: No changes to the neural network architecture or traffic scenario graphs, used as a baseline for comparing metrics.
- *Without lanelet-vehicle edges*: All lanelet-vehicle edges are removed from the traffic scenario graph. This prevents the graph neural network from incorporating road network information into the vehicle representation.
- *Without temporal vehicle edges*: All temporal vehicle edges are removed from the traffic scenario graph prior to processing. Thus, the vehicle representations can no longer encode information about previous time steps which we expect will negatively affect the trajectory prediction.
- *Without edge attributes*: All edge attributes are removed. Both spatial and temporal relationships are described by edge attributes, we expect this change to severely impact both decoders similarly as if both temporal vehicle and lanelet-vehicle edges had been removed.
- *Less GNN layers*: The number GNN layers is reduced from 8 to 4. Reducing the number of message passing steps reduces the node neighborhood size from which information can be aggregated. Since vehicle nodes are densely connected, we expect this change to primarily influence the ability of the drivable area decoder to predict distant lanelets.

The results of the ablation study are presented in Table 5.3.

The variant without lanelet-vehicle edges performs the worst overall; loss, F_1 score and accuracy of predicted trajectories are all individually significantly worse than any other variant. Interestingly, it is, in some cases, still able to approximately predict the road geometry. We suspect that the encoder network can partially compensate for the lack of road network features by making use of vehicle position information contained in vehicle and temporal vehicle edges, as vehicle trajectories follow the road geometry. The predicted trajectories also

Method	Epoch	Loss	F_1 score	ADE / FDE
Baseline	15	4.276	0.8892	1.36 / 3.21
W/o lanelet-vehicle edges	12	13.505 (↑ 216 %)	0.7923 (↓ 11 %)	2.83 (↓ 107 %) / 6.93 (↓ 116 %)
W/o temporal vehicle edges	15	4.216 (↓ 1 %)	0.8978 (↑ 1 %)	1.42 (↓ 4 %) / 3.30 (↓ 3 %)
W/o edge attributes	14	7.753 (↑ 81 %)	0.8419 (↓ 5 %)	1.89 (↓ 39 %) / 4.67 (↓ 45 %)
Less GNN layers	12	5.677 (↑ 33 %)	0.8859 (-)	1.58 (↓ 16 %) / 3.71 (↓ 16 %)

Table 5.3: Results of the ablation study.

substantially degrade in quality, which demonstrates that the trajectory prediction system learns to utilize and depends on the road network graph.

The results of the variant without temporal vehicle edges are very similar to that of the baseline. As expected, the drivable area prediction results are not affected by the removal of temporal edges. We assume trajectory prediction results are only slightly less accurate because vehicle node attributes contain current vehicle velocity and acceleration values which obviates the need to compute them using past vehicle states. However, past vehicle states can convey factors (for example driver intent) that are not otherwise available in the traffic scenario graph. For this reason we suspect that these factors might not be sufficiently captured by the traffic predictor. This can be addressed in future work.

The variant without access to edge attributes is still able to predict the approximate drivable area, even without spatial relationship information encoded in edge attributes. This can partially be explained by the fact that lanelet geometry is still accessible as it is stored in lanelet node attributes.

The model variant with less graph neural network layers has an only slightly lower F_1 score than the baseline model. Either the baseline model is unable to propagate information over more than 4 hops or the drivable area masks, due to their limited spatial extent, mainly show lanelets that are within 3 hops from the ego vehicles' lanelet.

Chapter 6

Future Work

Reinforcement Learning

Predictive state representations can benefit the training of policy learning algorithms like reinforcement learning (RL) [MKB16; Les+18]. Reinforcement learning is concerned with learning a policy π which maps the current state of the environment to an action which affects the state such that future rewards are maximized [SB18]. A RL agent continuously interacts with its environment by first observing the environment state o_t, computing a state representation from the observation via an observation-state mapping $s_t = \phi(o_t)$ and finally acting by mapping the state to an action $a_t = \pi(s_t)$ [JB14].

A pre-trained feature extractor like our encoder network can potentially improve both training speed and performance of the reinforcement learning process when used as the observation-state mapping $\phi(o_t)$. Our encoder network produces a low-dimensional continuous embedding vector representing the current vehicle state. The embedding vector is computed from a complex graph representation of the traffic scene. Because our proposed architecture can be (pre-)trained separately from the reinforcement learning agent, without incurring computational overhead for simulating actions taken in the environment, the training process is more efficient. The predictive tasks performed by the decoder heads of our model during training align well with the task of RL for control of autonomous vehicles.

For these reasons we believe that our model is well suited to be used in reinforcement learning. Future work could evaluate the usefulness of extracted representations for improving the RL training process.

Evaluation on Public Datasets

In our work we have focused on collecting, preprocessing and finally training with our own custom traffic scenario dataset. While this allows us to be very flexible and model the graph representation used in the dataset ourselves, it also prevents us from comparing the performance of our model to similar models from the literature. In addition, ensuring that the data in our dataset is mostly correct and realistic is a time-intensive process. By simulating vehicle traffic we forgo some diversity in vehicle behavior as simulated vehicles tend to avoid critical situations.

In future work we can adapt our mode and training code to be compatible with public trajectory prediction datasets such as the INTERACTION [Zha+19], nuplan [Cae+21] or Waymo [Sun+20] datasets. These datasets also bring new interesting challenges such as containing trajectories from heterogeneous agents; for example cars, cyclists and pedestrians as in the Waymo dataset.

Transfer Learning

As noted earlier, representations are the basis for many downstream machine learning tasks. We model inductive biases, prediction tasks and loss functions with the goal of incorporating as much relevant context into vehicle representations as possible. Future work could evaluate whether the representations produced by our graph neural network encoder can be directly used for clustering or classification of traffic scenarios [Gru+17; HBG20] or trajectories of traffic participants [Yao+17].

Chapter 7

Conclusion

We have systematically designed a unified traffic scenario graph representation that fuses information about vehicles, past vehicle states and the road network. In addition, we have proposed a graph neural network-based encoder architecture that leverages the relational structure of the traffic scenario graph to compute vehicle representations and a decoder composed of two prediction tasks to incentivize the system to learn predictive representations. The drivable area predictor reconstructs the drivable area in proximity to the vehicle while the trajectory predictor generates realistic future trajectories by combining the Conditional Variational Autoencoder architecture with a differentiable vehicle model that applies kinematic constraints.

The results from evaluating the system on a dataset of real-world road networks show that the proposed architecture is able to compute a compact and predictive vehicle representation that encodes both local context and can be used to predict future vehicle states. In most cases the representations alone are sufficient for capturing and reconstructing drivable area around a vehicle as well as for modeling a distribution over realistic future trajectories.

Graphs as structured representations for traffic scenarios and graph neural networks as a specialized neural network architecture for structured processing of graph data have demonstrated their suitability for the representation learning task.

We can conclude that the framework is able to learn predictive representation from data and can generalize to new traffic scenarios. Future work can utilize the representations in downstream tasks such as deep reinforcement learning methods that learn control policies for autonomous vehicles or clustering of traffic scenarios.

Appendix A

Appendix

A.1 Decoder Output Samples

We visualize 18 drivable area samples from the test dataset in Figure A.1. The predictions from our model are plotted alongside the corresponding ground truth drivable area masks. In addition, we visualize trajectory predictions in Figure A.2.

A.2 Dataset and Model Parameters

Table A.1 summarizes parameter values used during preprocessing of the dataset. Table A.2 shows parameter values used in the neural network architecture as well as during training. Table A.3 lists locations from which OpenStreetMap road network data for the dataset was collected.

Parameter	**Value**
Lanelet graph	
Maximum lanelet length	20 m
Traffic graph	
Vehicle edge distance threshold (D_{max}^{V2V})	42 m
Time step size (Δt)	200 ms
Time steps per traffic scenario graph (T)	20 $\widehat{=}$ 4 s
Observed time horizon (T_{obs})	5 $\widehat{=}$ 1 s
Prediction time horizon (T_{pred})	15 $\widehat{=}$ 3 s
Maximum temporal edge distance (T_{max}^{VTV})	4 time steps
Drivable area	
Prediction area centered at ego vehicle (D^{DA})	32 m $\times$ 32 m
Drivable area image size (R^{DA})	64 px $\times$ 64 px

Table A.1: Parameter values used during preprocessing of the dataset.

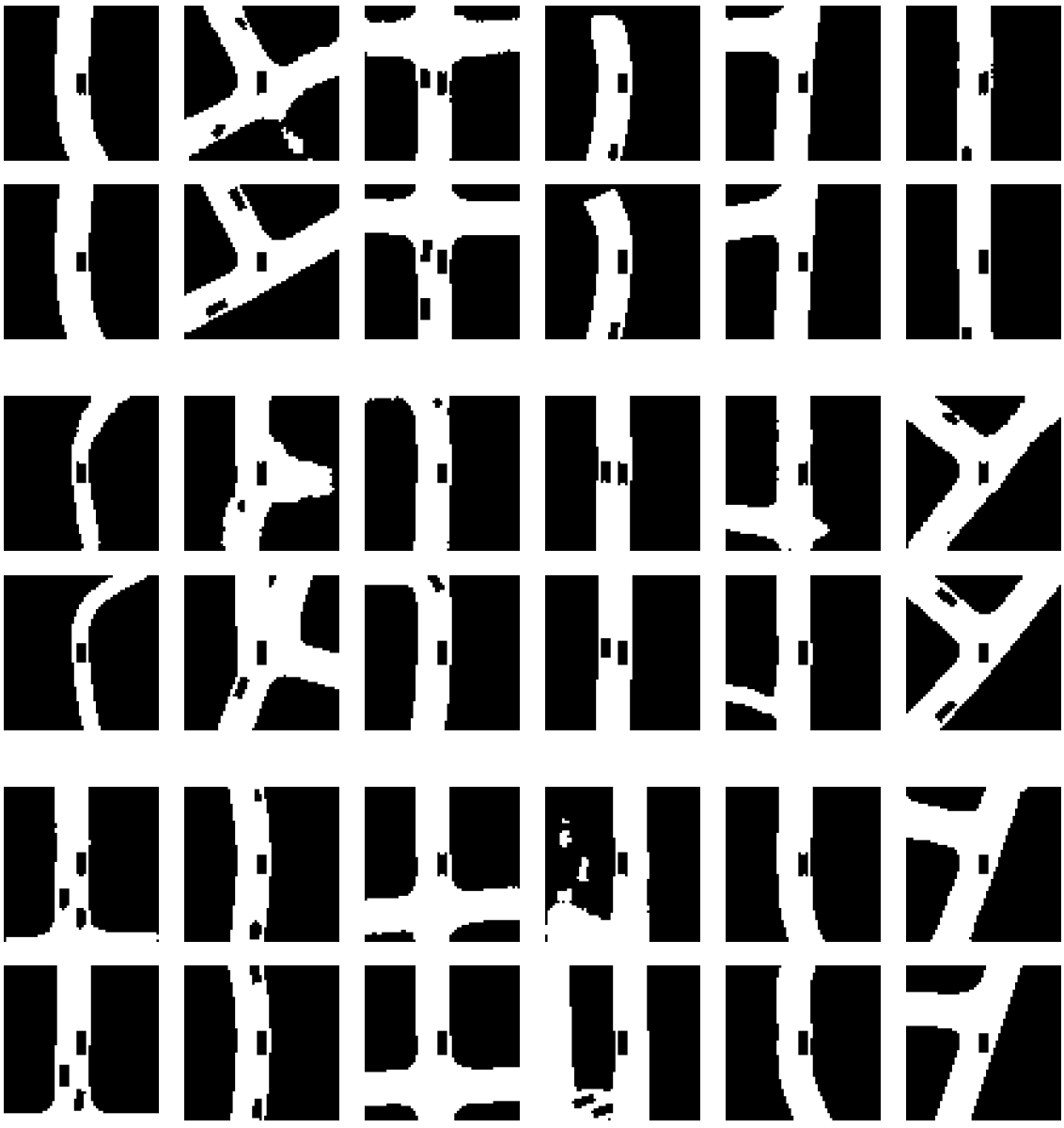

Figure A.1: Sample prediction from the drivable area decoder, shown together with ground truth drivable area masks. Three groups of predictions and ground truth data are visualized. The top rows in each group show binary predictions from our model while the bottom rows show corresponding binary ground truth masks.

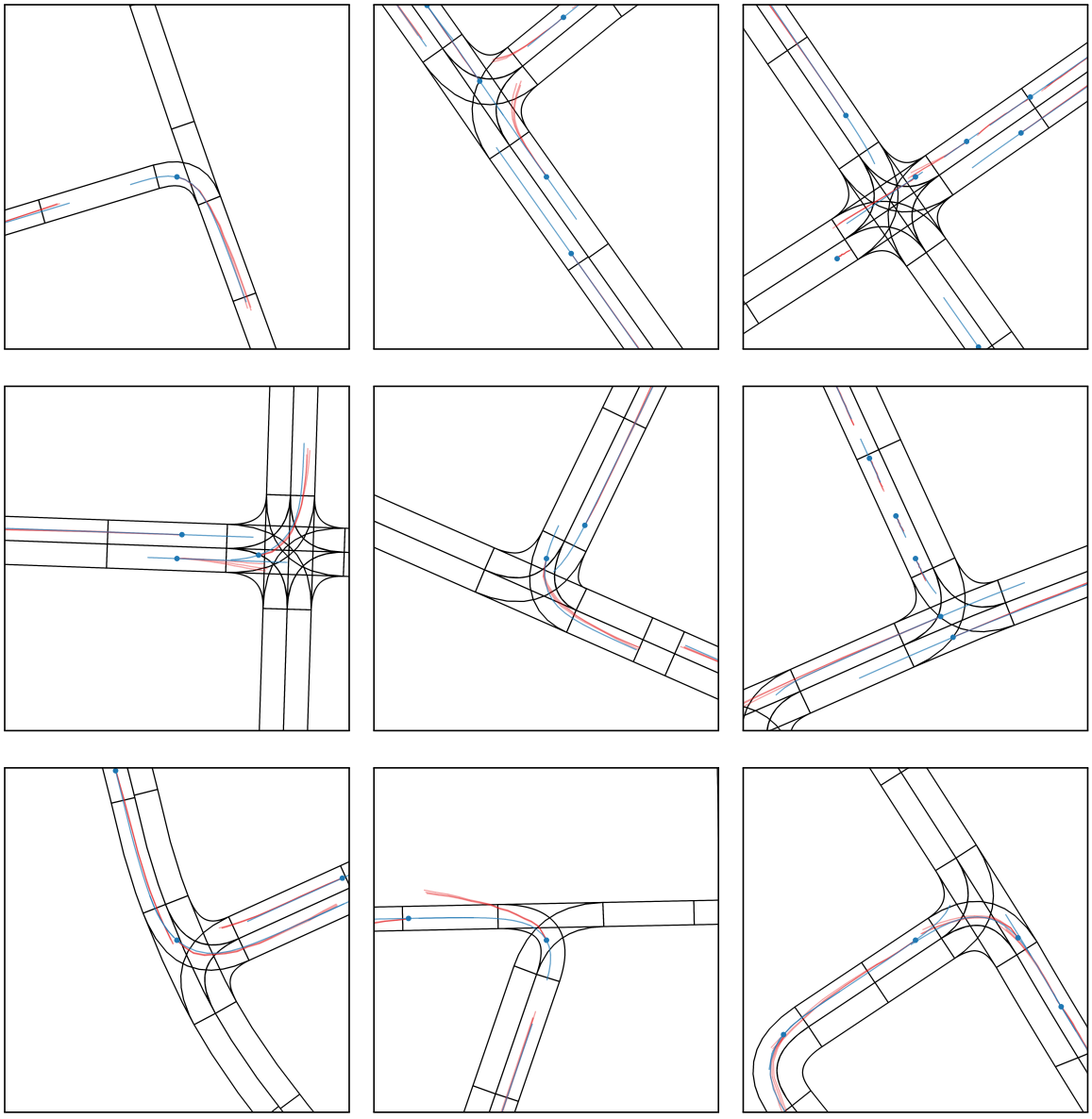

Figure A.2: Sample trajectory predictions from the trajectory decoder. Ground truth trajectory are visualized as blue polylines, trajectory prediction hypotheses are visualized as red polylines.

Parameter	Value
Scenario encoder	
Lanelet bound encoder	GRU [Cho+14], 64 hidden units
Temporal edge time vector dimensions	12
Node type embedding dimensions	10
Lanelet node feature dimensions	128
Vehicle node feature dimensions	128
Graph convolutional layers (L_{GNN})	8
GNN attention channels	128
GNN attention heads	16
GNN global context vector dimensions	128
Trajectory prediction	
CVAE latent dimensions	16
CVAE generator	MLP, 3 hidden layers, 256 hidden dimensions
Maximum steering angle (δ_{max})	$\frac{\pi}{2}$ rad
Maximum acceleration (a_{max})	$11.5\,\mathrm{m\,s^{-2}}$
Minimum longitudinal velocity (v_{min})	$-13.89\,\mathrm{m\,s^{-1}}$
Maximum longitudinal velocity (v_{max})	$33.33\,\mathrm{m\,s^{-1}}$
Hybrid Loss	
Sparsity loss weight (ω_1)	0.05
Temporal coherency loss weight (ω_2)	2.0
Drivable area loss weight (ω_3)	11.2
Trajectory prediction loss weight (ω_4)	1.0
Training	
Train / validation split	90% / 10%
Minibatch size	1
Optimizer	Adam [KB14], $\gamma = 0.000129$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$

Table A.2: Parameters used for the neural network architecture and for training.

City	Center coordinates	Sampling radius	Sample count
Berlin, Germany	52.5036, 13.4107	10 km	7850
Cologne, Germany	50.9383, 6.9531	6 km	2850
Hamburg, Germany	53.5550, 9.9951	10 km	7850
Lisbon, Portugal	38.7666, −9.1853	9 km	6360
Madrid, Spain	40.4152, −3.6941	8 km	5000
Munich, Germany	48.1476, 11.5589	8 km	10000
Oslo, Norway	59.9285, 10.7611	5 km	1964
Paris, France	48.8701, 2.3751	12 km	11300
Stockholm, Sweden	59.3168, 18.0332	6 km	2820

Table A.3: Locations from which OpenStreetMap road network data for the dataset was collected.

Bibliography

[Aki+19] Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. "Optuna: A next-generation hyperparameter optimization framework". In: *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 2019, pp. 2623–2631.

[Ala+16] Alahi, A., Goel, K., Ramanathan, V., Robicquet, A., Fei-Fei, L., and Savarese, S. "Social lstm: Human trajectory prediction in crowded spaces". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 961–971.

[AKM17] Althoff, M., Koschi, M., and Manzinger, S. "CommonRoad: Composable benchmarks for motion planning on roads". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 719–726.

[Bat+18] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. "Relational inductive biases, deep learning, and graph networks". In: *arXiv preprint arXiv:1806.01261* (2018).

[BZS14] Bender, P., Ziegler, J., and Stiller, C. "Lanelets: Efficient map representation for autonomous driving". In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. IEEE. 2014, pp. 420–425.

[BCV13] Bengio, Y., Courville, A., and Vincent, P. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[BSG11] Boots, B., Siddiqi, S. M., and Gordon, G. J. "Closing the learning-planning loop with predictive state representations". In: *The International Journal of Robotics Research* 30.7 (2011), pp. 954–966.

[BAY21] Brody, S., Alon, U., and Yahav, E. "How attentive are graph attention networks?" In: *arXiv preprint arXiv:2105.14491* (2021).

[Bro+21] Bronstein, M. M., Bruna, J., Cohen, T., and Veličković, P. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. 2021. DOI: 10.48550/ARXIV.2104.13478. URL: https://arxiv.org/abs/2104.13478.

[Bro+20] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[Bru+13] Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. "Spectral networks and locally connected networks on graphs". In: *arXiv preprint arXiv:1312.6203* (2013).

[Cae+21] Caesar, H., Kabzan, J., Tan, K. S., Fong, W. K., Wolff, E., Lang, A., Fletcher, L., Beijbom, O., and Omari, S. "nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles". In: *arXiv preprint arXiv:2106.11810* (2021).

[CC21] Chen, J. and Chen, H. “Edge-featured graph attention network”. In: *arXiv preprint arXiv:2101.07671* (2021).

[Cho+14] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. “On the properties of neural machine translation: Encoder-decoder approaches”. In: *arXiv preprint arXiv:1409.1259* (2014).

[De +18] De Bruin, T., Kober, J., Tuyls, K., and Babuška, R. “Integrating state representation learning into deep reinforcement learning”. In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1394–1401.

[DBV16] Defferrard, M., Bresson, X., and Vandergheynst, P. “Convolutional neural networks on graphs with fast localized spectral filtering”. In: *Advances in neural information processing systems* 29 (2016).

[Dev+18] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: https://arxiv.org/abs/1810.04805.

[Die+19] Diehl, F., Brunner, T., Le, M. T., and Knoll, A. “Graph neural networks for modelling traffic participant interaction”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 695–701.

[Doe16] Doersch, C. *Tutorial on Variational Autoencoders*. 2016. DOI: 10.48550/ARXIV.1606.05908. URL: https://arxiv.org/abs/1606.05908.

[Duc07] Duchi, J. “Derivations for linear algebra and optimization”. In: *Berkeley, California* 3.1 (2007), pp. 2325–5870.

[DB20] Dwivedi, V. P. and Bresson, X. “A generalization of transformer networks to graphs”. In: *arXiv preprint arXiv:2012.09699* (2020).

[FL19] Fey, M. and Lenssen, J. E. “Fast graph representation learning with PyTorch Geometric”. In: *arXiv preprint arXiv:1903.02428* (2019).

[Gil+17] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. “Neural message passing for quantum chemistry”. In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[GMS05] Gori, M., Monfardini, G., and Scarselli, F. “A new model for learning in graph domains”. In: *Proceedings. 2005 IEEE international joint conference on neural networks*. Vol. 2. 2005. 2005, pp. 729–734.

[Gru+17] Gruner, R., Henzler, P., Hinz, G., Eckstein, C., and Knoll, A. “Spatiotemporal representation of driving scenarios and classification using neural networks”. In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 1782–1788.

[HW08] Haklay, M. and Weber, P. “Openstreetmap: User-generated street maps”. In: *IEEE Pervasive computing* 7.4 (2008), pp. 12–18.

[HYL17] Hamilton, W., Ying, Z., and Leskovec, J. “Inductive representation learning on large graphs”. In: *Advances in neural information processing systems* 30 (2017).

[HVG11] Hammond, D. K., Vandergheynst, P., and Gribonval, R. “Wavelets on graphs via spectral graph theory”. In: *Applied and Computational Harmonic Analysis* 30.2 (2011), pp. 129–150.

[HBG20] Harmening, N., Biloš, M., and Günnemann, S. “Deep representation learning and clustering of traffic scenarios”. In: *arXiv preprint arXiv:2007.07740* (2020).

[He+16] He, K., Zhang, X., Ren, S., and Sun, J. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[HSP19] Hong, J., Sapp, B., and Philbin, J. “Rules of the road: Predicting driving behavior with a convolutional model of semantic interactions”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8454–8462.

[Hu+20] Hu, Z., Dong, Y., Wang, K., and Sun, Y. “Heterogeneous graph transformer”. In: *Proceedings of The Web Conference 2020*. 2020, pp. 2704–2710.

[IS15] Ioffe, S. and Szegedy, C. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[JCK20] Jeon, H., Choi, J., and Kum, D. “Scale-net: Scalable vehicle trajectory prediction network under random number of interacting vehicles via edge-enhanced graph convolutional neural network”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 2095–2102.

[JC16] Jolliffe, I. T. and Cadima, J. “Principal component analysis: a review and recent developments”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374.2065 (2016), p. 20150202.

[JB14] Jonschkowski, R. and Brock, O. “State Representation Learning in Robotics: Using Prior Knowledge about Physical Interaction.” In: *Robotics: Science and systems*. 2014.

[JB15] Jonschkowski, R. and Brock, O. “Learning state representations with robotic priors”. In: *Autonomous Robots* 39.3 (2015), pp. 407–428.

[Kaz+19] Kazemi, S. M., Goel, R., Eghbali, S., Ramanan, J., Sahota, J., Thakur, S., Wu, S., Smyth, C., Poupart, P., and Brubaker, M. “Time2vec: Learning a vector representation of time”. In: *arXiv preprint arXiv:1907.05321* (2019).

[KB14] Kingma, D. P. and Ba, J. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[KW13] Kingma, D. P. and Welling, M. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[Kin17] Kingma, D. P. “Variational inference & deep learning: A new synthesis”. In: (2017).

[Kin+14] Kingma, D. P., Mohamed, S., Jimenez Rezende, D., and Welling, M. “Semi-supervised learning with deep generative models”. In: *Advances in neural information processing systems* 27 (2014).

[KW16] Kipf, T. N. and Welling, M. “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[Kit+12] Kitani, K. M., Ziebart, B. D., Bagnell, J. A., and Hebert, M. “Activity forecasting”. In: *European conference on computer vision*. Springer. 2012, pp. 201–214.

[Kli+19] Klischat, M., Dragoi, O., Eissa, M., and Althoff, M. “Coupling sumo with a motion planning framework for automated vehicles”. In: *SUMO User Conference*. 2019, pp. 1–9.

[Kli+20] Klischat, M., Liu, E. I., Holtke, F., and Althoff, M. “Scenario factory: Creating safety-critical traffic scenarios for automated vehicles”. In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2020, pp. 1–7.

[Kre13] Kreyszig, E. *Differential geometry*. Courier Corporation, 2013.

[KL51] Kullback, S. and Leibler, R. A. “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[LB+95] LeCun, Y., Bengio, Y., et al. “Convolutional networks for images, speech, and time series”. In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[Lee+17] Lee, N., Choi, W., Vernaza, P., Choy, C. B., Torr, P. H., and Chandraker, M. “Desire: Distant future prediction in dynamic scenes with interacting agents”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 336–345.

[LVL14] Lefèvre, S., Vasquez, D., and Laugier, C. “A survey on motion prediction and risk assessment for intelligent vehicles”. In: *ROBOMECH journal* 1.1 (2014), pp. 1–14.

[Les+18] Lesort, T., Dıaz-Rodrıguez, N., Goudou, J.-F., and Filliat, D. “State representation learning for control: An overview”. In: *Neural Networks* 108 (2018), pp. 379–392.

[Li+20a] Li, J., Ma, H., Zhang, Z., and Tomizuka, M. “Social-wagdat: Interaction-aware trajectory prediction via wasserstein graph double-attention network”. In: *arXiv preprint arXiv:2002.06241* (2020).

[Li+20b] Li, J., Yang, F., Tomizuka, M., and Choi, C. “Evolvegraph: Multi-agent trajectory prediction with dynamic relational reasoning”. In: *Advances in neural information processing systems* 33 (2020), pp. 19783–19794.

[LYC19] Li, X., Ying, X., and Chuah, M. C. “Grip: Graph-based interaction-aware trajectory prediction”. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2019, pp. 3960–3966.

[Lop+18] Lopez, P. A., Behrisch, M., Bieker-Walz, L., Erdmann, J., Flötteröd, Y.-P., Hilbrich, R., Lücken, L., Rummel, J., Wagner, P., and Wießner, E. “Microscopic traffic simulation using sumo”. In: *2018 21st international conference on intelligent transportation systems (ITSC)*. IEEE. 2018, pp. 2575–2582.

[Ma+19] Ma, H., Li, J., Zhan, W., and Tomizuka, M. “Wasserstein generative learning with kinematic constraints for probabilistic interactive driving behavior prediction”. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 2477–2483.

[MKA21] Maierhofer, S., Klischat, M., and Althoff, M. “Commonroad scenario designer: An open-source toolbox for map conversion and scenario creation for autonomous vehicles”. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2021, pp. 3176–3182.

[MXL20] Mo, X., Xing, Y., and Lv, C. “Recog: A deep learning framework with heterogeneous graph for interaction-aware trajectory prediction”. In: *arXiv preprint arXiv:2012.05032* (2020).

[MXL21] Mo, X., Xing, Y., and Lv, C. “Heterogeneous edge-enhanced graph attention network for multi-agent trajectory prediction”. In: *arXiv preprint arXiv:2106.07161* (2021).

[MKB16] Munk, J., Kober, J., and Babuška, R. “Learning state representation for deep actor-critic control”. In: *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE. 2016, pp. 4667–4673.

[Pas+19] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).

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

[RMC15] Radford, A., Metz, L., and Chintala, S. "Unsupervised representation learning with deep convolutional generative adversarial networks". In: *arXiv preprint arXiv:1511.06434* (2015).

[RMW14] Rezende, D. J., Mohamed, S., and Wierstra, D. "Stochastic backpropagation and approximate inference in deep generative models". In: *International conference on machine learning*. PMLR. 2014, pp. 1278–1286.

[San+21] Sanchez-Lengeling, B., Reif, E., Pearce, A., and Wiltschko, A. B. "A Gentle Introduction to Graph Neural Networks". In: *Distill* (2021). DOI: 10.23915/distill.00033.

[Sca+08] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. "The graph neural network model". In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[Shi+16] Shi, W., Caballero, J., Theis, L., Huszar, F., Aitken, A., Ledig, C., and Wang, Z. "Is the deconvolution layer the same as a convolutional layer?" In: *arXiv preprint arXiv:1609.07009* (2016).

[SLY15] Sohn, K., Lee, H., and Yan, X. "Learning structured output representation using deep conditional generative models". In: *Advances in neural information processing systems* 28 (2015).

[Sun+20] Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., et al. "Scalability in perception for autonomous driving: Waymo open dataset". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2446–2454.

[SB18] Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

[Vas+17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[Vel+17] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. "Graph attention networks". In: *arXiv preprint arXiv:1710.10903* (2017).

[VPT16] Vondrick, C., Pirsiavash, H., and Torralba, A. "Anticipating visual representations from unlabeled video". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 98–106.

[Wal+16] Walker, J., Doersch, C., Gupta, A., and Hebert, M. "An uncertain future: Forecasting from static images using variational autoencoders". In: *European Conference on Computer Vision*. Springer. 2016, pp. 835–851.

[Wan+19] Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., and Yu, P. S. "Heterogeneous graph attention network". In: *The world wide web conference*. 2019, pp. 2022–2032.

[Wu+20] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. “A comprehensive survey on graph neural networks”. In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[Yao+17] Yao, D., Zhang, C., Zhu, Z., Huang, J., and Bi, J. “Trajectory clustering via deep representation learning”. In: *2017 international joint conference on neural networks (IJCNN)*. IEEE. 2017, pp. 3880–3887.

[Yin+21] Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. “Do transformers really perform badly for graph representation?” In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 28877–28888.

[Zha+19] Zhan, W., Sun, L., Wang, D., Shi, H., Clausse, A., Naumann, M., Kümmerle, J., Königshof, H., Stiller, C., La Fortelle, A. de, and Tomizuka, M. “INTERACTION Dataset: An INTERnational, Adversarial and Cooperative moTION Dataset in Interactive Driving Scenarios with Semantic Maps”. In: *arXiv:1910.03088 [cs, eess]* (Sept. 2019).