CHALMERS
UNIVERSITY OF TECHNOLOGY

The Multimodal IMR-TrajNet

End-to-end Deep Ego Trajectory Prediction using Front-facing Camera Images and Standard Definition Maps

Master's thesis in Complex Adaptive Systems

NAPAT BHAHOLPOLBHAYUHASENA
ARVID LAVENO LING

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024

Master's thesis 2024

The Multimodal IMR-TrajNet

End-to-end Deep Ego Trajectory Prediction using Front-facing Camera Images and Standard Definition Maps

NAPAT BHAHOLPOLBHAYUHASENA
ARVID LAVENO LING

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2024

The Multimodal IMR-TrajNet
End-to-end Deep Ego Trajectory Prediction using Front-facing Camera Images and Standard Definition Maps
NAPAT BHAHOLPOLBHAYUHASENA
ARVID LAVENO LING

Supervisor: Jakob Vinkås, Zenseact
Examiner: Lars Hammarstrand, Department of Electrical Engineering

Master's Thesis 2024
Department of Electrical Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2024

The Multimodal IMR-TrajNet
End-to-end Deep Ego Trajectory Prediction using Front-facing Camera Images and Standard Definition Maps
NAPAT BHAHOLPOLBHAYUHASENA & ARVID LAVENO LING
Department of Electrical and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Ego trajectory prediction is crucial for the development of autonomous vehicles (AV), enabling them to navigate complex environments safely and efficiently. While High-definition (HD) maps are commonly used due to their detailed information, they come with high computational costs and scalability issues. Standard Definition (SD) maps, on the other hand, offer a more scalable and cost-effective solution, providing sufficient conceptual context to enhance prediction models. Moreover, these maps may also contain navigation routes that further indicate the future behavior of the AV. By integrating SD maps with front-facing camera images, we can capture the real-world scenario more accurately, addressing potential inaccuracies in the maps and improving the robustness of the prediction.

Our method explores different ways to extract and utilize features of SD maps for trajectory prediction, employing both Graph Neural Network (GNN)-based and transformer-based map methods for conducting this extraction. We tested various approaches for fusing the features from these modalities, including normal concatenation and cross-attention mechanisms, and for decoding the trajectories. Additionally, we introduced an auxiliary task to guide the model in learning more accurate trajectory shapes and utilized multimodal predictions to capture the inherent variability in possible driving paths.

The results demonstrate a significant improvement in trajectory prediction accuracy, with our approach achieving up to four times better performance compared to the baseline model that relies solely on visual data. The inclusion of SD maps and route information not only reduces errors but also enhances the model's robustness across various driving scenarios. These findings highlight the potential of using SD maps to advance autonomous vehicle technology.

Keywords: autonomous vehicles, deep learning, trajectory prediction, multimodal predictions, standardised maps, high-definition maps, graph neural networks, transformers, VectorNet.

Acknowledgements

We would like to extend our gratitude towards our supervisor at Zenseact, Jakob Vinkås, for greatly guiding the thesis from start to finish. We would also like to give a thanks to our examiner at Chalmers University of Technology, Lars Hammarstrand, for providing great insights in various aspects of the project.

Napat Bhaholpolbhayuhasena & Arvid Laveno Ling, Gothenburg, 2024-06-28

Contents

List of Figures

List of Tables

1

Introduction

Autonomous Vehicles (AVs) have emerged as a promising solution to reduce human error-related road crashes, which account for approximately 95% to 96% of all auto accidents [1]. A critical component of AV technology is the capability to predict the future behaviour of traffic participants. Of particular interest is the likely future behaviour of the autonomous vehicle itself, ego, which we refer to as ego trajectory prediction.

Ego trajectory prediction is fundamental to the operational safety and efficiency of AVs. It involves forecasting the future path of the vehicle based on its current state and understanding of the surrounding environment. This prediction is crucial for navigating complex traffic scenarios, avoiding collisions, and making informed decisions in real-time. The ability to anticipate future movements accurately ensures that AVs can coexist safely with other road users, including pedestrians, cyclists, and manually driven vehicles.

However, the task of accurately predicting the ego trajectory is fraught with challenges. It requires processing and interpreting vast amounts of data from various sources, understanding dynamic environmental factors, and dealing with the inherent unpredictability of road scenarios. Moreover, the possibility of integrating a multitude of input modalities provides a wide variety of scopes in which one may attempt to present a solution. In terms of input space, typical solutions may be grouped into three categories: *track-history*-based algorithms, where the track history concerns previous positions of the ego vehicle and surrounding agents; *birds-eye-view*-based (BEV) algorithms, that takes a representation of the current scene (expressed in a birds-eye view) as an input; and *raw-sensor-data*-based algorithms [2]. Within this thesis, we are interested in the class of solutions that consider raw data as an input. Specifically, camera data. Our hope is that the addition of maps to this modality may help inference of paths that move in the bordering domain of the camera's field of view, or for scenarios where the camera is partially obstructed.

1.1 Relevant work

This section aims to briefly map the current state-of-the-art methods, used for solving such trajectory prediction problems that have been formulated above. A typical sensor setup for a vision stack intended to be used for autonomous driving often combines camera data and LiDAR data to improve inference under conditions where one

or the other modality shows weaknesses. For singular modality with camera data, a common approach to infer trajectories is to feed the images through a CNN-based vision backbone [3]. Another approach is to cast the prediction problem into a two-step solution where an instance segmentation problem of the camera image is first solved, using a vision stack, and trajectory prediction is performed afterwards. In [4], the authors conduct a full-resolution segmentation of images from a variety of road scenes using a ResNet-like architecture[5].

Of more novelty is the class of methods that formulate the prediction problem as a downstream task where HD maps, perhaps produced by the aforementioned vision stack, are used to infer the trajectory. In [6], three main categories of how the maps are encoded within the network are identified; rasterized-based, graph-based and point cloud-based. In [7], the authors propose a CNN architecture, IntentNet, to extract road features from 2D-encodings of the dynamic maps, leveraging LiDAR. VectorNet [8], feeds vectorized versions of the maps into a GNN (graph neural network) in order to learn the scene context. Similarly, LaneGCN [9] achieves fantastic performance, leveraging graph convolutional neural networks. Meanwhile, TPCN [10] splits temporal and spatial dimensions through a temporal point cloud network, modelling actors as an unordered point set.

Preceding studies employ the same fundamentals with slight alterations or append some new mechanism to previously proposed architectures. HOME [11] combines a CNN module, an RNN module and an Attention module to produce possible future trajectories as a heatmap of probabilities. The proceeding solution, GOHOME [12] accomplishes the same objective with a graph-oriented approach. DenseTNT [13], adopts VectorNet to encode the scene context of the HD map and constructs a dense set of goal coordinates as a base for prediction.

To our knowledge, the current, best-performing, solution is known as GANet [6]. Here, we regard best as the best-performing model on the Argoverse 2 [14] leaderboard. GANet adopts LaneGCN for encoding the motion of surrounding agents and the road scene, and predicts a goal area of where the agents' future trajectories will reside. These are then used to predict a set of sequences of birds-eye view coordinates.

Some recent ventures have also tried to draw inspiration from the successful emergence of NLPs, implementing transformer-based solutions. Both Forcast-MAE [15] and Traj-MAE [16], propose a masked encoder model in order to predict trajectories from HD maps in a self-supervised way and contribute a framework for pre-training of the network. Most recently, an end-to-end self-supervised model called SEPT [17] has been proposed, which utilizes cross-attention in order to decode the trajectory from the scene context.

Offline HD maps are detailed in nature and are considered expensive and difficult to maintain. Parallel to the aforementioned studies some research has also been dedicated to finding ways to decrease these requirements by generating the maps in an online setting with machine learning-based methods. VectorMapNet [18] is an example of this, extracting BEV features from onboard sensors and then generating a vectorized map from these. A newer network model called SMERF [19] feeds

a standard definition map into a transformer encoder which is then used as an additional input to the BEV-feature-extracting module, creating a prediction of the lane-topology. The authors in [20] and [21] also explores the usage of navigation maps (SD maps) as a replacement for HD maps.

1.2 Aim

This thesis aims to improve trajectory prediction within the context briefly mentioned in Section 1. Specifically, we want to investigate whether or not map data improves prediction when leveraged by an image-based backbone. Acknowledging that state-of-the-art research in most cases utilizes HD maps as a way to increase prediction performance, we propose to adopt those modelling approaches, hypothesising that they'll be applicable for SD maps too, as shown in [21]. We select two such methods, referred to as a graph-based method and a transformer-based method, which we then modify in such a way that front-facing camera images can be fused with the map data.

1.2.1 Objective

Based on the aim of this thesis, which is to compare prediction performance with or without map data, we first need to verify that our chosen methods of leveraging maps are viable options. We think that these methods are well described within the following steps: *extracting* features of the map, *fusing* found features with the features from the camera modality and *decoding* the path. How one is to execute these steps is thus an integral part of our investigation. We hypothesize that the *extraction* of features in the maps, i.e, with what neural network we attempt to learn from the maps and how the map modality is represented, will be of most relevance for the performance of our models, ergo why we consider two methods of doing so. However, the proceeding tasks may be of equal importance.

As we're leveraging map data to improve prediction performance, one may consider what information within the map that the network may attempt to make use of. Possibly, the network learns to query the map for feasible paths from the vehicle's current position, or it learns more subtle details in the map. By also feeding the marked route in the map to the network, we hope to be able to draw conclusions related to this question, based on whether the trajectory prediction improves further or if it inhibits generalization.

We summarize our objectives as the following research questions:

1. How may one leverage standard definition maps in trajectory prediction problems with front-facing camera images, i.e. *extract, fuse* and *decode* them, and what is the potential performance increase in comparison to not leveraging these maps?

2. Does the inclusion of route data further improve prediction performance?

1.2.2 Scope

Based on our proposed objectives we condition our scope with the following limitations:

- We completely disregard historical data, such as ego trajectories prior to the current timestamp of the respective scene. The most advanced, and also best-performing algorithms, tend to indeed make use of the historic motion of the ego vehicle [6]. We expect this to limit the performance of our solution when compared to this class of algorithms, however, we simply deem it too large of a scope for this thesis.
- With the proposition of combining map data with camera data, we argue that the *dense* information in our input changes from being contained within the HD map to being contained within our images. Our purpose is not to compare the camera, as a modality, to HD maps. In practice, HD maps are often the amalgamation of multiple data types, camera data included, that have been processed and simplified [18]. Because of this, raw sensor data should always contain more information than the produced map downstream, disregarding the fact that fused modalities may contain more information than the original single modality. For this reason, we are not interested in comparing our results with HD-map-based predictions.

Furthermore, we also assume that:

- When fetching the local navigation map, we make a Mercator projection around the GPS coordinate of the ego vehicle at the current frame. In doing so, we assume that the resulting localized map is true to the extent that resulting deformations around the north and south edges of the map are negligible.

1.3 Aspects of ethics and sustainability

Because of the inherent "black-boxness" of neural networks, one encounters a plethora of ethical ambiguities when studying them. Specifically, being able to guarantee that the algorithm does not exert behaviours of ill intent is not simply possible. For trajectory prediction, this leaves the vulnerability of the algorithm miscalculating a safe path, due to a poor prediction, or intentionally crashing due to a completely false prediction.

Furthermore, when working with labelled data, many organisations, especially companies, outsource the data annotation task to other companies. Some of these actors are less serious than others and are known to have poor working conditions and pay.

There is also the aspect of energy consumption, in connection to training the network itself, to be considered. The amount of computing power needed to train modern models is immense and likewise, the energy these processors consume to do so is not to be overlooked.

Due to these combined factors, one must consider the necessity of employing a neural network approach for solving the problem. In our case, the need is somewhat clear-cut. There are far too many tunable parameters within trajectory predictions for classical methods to be sensible and generalize well. Efforts are therefore made to avoid unnecessary computational load, whilst training, and to verify the origin of third-party data sets, if leveraged.

1.4 Contributions

We propose a new method for conducting trajectory prediction using front-facing camera images and standardised maps. By considering a set of different modelling approaches and what different combinations of input modalities to be used, as well as investigating under what circumstances these models operate properly and poorly, we finalize a trimodal model, which takes images, maps, and routes as inputs, called the IMR-TrajNet.

2

Theory

In this chapter, the reader is given a broad explanation of the underlying topics needed to understand the contributions presented in this thesis. First, we cover the basics of trajectory prediction. Furthermore, we consider how one needs to approach the formatting of the map data in order to properly fit this modality into this type of prediction problem. Moreover, we describe the fundamental concepts that make up the feature extraction models that will leverage this medium, starting with graph neural networks and finally touching upon transformers. More avid readers are encouraged to inspect the bibliography for further information on the topic.

2.1 Trajectory prediction

The task of trajectory prediction is one of deducing a set of sequential points, originating from a moving object of interest, in some real-valued space, that accurately describes the future positions of said object. In recent years, this problem is often posed as a learning-based task, that makes use of neural networks. In a sense, one defines the problem as a function approximation problem, where an input X that describes the state and/or attributes of the object of interest is mapped to the object's target trajectory Y through some function g, i.e. one seeks $g : X \to Y$. This is commonly accomplished by what is known as supervised learning.

2.1.1 Supervised learning

In supervised learning, we use known information of the function g in the form of independent measurements called labels, to find a suitable approximation. This is achieved by an iterative process where one defines a scoring function $f : L \times L \to \mathbb{R}$, where L is a labelled set consisting of pairings of elements in the input set X and the output set Y, to evaluate the current accuracy of the approximation in relation to these labels. The goal of this procedure is then to minimize this score, using some optimization algorithm, like gradient descent, to find

$$\min_{\hat{g}} \frac{1}{N} \sum_{i}^{N} L(y_i, \hat{g}(x_i)), \tag{2.1}$$

where $\hat{g}$ represents the approximated function we are trying to learn and N is the size of the labelled set. Typically, in trajectory prediction tasks, one assigns this

scoring function, or loss function, such that one minimizes the mean squared error (MSE) between the labels and their corresponding prediction

$$\text{MSE} = \frac{1}{N}\sum_{i}^{N}(y_i - \hat{g}(x_i))^2 \tag{2.2}$$

2.2 Neural networks

Using supervised learning, one can train a neural network (NN), i.e. a network of connected perceptrons [22], on the labelled set L to approximate the function $g : X \to Y$. In this section, we touch upon the three architectural methods of structuring such neural networks for trajectory prediction, that are relevant to our work. Note that within our context, neural networks exclusively refer to deep neural networks [23], which is a class of NNs that chains several layers of perceptrons together. The three network architectures are:

- **Convolutional Neural Networks with residual connections**: This class of networks are often used within the context of processing images for general classification tasks and utilizes convolutional operations to calculate pooled features of said images using a learned kernel [24]. Residual connections, i.e. connections that skip one or several proceeding layers, are then added in addition to these convolutional layers.
- **Graph Neural Networks**: This class of networks maintains a representation of a graphed data structure as a list of nodes and their connecting edges. The number of nodes and edges conveyed within each entry of this list are defined by set regions of the graph and an arbitrary initial state is assigned to each entry. Then a learned parametric function updates the state of all entries until these have converged. These states and their corresponding local nodes and edges are then passed through a second learned parametric function to produce an output that can be matched to the labelled data in question.
- **Transformers**: This class of networks uses a mechanism called attention that maintains a set of queries, keys and values, represented by fully connected layers. These are used to compute attention values based on the input data, which are weighted by a softmax operation. By stacking layers of these operations together, one is able to extract high-order relationships between the particular elements of the input and from these classify the input in its entirety.

Regarding the inputs X that are fed into the network, one must of course make sure that these are interpretable by said network, which requires the individual parts of the input to, in turn, be interpretable by the perceptron, something that is not necessarily true for the map data.

2.2.1 Maps for neural networks

Maps are in their essence a structured data type where data points are connected by one another. Typically, the maps are stored as multiple ordered sets of points

that are referred to as *links*. Each link defines a set of points that are connected by at most two other points within the set. This ensures that each link has a distinct starting point and an end point, and that the connecting points between them contain no branches, Figure 2.1 depicts this property. There can be any number of links within a map, and each link can contain any number of points. As neural networks generally take an input format resembling an array/matrix/vector, one needs to find ways of representing the connections between points of said data whilst abiding by this requirement. Within this thesis, we will refer to this process as the *encoding* of maps.

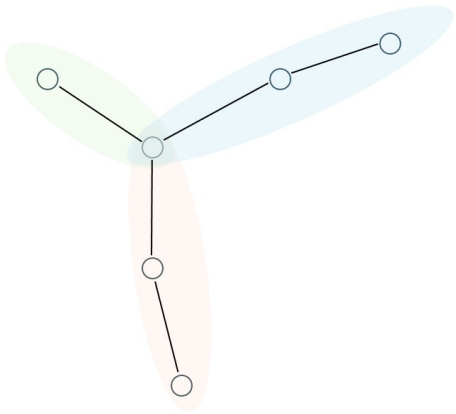


Figure 2.1: Arbitrary map depicting how points are divided between links (each link is represented by a different color). By enforcing links to split at branches in the map, chaining the points that makes up a link encodes their connections implicitly.

2.2.1.1 Encoding maps

For graph neural networks, the procedure of encoding a map into an array-like format is somewhat simple. Since the model architecture represents topological data well, one may resort to a simple representation of the data as just a list of the edges of the connected points within the map, i.e.

$$\{x_s, y_s, x_e, y_e\}, \tag{2.3}$$

where s and e are the start and end points respectively of a map edge in a 2D-cartesian coordinate system.

Meanwhile, for non-graph architectures, such as models mainly using self-attention[1], the literature is not necessarily in agreement of how to best create a suitable representation. For instance, polyline-representations, where connected points without branches (essentially links) are stacked into arrays, are used in [19], whilst some authors, as in [25], resort to using lists of edges, that is, the same representation as described in (2.3).

[1]Some graph-based architectures do indeed make use of self-attention, in this case, we are mostly referring to transformers.

2.2.1.2 Encoding attributes of maps

A benefit of maintaining maps in a structured data format is that one easily assigns attributes to a given data point to reflect real-world properties of the represented road. Common attributes are for example the number of lanes that said road contains, the speed limit and the road class, i.e. information regarding if the road is a highway or a street road. Often, this information is kept at a *link*-level, meaning that if one translates the map into an edge representation described above, link attributes must be copied and appended to the resulting edge

$$\{x_s, y_s, x_e, y_e, \text{attributes}\} \tag{2.4}$$

.

2.2.1.3 Localizing maps

Often, the trajectory prediction problem is stated within a local frame of reference. Especially for vehicle-based trajectory prediction research the common agreed upon point of reference is the world-view centred around the *ego*-vehicle. This world-view is often described by a cartesian coordinate system. On the other hand, maps are standardised to be represented using the *geodetic coordinate system* and as such a transformation between these measures is preferred.

Specifically, one transforms geodetic coordinates to a local cartesian system by the *Mercator projection* around a specific centre coordinate, i.e. the longitudinal and latitudinal position of the ego vehicle. The transformation is

$$x = R(\lambda - \lambda_0), \quad y = R \ln\Big[\tan(\frac{\pi}{4} + \frac{\varphi - \varphi_0}{2})\Big], \tag{2.5}$$

where R is the radius of the earth, φ is the latitude, λ is the longitude and λ_0, φ_0 is the longitude and latitude of the centre position of the projection chosen as the position of the ego vehicle.

2.3 Graph neural networks

Graph Neural Networks (GNNs) are a class of neural networks designed specifically for graph data. They are employed in scenarios where data is naturally represented as graphs, such as social networks, molecular structures, and communication networks.

2.3.1 Fundamentals of graph neural networks

A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set of nodes $\mathcal{V}$ and edges $\mathcal{E}$, where each node $v \in \mathcal{V}$ has its own features $\mathbf{x}_v$ and each edge $(u, v) \in \mathcal{E}$, connecting nodes u and v, may also have features $\mathbf{x}_{uv}$. The primary goal of a GNN is to learn a representation vector $\mathbf{h}_v$ for each node v, which captures not only the features of the node but also the features of its neighbourhood [26].

The core mechanism of GNNs is the message-passing scheme, where nodes iteratively exchange information with their immediate neighbours. The process can be described by two functions:

- **Message Function:** $M_t^{(l)}(\mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)}, \mathbf{x}_{uv})$, which computes the message sent from node u to node v at layer l based on their features from the previous layer $l-1$.
- **Update Function:** $U_t^{(l)}(\mathbf{h}_v^{(l-1)}, \sum_{u \in \mathcal{N}(v)} m_{uv}^{(l)})$, which updates the feature vector of node v by aggregating messages $m_{uv}^{(l)}$ from its neighbours $\mathcal{N}(v)$.

After L layers of message passing, the output $\mathbf{h}_v^{(L)}$ can be used for downstream tasks like node classification, link prediction, or graph classification.

2.3.2 Graph convolutional networks (GCN)

Graph Convolutional Networks (GCNs) operationalize the concept of convolution, traditionally applied to regular grid data, to the domain of graph-structured data. The key idea is to define convolution in terms of the graphs structure, specifically leveraging the adjacency matrix and node features.

The convolutional operation $\mathbf{h}_v^{(l)}$ in a GCN is designed to aggregate features from a node's local neighbourhood

$$\mathbf{h}_v^{(l)} = \sigma\left(\mathbf{W}^{(l)} \sum_{u \in \mathcal{N}(v) \cup \{v\}} \frac{1}{\sqrt{|\mathcal{N}(u)||\mathcal{N}(v)|}} \mathbf{h}_u^{(l-1)}\right), \tag{2.6}$$

where $\mathbf{W}^{(l)}$ is the weight matrix for layer l, $\mathcal{N}(v)$ denotes the neighbourhood of node v, including v itself, and σ is a nonlinear activation function like the ReLU. This formulation uses the symmetric normalization of the adjacency matrix to ensure smooth propagation of features across the graph.

2.3.3 Graph attention networks (GAT)

Graph Attention Networks extend GCNs by integrating attention mechanisms, which allow for adaptive importance weighting of a node's neighbours.

The attention mechanism α_{uv} in GATs computes the relative weights with which the features of neighbouring nodes are aggregated:

$$\alpha_{uv} = \text{softmax}_v\left(\text{ReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_u \| \mathbf{W}\mathbf{h}_v]\right)\right), \tag{2.7}$$

where $\mathbf{a}$ represents a learnable weight vector, and softmax_v is applied over all neighbours u of node v to make coefficients easily comparable across different nodes.

$$\mathbf{h}_v^{(l)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \alpha_{uv} \mathbf{W}\mathbf{h}_u\right) \tag{2.8}$$

This dynamic weighting scheme allows GATs to focus more on important neighbours, enhancing model flexibility and capacity to capture complex patterns in data.

2.4 Hierarchical graph neural networks for SD map processing

Hierarchical Graph Neural Networks (HGNNs) are designed to handle complex spatial data through a structured multi-layer approach. Specifically tailored for SD map data, these networks utilize two principal layers: a subgraph layer for localized data processing and a global graph layer for integrating these localized insights into a comprehensive map understanding.

2.4.1 Subgraph layer

The subgraph layer is foundational in hierarchical graph processing, dealing with highly granular data:

- **Vectorization:** Elements of the SD map are vectorized with attributes including position, direction, and type, as described in Section 2.2.1.2.
- **Local Graph Construction:** Each vector is treated as a node within local graphs. Nodes are interconnected based on spatial proximity or functional relationships, capturing the physical and logical structure of map features.
- **Polyline Subgraph Construction:** To exploit spatial and semantic locality, subgraphs are constructed for each polyline, with nodes connected within their specific polyline. This hierarchical structuring enhances the network's ability to focus on localized features within the polyline context.
- **Feature Extraction Process:**

$$v_i^{(l+1)} = \phi_{\text{rel}}\left(\text{genc}(v_i^{(l)}), \phi_{\text{agg}}\left(\sum_j \text{genc}(v_j^{(l)})\right)\right), \tag{2.9}$$

  where $v_i^{(l)}$ represents node features at layer l, $\text{genc}(\cdot)$ is a node feature transformation function realized by a multi-layer perceptron (MLP), $\phi_{\text{agg}}(\cdot)$ is an aggregation function typically using max pooling, and $\phi_{\text{rel}}(\cdot)$ applies a relational operation such as concatenation between node features.

  This MLP comprises a fully connected layer followed by layer normalization and ReLU non-linearity, emphasizing the efficient processing of node characteristics within each subgraph.

- **Polynomial Feature Aggregation:**

$$p = \phi_{\text{agg}}\left(\{v_i^{(L_p)}\}\right). \tag{2.10}$$

  Here, $\phi_{agg}(\cdot)$ aggregates the processed features from all nodes in the polyline, effectively capturing the comprehensive polyline-level features after L_p layers of processing.

This approach ensures that each subgraph captures the essential characteristics and interactions of its elements with high precision, crucial for tasks requiring detailed feature recognition and interpretation.

2.4.2 Global graph layer for high-order interactions

Building on the detailed local features extracted from the subgraphs, the global graph layer models high-order interactions across the entire SD map:

- **Aggregation of Local Features:** Features from subgraphs are aggregated into a global interaction graph, where each node feature set $\{p_1, p_2, ..., p_P\}$ represents integrated polyline features.
- **Modeling with Global Interaction Graph:**

$$p_i^{(l+1)} = \text{GNN}\left(p_i^{(l)}, A\right). \tag{2.11}$$

  Here, $p_i^{(l)}$ is the set of polyline node features, A is the adjacency matrix representing connectivity based on spatial distances, assumed to be fully connected for simplicity. The GNN uses a self-attention mechanism to process these interactions:

$$\text{GNN}(P) = \text{softmax}(PQ^T K)PV, \tag{2.12}$$

  where P is the node feature matrix, and PQ, PK, and PV are its linear projections.

- **Feature Decoding for Decision-Making:**

$$v_{\text{future},i} = \phi_{\text{traj}}\left(p_i^{(L_t)}\right) \tag{2.13}$$

  Decoding the future trajectories from the nodes, using MLPs or other advanced decoding mechanisms like anchor-based approaches for varied trajectory generation.

2.5 Transformers

As elaborated upon in Chapter 1, another popular approach to learning a map-to-trajectory model is to utilize a transformer of the type described in [27]. In general, the idea is to encode and decode map information in a non-sequential way. As maps are topological data structures that need to be decomposed in a semi-unordered way, see Section 2.3, the transformer's *perfect memory* may be of good use when synthesizing information from the medium as it can possibly distinguish important connections within the map regardless of this unordered property. Thus, we hope to learn the most important links within the map.

2.5.1 Attention

An important mechanism within these models that makes relationships between map points possible to identify is the *attention* mechanism. By representing the input data as a sequence of *queries, keys* and *values*, one is able to produce an output of whom's elements describe the relative importance of each of these values in the context of said queries and keys. Specifically,

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2.14}$$

where Q, K, V are matrix forms of the queries, keys and values and d_k is the dimension of the keys.

2.5.2 Cross attention

The length of the queries and keys of the attention operation do not necessarily have to align with each other. Only the *embedding dimension*, i.e. the higher order space onto which we map our input data, has to agree. As such, it is possible to fuse two sequences of different modalities by representing the queries and keys with the first and the value vector with the second. In doing so, one essentially asks the network to relate information of the second sequence based on the first. Assuming that we have two arbitrary sequences s_1, s_2,

$$\text{Cross-Attention}(Q_{[s_1]}, K_{[s_2]}, V_{[s_2]}) = \text{softmax}\left(\frac{Q_{[s_1]}K_{[s_2]}^T}{\sqrt{d_k}}\right)V_{[s_2]}, \tag{2.15}$$

where indices denote the sequence from which the matrices are calculated.

2.5.3 Non-autoregressive transformers

Commonly, one constructs layers of these attention operations on the input data called *encoder layers* that parses the information in a deep neural networks approach, after which the information is then decoded in a similar manner. In order to properly decode the information, the transformer architecture lets the model cross reference the assignment of importance of different pieces of the input information with the target information, often referred to as *teacher enforced learning* [28]. This way, the network basically cheats during training to learn relationships between the input and the target label. Then, during inference, one simply feeds the known bits of information about the target, in trajectory prediction that could for instance be the current position of the object, and lets the network predict the proceeding bits of information. In order to yield the entire target label, especially for trajectory prediction, the network recursively carries out inference on its own deductions until the desired information density is reached, in our case the entire future trajectory has been produced. This is often referred to as autoregressive inference.

There are two clear drawbacks of utilizing autoregressive inference. First of all, the inference time is linear in its complexity leading to a longer time of computation in

comparison to, perhaps more naive, one-step methods. Secondly, and perhaps more importantly, assuming that some slight prediction errors occur during inference, the model will have to deal with accumulating errors due to its own inaccuracies, something that it has not necessarily been able to learn. Due to these aspects, some novel transformer-based approaches replace teacher-enforced learning with a non-autoregressive approach [28], [29]. Typically, one leaves the architecture unchanged whilst replacing the information about the target label that the model receives in its encoder stage with learned parameters. These are in a sense arbitrary objects or entities that, due to the parameter assigned to them, will encourage the network to emphasise different parts of the input data during inference. Then, by utilizing some secondary operation, one can assign a confidence value to each entity's contribution to the estimation and thereby yield a prediction. This way, model uncertainties are no longer compounded and inference is carried out in a one-step manner. We will refer to this class of models as non-autoregressive transformers.

3

Methods

In this chapter, the reader is introduced to the general methodology for conducting trajectory prediction within the context described in Chapter 1. In a sense, this method can be described as a series of *permutation studies*. That is, by sequentially augmenting or altering components of our proposed models we aim to deduce the proper way of leveraging the medium, *maps*, through the discovery of how different sub-components in the model relate to prediction performance. These studies, or rather experiments, are carried out in the order as prescribed before. This ordering is maintained within this chapter as well, meaning that we will first discuss various methods of extracting features from the maps, after which methods for fusing and decoding this information are covered. Finally, we provide a quick discussion on the type and quality of the data used in the later experiments along with the metrics that are used for evaluation.

3.1 Feature extractor studies

Given the inherent structure of the maps, we hypothesise that an optimal way of retaining this information of structure exists and that it is beneficial to indeed make use of this information when inferring trajectories as described previously. As mentioned in Chapter 1, both GNNs and Transformers have been used previously to carry out predictions from maps. We assume that both approaches are applicable for our context as well. Therefore we propose an architecture as described in Figure 3.1, with two variants. Namely, one that relies on a GNN, as described in Section 2.4, and one that relies on a transformer architecture, as described in Section 2.5.3. This architecture also establishes the baseline design which performance comparisons are made. This baseline is simply defined as the feature extraction block that inputs images and a decoder block identical to the aforementioned models.

We train our models using the concept of supervised learning and use the $L2$ loss (MSE loss) as our scoring function. The general training approach also follows that of the double decent dogma [30], which essentially allows us to assume that our model is large enough for the *bias-variance trade-off* to not be a concern.

In order to limit variability within experiments the method of encoding the maps is chosen to be as similar as possible between architectures. In particular, the encoding scheme described in (2.3), i.e. that of edge representation, is chosen. The considered attributes, that are appended to each edge, are the speed limit of the road; the road

class, e.g. *highways*, *streets*, etc.; the number of lanes of said road; the direction of traversal of the road, i.e. backwards or forwards; and an indicator for whether the road is part of a route or not.

After features have been extracted we fuse the resulting feature vectors through some chosen operation. In the default case described in Figure 3.1, this operation corresponds to a simple concatenation of both vectors. Then finally, logits are transformed to the desired output space by a *multi-layered perceptron* (MLP).

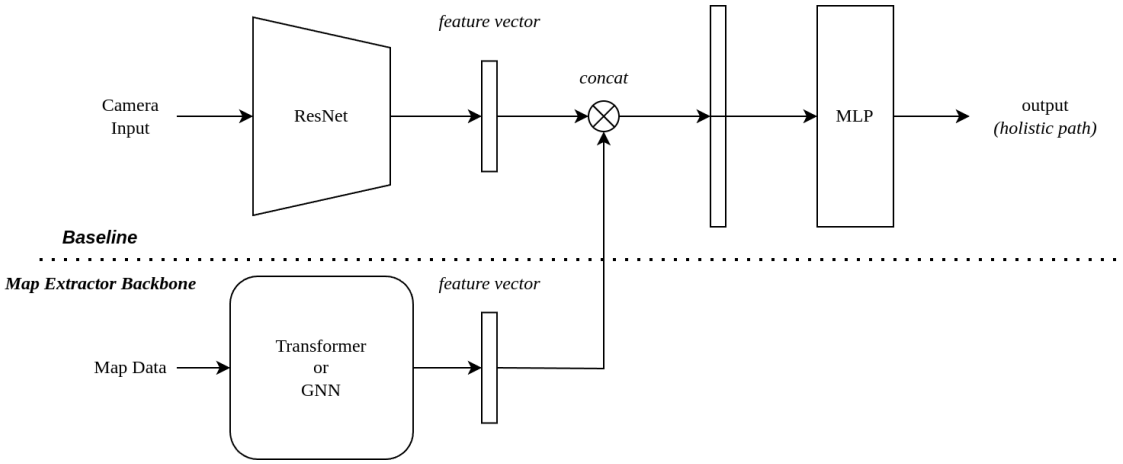


Figure 3.1: Schematic describing the architecture of the initial models proposed in the thesis. The figure shows how the baseline model will be extended in order to be capable of leveraging map data.

Using this architecture we train a set of networks where different combinations of input sources are provided. For the baseline model, only the image modality is usable whilst our proposed map extractors could input both maps and maps containing routes. Note that individual extractors may not test all combinations of input modalities.

3.1.1 Feature extractor implementations

In the baseline model, i.e. the model that only leverages images for predictions, data is passed through a CNN with residual connections. The architecture in particular is based on the ResNet architecture presented in [5].

The default models, as described previously, append their own encoders on top of this baseline, one GNN-based and one transformer-based. Their particular implementations are defined as follows:

Heirachical graph neural extractor - We implement a network following the implementation of the VectorNet model described in [8]. Map elements are encoded using edge representation and formed, based on their related links, into individual graphs called *polylines*. A global graph is then supplied these sub-

graphs and is then queried for the requested feature vector, as described in Section 2.4.

Transformer extractor - We implement a transformer with encoder and decoder layers as described in [27]. Here, the model is made to leverage teacher-enforced training, meaning that the bits of the target label are shown during training, encouraging the network to predict the removed information. During evaluation, one therefore queries the model recursively as described in Section 2.5.3.

3.1.2 Encoding routes

With the purpose of also investigating the possible benefits of supplying route information, i.e. the planned points within the map that the vehicle intends to visit, to the model, we chose to encode this information as attributes within the map itself. This implementation follows the description provided in Section 2.2.1.2. Of course, there are several other possible ways of supplying this information, route information is collected separately from the generation of the map and could thereby be supplied as its own modality, but our method requires no alterations of the network architecture to work allowing for quicker implementation.

3.2 Further model augmentations

Following the studies of viable feature extraction methods when utilizing maps, we question the previously established method for fusing the features from each modality. Concatenating each feature vector may for instance be too much of a naive approach if either modality is non-trustworthy for certain data points. In such a case the model may benefit from being forced to trust only one modality but allowed to attempt improving the features from this first modality by consolidating the second.

We also question the suitability of the established training policy by adding an auxiliary task to said policy. This policy hopes to stress the importance of correctly predicting the shape of the labelled trajectory.

Furthermore, we extend our model(s) to make use of multimodal predictions instead of singular predictions, in order to potentially improve the decoded path. Supported by [31], we argue that ego-vehicle-based driving is stochastic in the sense that two identical sets of inputs (the same image and map) may have multiple mappings to the codomain of the function we intend to approximate. In one-to-one supervised learning, i.e. one input mapping to one label, this property of, at best, partial invertibility, would force the model to try to average the output labels. A simple example could for instance be when the vehicle is driving next to an off-ramp. In this scenario, taking the off-ramp or continuing driving along the current road are both feasible short-term driving outcomes, but the model risks proposing an action of something in between. In cases concerning off-ramps, this tends to equate to off-road driving.

In order to combat this, one can carefully filter the training data such that only labels matching a predefined policy remain. This would lead to the model only, and always, considering one of the possible trajectories when making its prediction. This is of course not ideal, as in real-world driving, one may want to be able to propose either of the possible outcomes mentioned before. A much better approach is therefore to disregard the objective of predicting a single, distinct trajectory and instead train the network to predict all feasible trajectories, which is referred to as multimodal predictions.

3.2.1 Cross attending modalities

Based on the default models' performance in comparison to the baseline model, we propose to conduct ablations on the topic of fusion of the modalities, by replacing the previously mentioned concatenation operation with an, arguably, more sophisticated cross-attention operation, see Section 2.5.2. This way, one hopes that the model may make prioritization of which features to amplify and which to dampen in one of the produced feature vectors based on the other.

In what order one should relate the modalities to one another is not necessarily clear. However, within the context of this thesis, we view the map modality as an additive extension of previous research, i.e. image-based predictions are proposed to be improved by leveraging maps. Therefore, only one configuration of this model is to be considered. That of when the map modality features make up the query vector and the camera modality make up the key and value vector.

3.2.2 Auxiliary tasks

In our ego trajectory prediction method, we employ an auxiliary loss to ensure that the predicted trajectory aligns closely with the map. This auxiliary loss complements the primary L2 loss, by giving a measure of how well the predicted trajectory aligns with the closest route element in the map in terms of shape. For each predicted point $p_i = (x_i, y_i)$, we find the closest point in the route by projecting p_i onto the route. This gives us a set of projected points p'_i.

Next, we compute the headings between consecutive points in the predicted trajectory and the consecutive points in the projection. The headings are calculated through

$$h_i = \text{atan2}(y_{i+1} - y_i, x_{i+1} - x_i), \tag{3.1}$$

$$h'_i = \text{atan2}(y'_{i+1} - y'_i, x'_{i+1} - x'_i), \tag{3.2}$$

where h_i and h'_i are the headings between points $p_i \to p_{i+1}$ and $p'_i \to p'_{i+1}$, respectively and atan2 is the two-argument arctangent function. This arctangent function takes into account the signs of both arguments to determine the correct quadrant of the angle.

The auxiliary loss is then calculated as the sum of the absolute differences between

the headings of the predicted points and the projected points by

$$\mathcal{L}_{\text{aux}} = \sum_{i=1}^{N-1} |h_i - h'_i| \,. \tag{3.3}$$

The total loss function used for training is the weighted combination of the primary L2 loss and the auxiliary loss:

$$\mathcal{L}_{\text{total}} = \alpha \mathcal{L}_2 + \beta \mathcal{L}_{\text{aux}}, \tag{3.4}$$

where α and β are weighting factors that balance the contributions of the L2 loss and the auxiliary loss, respectively.

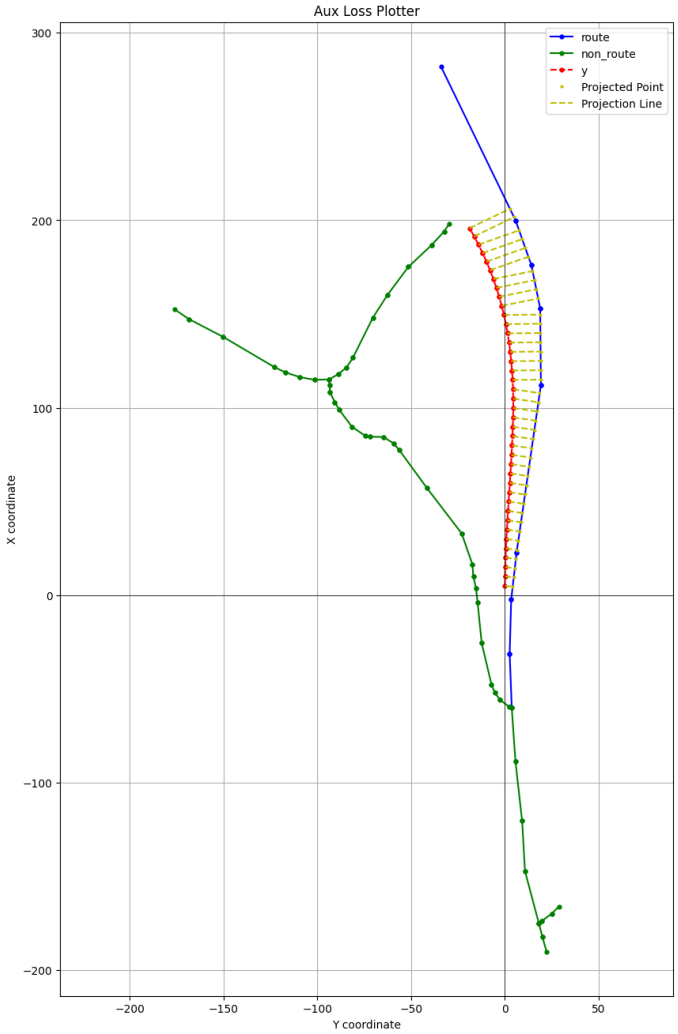


Figure 3.2: Visualization of the auxiliary loss computation. The predicted points p_i are projected onto the route to obtain p'_i. The headings between consecutive points are used to calculate the auxiliary loss.

This auxiliary loss intends to ensure that the predicted trajectory not only minimizes the distance to the ground truth trajectory but also aligns well with the road network, resulting in more realistic and feasible path predictions.

3.2.3 Multimodal predictions

A further improvement to consider is that of extending the models to accommodate for multimodal predictions. Discarding the idea of predicting a single deterministic trajectory from the input data, some studies resort to predicting a distribution of feasible paths that the vehicle may take after the point of where the prediction is to start. Typically one lets the network assign a confidence to these predictions in order to gauge their probability of occurring in an actual driving scenario. The way of reasoning around the trajectory prediction problem is therefore slightly different compared to the single prediction perspective described in Section 2.1. Instead, one argues that multimodal trajectory prediction (MTP) captures the apparent complexities within a driving scenario of different agents and obstacles. There are simply multiple possible actions (paths) for a vehicle to take at a certain point of time that are not necessarily wrong (or dangerous), and a human would consider all of these during driving before after picking one, not necessarily with a proper intent. It is therefore naive to expect a network to be able to infer these random or illogical decisions from the input data and could therefore improve, in terms of robustness[1], if allowed not to consider this aspect of the trajectory prediction problem.

Multimodal predictions can be implemented within a deep network by many different methods. In this study we chose to follow a query-based method initially described in [29] for object detection within images, and later extended to online HD-map construction through [32]. In order to map this method onto our particular problem description, we let a set of queries define arbitrary entities, simply a set of learned parameters, with the idea of having these represent one prediction each, thereby allowing for a set of trajectories to be predicted during inference. In particular, one feeds these as a value vector to an encoder-decoder transformer, described in Section 2.5.3.

In order to encourage diversity within the sampled trajectories of the output, we propose to introduce a slightly different loss than the normal regression loss provided in Section 2.1. Specifically, we use a similar MTP loss as described in [33], which uses a matching algorithm to find the prediction with the minimal L2 loss:

$$q^* = \underset{q \in 1,\dots,Q}{\arg\min}\ \mathrm{L2}(\tau, \hat{\tau}_q), \tag{3.5}$$

$$\mathcal{L}_{\text{reg}} = \mathrm{L2}(\tau, \hat{\tau}_{q^*}), \tag{3.6}$$

where τ denotes the target trajectory of the sample and $\hat{\tau}$ denotes the predicted trajectory from the specific query q in the set of considered queries of size Q. This loss is then combined with the binary cross entropy loss between the confidence scores of each prediction, supplied by a separate prediction head of the model, and the previously matched prediction

$$\mathcal{L}_{\text{bce}} = -\frac{1}{Q}\sum_{q=1}^{Q} \delta_{qq^*} \log \hat{c}_q + (1 - \delta_{qq^*}) \log(1 - \hat{c}_q), \tag{3.7}$$

[1]Robustness here referring to the minimisation of outliers, that is, reducing how wrong the network is when it is wrong.

where δ and $\hat{c}$ denote the labeled class and the predicted class, respectively. Class labels are inferred by applying δ as the Kronecker delta, obtaining the value one if $q = q^*$ and zero otherwise. The combined MTP loss then becomes

$$\mathcal{L}_{\text{mtp}} = \alpha\mathcal{L}_{\text{bce}} + \mathcal{L}_{\text{reg}}, \tag{3.8}$$

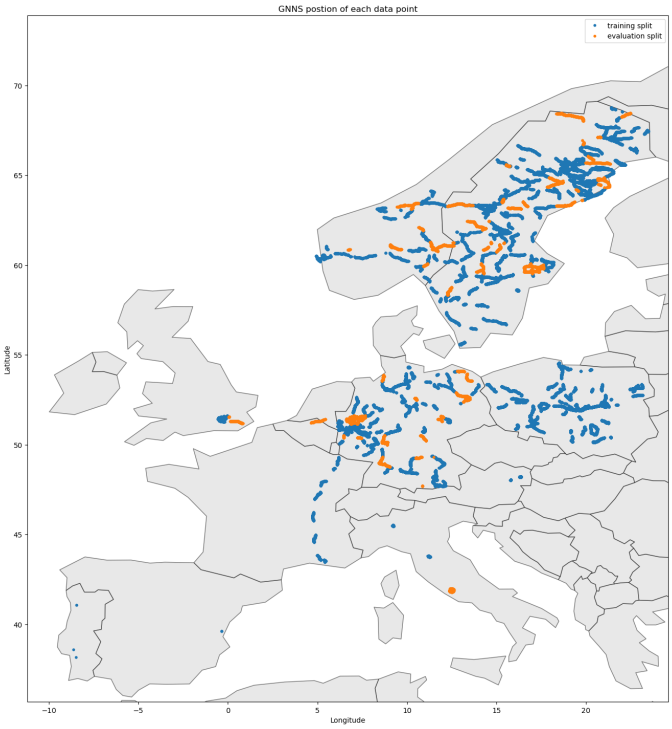
where α is a weight factor. With this loss, we can train a set of predictions of size Q that maps to one query each. During inference one thus receives a sampled distribution of trajectories, which hopefully captures multiple distinct feasible driving behaviours which could be used to deduce the actual future trajectory of the vehicle in a proceeding task.

3.3 Data

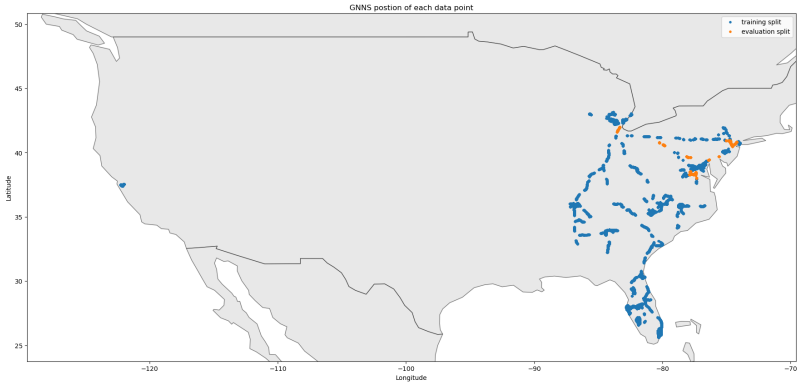
In order to train our models with supervision, we use a labelled dataset consisting of images, GPS positions, routes, and ground truth labels. The ground truth is synthetically generated by collecting consecutive GPS points from the time when the matching image is captured until a travelled distance of 200 meters is reached. The routes are similarly generated, with the additional step of matching the GPS positions to suitable map elements.

Prior to training, we generate local maps from the stored GPS positions where the images were captured. The map elements fetched in this manner are referenced from OSM [34]. These maps are softly constrained to a size of 400 by 400 meters. If a map element extends beyond this boundary, it is retained, but elements completely outside the range are discarded.

The dataset contains approximately 290k data points, which are split into a training set of 250k points and an evaluation set of 40k points. The splits are depicted geographically in Figure 3.3.



(a)



(b)

Figure 3.3: Geographical distribution of training and evaluation data. (a) shows splits in Europe and (b) splits in the US. Blue points represent data points in the training set, and orange points represent those in the evaluation set.

Below, by analyzing the trajectory of the datasets based on the cumulative trajectory yaw difference, we provide a visualization of the distribution of different distinct scenarios in the data. The cumulative yaw difference, which reflects the total change in the vehicle's heading over the entire trajectory, is calculated as follows:

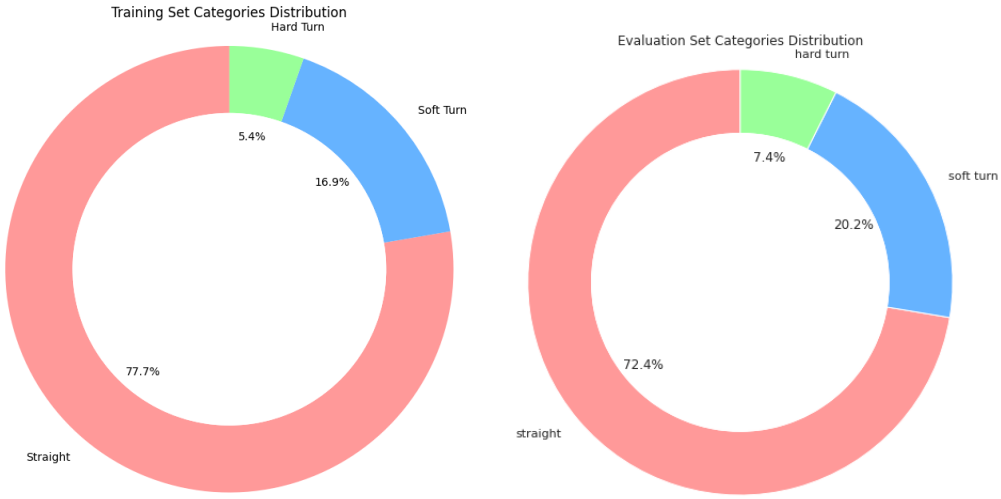
$$\Delta\varphi_{\text{cumulative}} = \sum_{i=1}^{N-1} |\varphi_{i+1} - \varphi_i| \,, \tag{3.9}$$

where φ_i and φ_{i+1} are the yaw angles at consecutive points i and $i+1$, and N is the total number of points in the trajectory.

Based on the cumulative yaw difference, we categorize the trajectories into three types:

- **Straight Data**: $\Delta\varphi_{\text{cumulative}} \in (0, 5)$
- **Soft Turn**: $\Delta\varphi_{\text{cumulative}} \in (5, 15)$
- **Hard Turn**: $\Delta\varphi_{\text{cumulative}} > 15$

The proportions of the training and evaluation datasets across these categories are shown in Figure 3.4. The distribution of these categories, within the training and evaluation data set respectively, are depicted in Figures 3.5 and 3.6

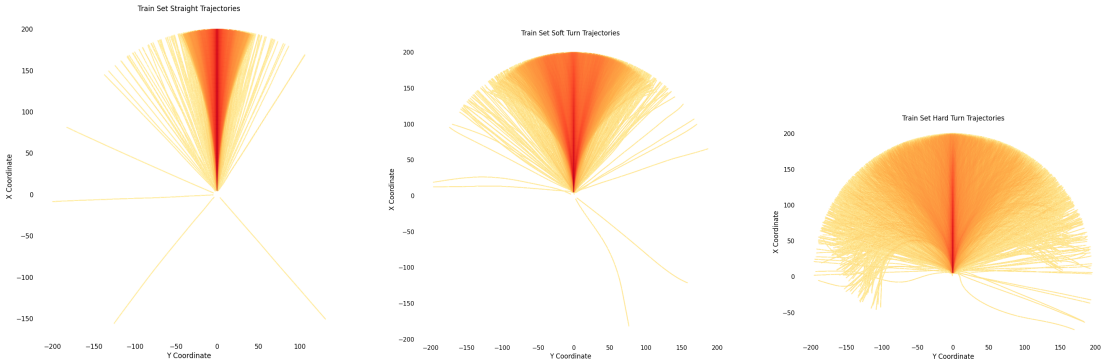


(a) (b)

Figure 3.4: Distribution of the training (a) and evaluation dataset (b) based on trajectory yaw difference, grouped by the above defined categories of scenarios.

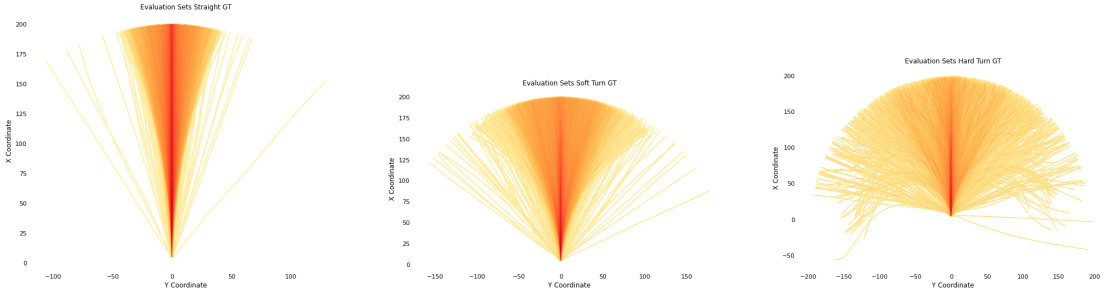
3.3.1 Inaccuracies in the maps

Since the maps are generated from GPS positions some inaccuracies in regard to centring and orientation of the maps in relation to the ground truth may occur. This may be desirable if one expects real-world driving to have the same inaccuracies within measured samples (GPS positions) as the gathered training data. On the



(a) (b) (c)

Figure 3.5: The distribution of ground truth trajectories in the training dataset over a set of different scenario types, (a) straight scenarios, (b) scenarios with soft turns, and (c) scenarios with hard turns.



(a) (b) (c)

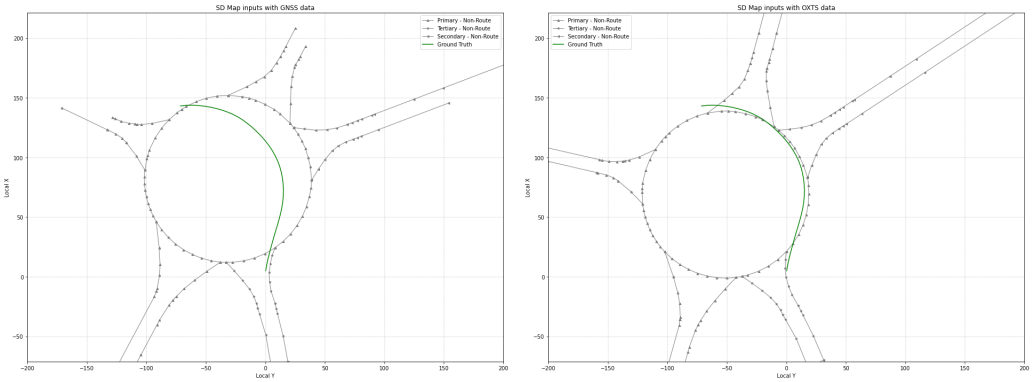
Figure 3.6: The distribution of ground truth trajectories in the evaluation dataset over a set of different scenario types, (a) straight scenarios, (b) scenarios with soft turns, and (c) scenarios with hard turns.

other hand, essentially training the network to filter these inaccuracies between the maps and the ground truth may yield a brittle model for out-of-distribution data, i.e. data that contain a different noise.

To avoid discrepancies, we create two versions of our dataset. One where maps are generated with GPS data from commercial grade instruments, i.e. devices one could find in a normal car, and one using a significantly more accurate device, which we will refer to as OXTS. For the purposes of this thesis, we will consider the later measurement to be near perfect. The difference in the map to ground truth missmatch between the two methods can be viewed in Figure 3.7.

3.3.2 Concerning the mix of routes and no routes

The mix of data points containing routes and data points that do not contain routes are roughly proportioned 1:1. Though a sober mix in the data could yield a better model in terms of robustness, we hypothesise that the current ratio of "non-routes" will greatly inhibit the models learnt trust towards this part of the data, which



(a) (b)

Figure 3.7: Depicts the difference in alignment between the map and the ground truth trajectory when different sources for measurement are used. (a) shows the miss-alignment when GPS (or rather GNSS) data is used, while (b) shows the improvement when OXTS is used.

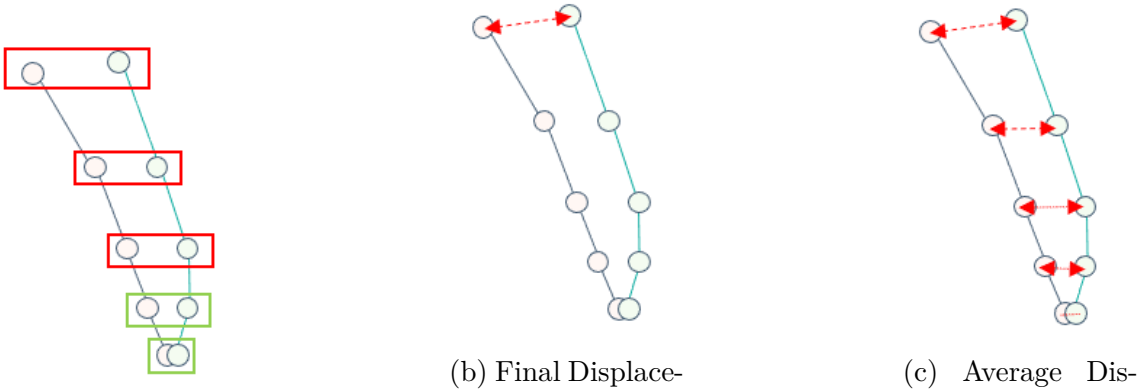
may affect general performance negatively. As mentioned previously, route data is generated from the ground truth. This means that all data points could contain route information. Evidently, this is not the case, which possibly has to do with the quality of the missing routes being subpar, and therefore being removed or skipped by the currently used algorithm. As such we apply a greedy algorithm, described in Appendix B, to fill these gaps, creating a third version of our dataset as a result.

3.4 Evaluation metrics

Evaluation of the ego-trajectory prediction performance will be based on specific metrics referenced from the Argoverse[14] motion forecasting dataset metrics, one of the prominent benchmarking datasets within the research field. Here are details on the key metrics (an illustration of the metrics is provided in Figure 3.8:

- **Miss Rate (MR)**: This metric measures the ratio of predictions where the predicted trajectory endpoint is not within 2.0 meters of the ground truth endpoint.
- **Final Displacement Error (FDE)**: This metric calculates the L2 distance between the endpoint of the predicted trajectory and the ground truth.
- **Average Displacement Error (ADE)**: This metric represents the average L2 distance between the predicted trajectory and the ground truth over the entire prediction horizon.

When evaluating multimodal networks we calculate these metrics based on the specific trajectory of the set of estimated trajectories that minimizes the $L2$ loss of the particular scenario that the measurement is made during. We refer to the metrics calculated with this method by their name with a preceding "min" attached, i.e. *minFDE*, *minADE, minMR*. Additionally, we calculate a β-minFDE score which



(a) Miss Rate

(b) Final Displacement Error

(c) Average Displacement Error

Figure 3.8: Illustrations of the evaluation metrics.

adds the squared inverse confidence to the FDE score in order to capture the confidence of the prediction and its quality, i.e:

$$\min\beta\text{-FDE} = (1 - \rho)^2 + \text{minFDE}, \tag{3.10}$$

where ρ is the assigned confidence to the corresponding trajectory.

4

Experimental results

In this chapter, we present the resulting changes in performance, based on the design alterations that are to be tested. Unless specified differently, all experiments are carried out over a training time of 40 epochs with a training set of 250k samples and an evaluation set of 40k samples.

4.1 Feature extractor experiments

First, we establish a measure of the baseline model along with the, to be compared, default models, by conducting a modality permutation study. Here, the permutations are labelled by the flags; *image*, *map*, and *route*, which correspond to the particular input modalities that are used during the training and evaluation of each respective model. The specific feature extractor, used in each test, follows the implementation described in Section 3.1. The results are depicted in Tables 4.1.

Table 4.1: Measured performance after evaluating the baseline model and our models (with their configurations). Lower is better. The best metrics are marked in bold.

Extractor	Configuration			$L2$ loss ↓	FDE ↓	ADE ↓	MR ↓
	Image	Map	Route				
ResNet	✓	-	-	68.8033	9.6733	3.6102	0.6471
Transformer	✓	✓	-	68.8463	9.8993	3.7767	0.6788
	✓	✓	✓	71.2780	10.383	3.9082	0.6954
GNN	-	✓	-	44.9999	7.5270	2.9849	0.6832
	✓	✓	-	**32.0757**	6.3039	2.3869	0.6036
	✓	✓	✓	34.5250	**6.1778**	**2.2358**	**0.5542**

Furthermore, we collect the performance of the different configurations of our best feature extractor (the GNN) over the set of scenarios specified in Section 3.3 in Table 4.2

Table 4.2: Measured performance of predefined scenarios of the best default model (with its configurations).

Scenario type	GNN conf.			$L2$ Loss	ADE	FDE
	Image	Map	Route			
straight	-	✓	-	8.8956	1.6168	4.1636
	✓	✓	-	3.0934	1.2542	3.2244
	✓	✓	✓	**1.9784**	**1.0005**	**2.6642**
soft turn	-	✓	-	45.4195	4.3144	10.3332
	✓	✓	-	29.7856	3.5154	8.7771
	✓	✓	✓	**24.6780**	**3.0237**	**8.5815**
hard turn	-	✓	-	394.9259	12.6766	32.6108
	✓	✓	-	**306.5216**	**10.7096**	**29.3066**
	✓	✓	✓	377.7383	12.1083	33.8153

4.1.1 Augmenting the data

As elaborated upon in Section 3.3, some inaccuracies and information deficits exist in the data, which could be argued to limit the network's capability of properly approximating the function in question. Due to this, we propose to test the two improvements in the quality of the data described in Section 3.3. First, we denoise the maps by re-aligning their elements closer to the ground truth, minimizing possible inaccuracies in the measured heading angle of each sample. Second, we increase the number of samples within the dataset that contain route information in order to support model configurations that leverage this modality. In both of the following experiments only the best-performing feature extractor, the GNN, is considered.

4.1.1.1 Denoising the maps

As mentioned above, the relative miss-alignment between the maps and the ground truth is remedied by the method described in Section 3.3.1. The resulting improvements in performance are depicted in Table 4.3.

An arising concern from applying this adjustment to the data is that a system trained on such semi-perfect data may turn brittle towards imperfect data. To investigate the validity of this concern we show a side-by-side comparison of our models of interest when validated on the original, non-aligned data. Results are shown in Table 4.4

Table 4.3: Measured performance of the specified model when the inputted SD maps are improved in terms of how well they align with the groundtruth. The improvement is indicated by the change of the sourced GPS position during map generation. OXTS is considered to be a more accurate source, as described in Section 3.3.1.

GNN conf.			map source	$L2$ loss	FDE	ADE	MR
Image	Map	Route					
-	✓	-	GNSS	44.9999	7.5270	2.9849	0.6832
			OXTS	**41.7100**	**6.6970**	**2.7217**	**0.6026**
✓	✓	-	GNSS	32.0757	7.3039	2.3869	0.6036
			OXTS	**31.7525**	**5.5400**	**2.0804**	**0.5187**
✓	✓	✓	GNSS	34.5250	6.1778	2.2358	0.5542
			OXTS	**28.4182**	**5.1013**	**1.9760**	**0.5070**

Table 4.4: The implications of augmenting the data as described previously if the resulting model is required to predict non augmented data. Only the best model, i.e. the model with all modalities available is presented.

GNN with image, map & route configuration					
Training data	Validation data	$L2$ loss	FDE	ADE	MR
GNSS	GNSS	34.5250	6.1778	2.2358	0.5542
OXTS	GNSS	**29.9412**	**5.5591**	**2.0923**	**0.5438**

4.1.1.2 Increasing route availability

As mentioned in Section 3.3.2, the route availability in the current model is approximately 50%. We suspect that the model which leverages routes may be hindered, in terms of performance, by this limited availability. Therefore, we train a GNN extractor model with all modalities using data with full route availability. This model, along with the original model, is then tested on datasets with both 50% and 100% route availability. The results are presented in the Table 4.5.

Table 4.5: Comparison of $L2$ loss, FDE, ADE and MR for the GNN model (with all modalities included), trained on 50% mixed routes and 100% routes, evaluated on both 50% mix and 100% route datasets.

GNN with image, map & route configuration					
Training set (% routes)	Validation set (% routes)	$L2$ loss	FDE	ADE	MR
50%	50%	34.5250	6.1778	2.2358	0.5541
	100%	35.2033	6.7686	2.4346	0.6388
100%	50%	32.3891	6.7686	2.4220	0.6422
	100%	**20.1737**	**3.9692**	**1.6440**	**0.4432**

4.1.2 Removing the map modality

Followed by the data augmentations (oracles) described above, we consider a fourth permutation of the set of supplied input modalities. Specifically, we remove the map modality from the fully configured GNN model, training a model with only images and routes.

In the following experiment, we choose to keep the augmentation of the data, described in Section 3.3 and tested in 4.1.1. This has to do with the necessity of routes in said experiment along with the need to have well-aligned maps-to-ground truths in order to generate these extra routes, see Appendix B for a more detailed explanation of why that is.

Removing the map modality in practice is accomplished by deleting all elements in the map that do not contain a route, thus creating a minimal representation of the map that only captures information that could be considered attained by the route. This filtering of the elements of the map is applied on both the training and evaluation data. Results are shown in Table 4.6.

Table 4.6: Measured performance in the case where one removes that map modality from the input data. All data points in both the training and validation sets contain routes and are calibrated by OXTS. Only the GNN extractor is considered.

GNN conf.			$L2$ loss	FDE	ADE	MR
Image	Map	Route				
✓	✓	✓	20.1737	**3.9692**	**1.6440**	**0.4432**
✓	-	✓	**18.0419**	4.2699	1.9096	0.4654

4.2 Fusion experiment

With the previous investigation of what extractor model would be of use for our problem description done, we test the suitableness of concatenating the feature vectors from this extractor head with the features from the image encoder, the ResNet, by leveraging the GNN as the extractor. We keep using the oracles investigated previously.

In order to investigate the potential need of the concatenation operation described in Figure 3.1, we contrast its performance with that of a cross-attention operation, if said operator replaces the previous. This new operation follows the description in Section 3.2.1. In particular, we implement an operator that takes the map modality as queries and utilizes these to compute attention values of the features from the image. These values are then multiplied with the image features, as described in Section 2.5.2, in order to possibly produce a set of features of improved quality.

Results are shown in Table 4.7. Trajectories are decoded in the same way in the extractor experiments, via an MLP.

Table 4.7: Measured performance of the GNN feature extractor model, with all modalities available, if one replaces the concatenation operation shown in 3.1 with a cross-attention operation that uses that map features as queries, and the image features as keys and values.

GNN with image, map & route configuration				
Fusion Module	$L2$ loss	FDE	ADE	MR
concatenation	**20.1737**	**3.9692**	**1.6440**	**0.4432**
cross-attention	37.2011	7.0122	2.4233	0.5888

4.3 Auxiliary task experiment

In this study, by altering the training procedure of the specified model by introducing an auxiliary loss we hope to encourage further trust, or reliance, towards the map modality during inference. This auxiliary loss is defined as a measure of the agreement between the prediction and the closest route element in terms of the difference in angle between them. The idea is thus to enforce predictions to follow the shape of the route more closely. The results of the corresponding experiment are depicted in Table 4.8. The full algorithm for calculating this auxiliary loss is provided in Section 3.2.2.

Table 4.8: Measured performance of a GNN extractor configured with all modalities if one applies an additional auxiliary loss during training.

GNN with image, map & route configuration				
Training Policy	$L2$ loss	FDE	ADE	MR
$L2$ loss	**20.1737**	**3.9692**	**1.6440**	**0.4432**
$L2$ and auxiliary loss	21.7465	4.1213	1.7024	0.4449

4.4 Multimodal experiment

Finally, we extend the decoder head of our GNN model to make multimodal predictions. We follow the prediction policy described in Section 3.2.3. Specifically, we implement multiple predictions by utilizing a non-autoregressive transformer with queries mapping to separate trajectories as features to be decoded.

Whilst being decoded into human-readable trajectories by a 2-layered MLP, we also task the network to give a confidence value that measures the feasibility of each prediction. We pass these confidences through a softmax operation in order to frame said values as probabilities. The goal is then to teach the network to assign a high probability to the trajectory that minimizes the $L2$ loss and a low probability to the remaining paths.

We suspect that the problem of assigning the correct confidence to the right path grows more difficult with more predicted trajectories. This suspicion mainly stems from the way confidences contribute to the loss. Many predictions may yield sets of

outputs that are essentially duplicates which the model reasonably assigns an equal confidence. This may however not be the sort of behavior that actually minimizes the loss. Meanwhile, more predictions clearly increase the likelihood of the model finding the labelled trajectory within its set of predictions. Ergo, more predictions may yet be preferable.

Because of this ambiguity in the choice of the number of predictions, we treat this as a hyperparameter and train three networks with different numbers of queries, namely $3, 6$ and 9. The performance of the resulting model is shown in Table 4.9. Note that we limit this part of the study by only considering the GNN as the relevant feature extractor for the map and route modalities.

Table 4.9: Resulting metrics if the GNN extractor model, with all input modalities being leveraged, is recast as a model that produces multimodal outputs.

GNN multimodal model with all components configured				
Number of queries	minFDE	minADE	minMR	β-minFDE
3	14.1510	5.4670	0.8811	14.4705
6	10.5242	4.1257	0.7913	10.9969
9	**8.3934**	**3.3258**	**0.7330**	**8.9586**

5

Discussion

In this chapter, we analyze the findings from the experiments and results presented in Chapter 4. We begin by discussing the performance of our chosen methods for extracting features of the SD map; the Transformer and the GNN approaches along with their respective configurations. Following this, we evaluate the tested idea that using more precise data can improve the prediction quality. This discussion is then followed by one related to the results found when examining the concatenation operation in relation to a cross-attention operation. After that, we discuss our findings related to the introduced auxiliary task that has been tested. We finalize the discussion by showing some qualitative results of some interesting scenarios, and by covering the found results of the model extension into multimodal predictions.

5.1 Evaluation of extractor related experiments

In this section, we analyze the performance of each extractor, one at a time, and draw certain connections between different configurations. In order to keep a neat distinction between different configurations of our different models, we use the acronyms I, M, and R to indicate which input modalities are used in the particular modal that is being discussed. Here, I:=image, M:=map, and R:=route.

5.1.1 Transformer extractor evaluation

First, we analyze the performance of the Transformer extractor against the baseline ResNet model, referencing the results from Table 4.1.

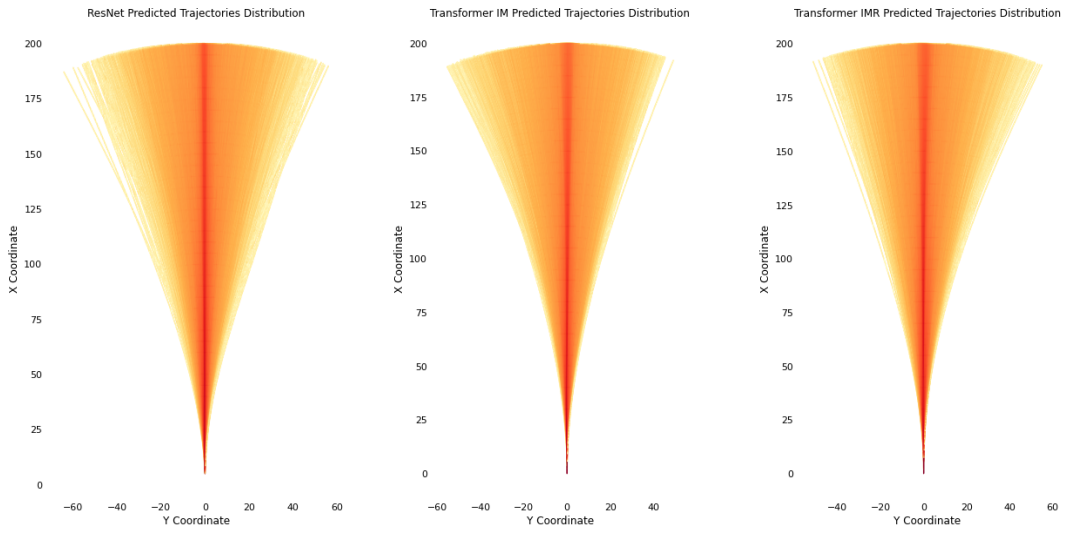
Both models exhibit similar performance in terms of evaluation metrics according to Table 4.1. As an example, both models show very similar L2 losses. This pattern of comparable values holds true across both configurations and all scenario types, e.g. straight, soft, and hard curves. Ergo we state that the models perform at a similar level, indicating that the Transformer extractor doesn't offer a substantial improvement over ResNet in this regard. In fact, the slight increase in various metrics implies that the Transformer extractor might produce features that are mostly interpreted as noise and, in turn, limit performance.

The distribution of predicted trajectories for the ResNet, Transformer IM, and Transformer IMR models, visualized in Figure 5.1, further reinforces this argument of the

Transformer, at best, not contributing to the prediction. Figures 5.1a, 5.1b, and 5.1c illustrate that all three models exhibit highly similar trajectory distributions.

Quite possibly, this relates to the teacher-enforcing technique used for training this network. During evaluation, the policy of recursively extending the prediction causes features extracted early in this recursion to have a compounding influence on the final output. As the features from the ResNet are always present and unchanging, these may therefore implicitly dominate the prediction.

Interestingly, both models struggle with accurately predicting curvy roads. This limitation can likely be attributed to the restricted field of view provided by the images. Such limitations highlight the difficulties of relying solely on image data for trajectory prediction, as reseeding horizons in images leads to insufficient contextual information at the boundary of said horizons to effectively handle such cases.



(a) (b) (c)

Figure 5.1: Distribution of predicted trajectories for baseline and Transformer models. (a) shows the distributions for the ResNet model while (b) and (c) show the distributions for the Transformer IM and IMR models respectively.

The visual confirmation from Figure 5.2 strengthens our argument. All three models (ResNet, Transformer IM, and Transformer IMR) demonstrate strikingly similar trajectories and struggle in scenarios with limited visibility, as exemplified by the sharp turn in this figure.

In summary, while the Transformer extractor offers additional information processing capabilities, the results from Tables 4.1 and the trajectory distributions in Figure 5.1 collectively suggest a minimal impact. The similar performance metrics across models indicate that the Transformer extractor is not significantly enhancing the model's predictive abilities and that ResNet features remain a critical component for overall performance.

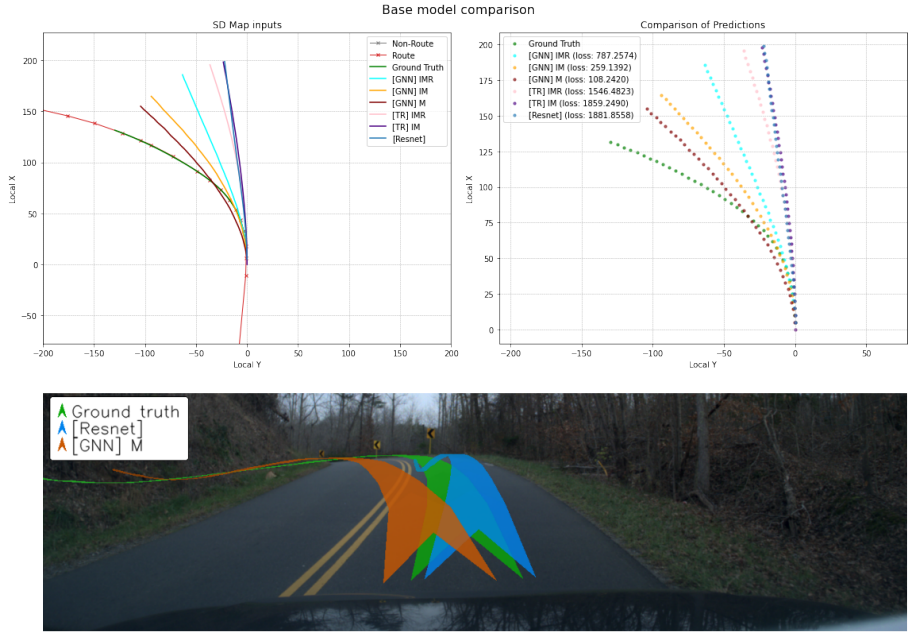


Figure 5.2: Predictions during inference of the baseline and the best default model in the case where the camera has a limited horizon.

5.1.2 GNN extractor evaluation

This section analyzes the performance of the GNN extractor compared to the baseline ResNet and Transformer models. We reference the results from Tables 4.1 and 4.2 for the discussion.

In general, the results collected in Table 4.1 highlight that the GNN models outperform the ResNet and Transformer models across all metrics. Qualitatively, as shown in Figure 5.3, we can clearly observe that using the GNN model significantly outperforms the other models. In this scenario, the camera's field of view is limited to a certain horizon, and as mentioned earlier, those models struggle to yield accurate predictions during sharp turns. However, the GNN model, including its variants M, IM, and IMR, can predict these cases perfectly.

The distribution of the predicted trajectories for the GNN models, shown in Figure 5.4, supports these findings. The figures illustrate that the GNN models are capable of more accurately predicting complex trajectories compared to the ResNet and Transformer models. We argue that this performance boost comes from two sources:

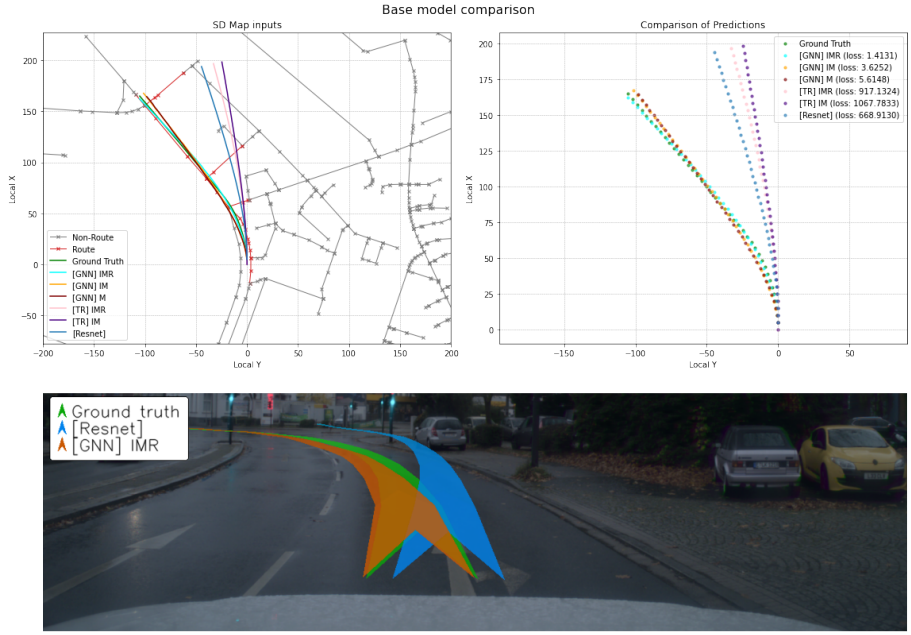


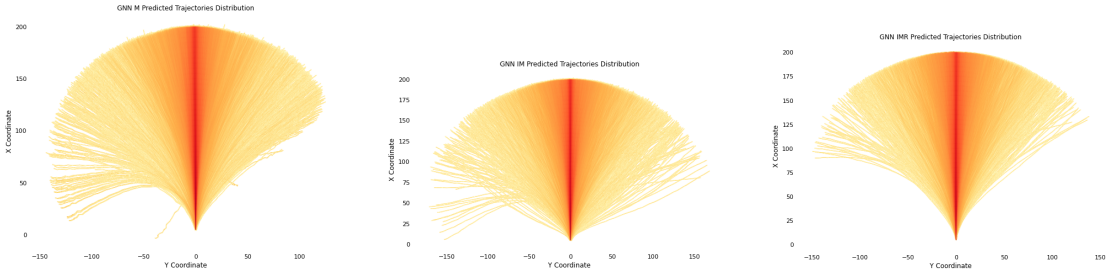
Figure 5.3: Predictions during inference of the baseline and the best default model in the case of taking a hard turn.

a) The GNN extractor is not at all affected by increased difficulty of prediction in certain regions of the output space, unlike the image extractor which struggles to make good predictions at the boundary of the horizon in the image.

b) The image provides proper information about what local coordinates are part of the drivable path of the road scene which the map medium, in itself, does not necessarily know.

Combining both modalities does therefore allow the model to have the best of both worlds, enabling the model to see beyond the horizon of the image and constraining the prediction to the drivable path, viewable in the image. Evaluating each model with respect to the distance travelled, one can observe this trade-off more clearly, see Figure 5.5. Traversing along the prediction, the GNN model with no images, exhibits a larger error at the start of prediction, compared to the ResNet, yet a lower error at the end of prediction. Meanwhile, models that combine input modalities are shown to suppress this unconstrained behaviour at the start of prediction, although not perfectly. This is also qualitatively observable in Figure 5.2, where we can see how the GNN M model allows itself to cross the solid lines that separate the different driving directions of the road, whilst the other two configurations do not.

Finally, adding route information to the model appears to result in a surprisingly low performance increase, in some cases even a performance decrease. Interestingly, Table 4.2 indicates that the GNN IMR model outperforms GNN IM on straighter and moderate turn scenarios, but falls behind on handling sharp turns which gives it a slightly lower performance in terms of loss. This behaviour will be elaborated further upon in the next section.



(a) (b) (c)

Figure 5.4: Distribution of the predicted trajectories for the GNN models. (a) shows the distribution for the M configuration of the GNN extractor, (b) the IM configuration, and (c) the IMR configuration.

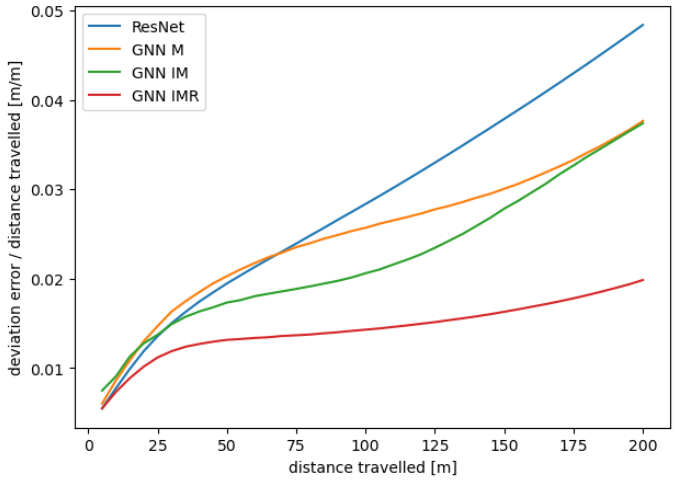


Figure 5.5: The deviation error over distance travelled of the predicted future trajectory of the ego vehicle for all sets of configurations of the GNN extractor. Note that the error has been scaled by the same distance travelled in order to further pronounce differences between individual configurations of the models.

5.2 The effects of applying oracles to GNNs

As touched upon in Section 5.1.2, the route seems to, somewhat surprisingly, not improve the model's prediction performance. We argue that this may come from this particular model's lack of trust in the modality. Furthermore, when observing cases where sharp turns are a factor, as in Figure 5.2, we observe a general performance that is, to a degree, disappointing if one considers the prediction quality at the end of each trajectory. Due to this, two oracles have been tested in Section 4.1.1 of whom's results are discussed here. We start by discussing the implications of having better maps when using GNN extractors and continue by elaborating on the effects of using a higher quantity of route data.

5.2.1 Implications of denoising the maps

In Table 4.3 results from studying the effects of denoising the maps by recovering previously mentioned miss-alignments with it the the ground truth, are presented. The results indicate that the overall performance improved by approximately 10% - 20% across all metrics when trained with OXTS data. This finding suggests that the models learn better when utilizing more accurate data. One reason for this is indeed, as illustrated in Figure 3.7, the shifting of the map links to align with the ground truth. This removes the requirement on the model to compensate for potential inaccuracies or to distrust the corrected map link.

As mentioned in Section 4.1.1, a concern arising from leveraging oracles such as this augmentation of the map data is that it turns the model brittle towards cases where the non-improved data is used during evaluation. In the case of our model, we can see that this is not the case, judging from the results shown in Table 4.4. Although the model, take IMR as an example, did not perform as well when evaluated with GNSS data as it did with OXTS data, the model still performed approximately 10% better than the model trained with GNSS data, when both are evaluated on GNSS data. Ergo, there seems to be little to relatively no drawback with training using, the perhaps more naive, OXTS data.

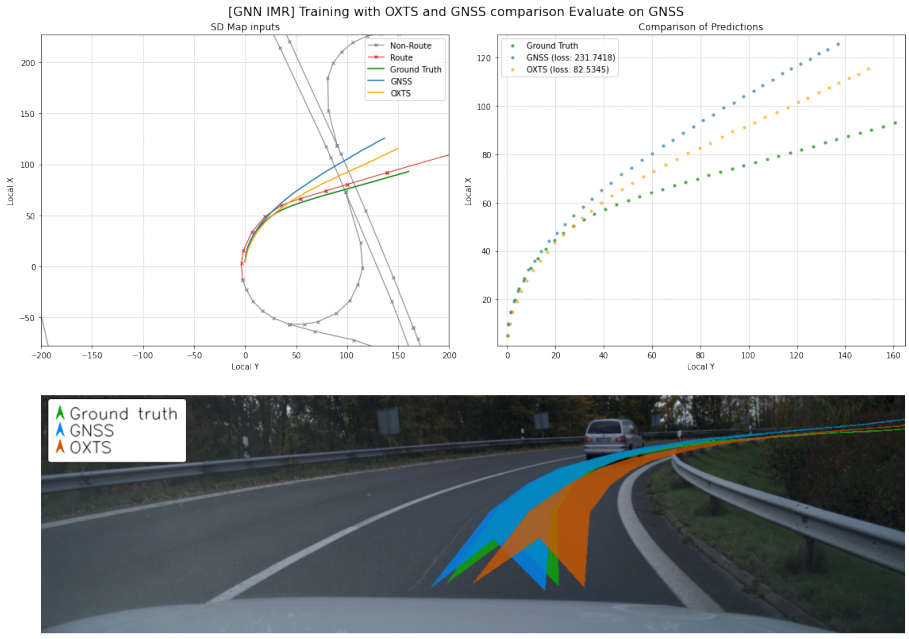


Figure 5.6: Predictions during inference of the GNN IMR model trained on OXTS and GNSS and evaluated on GNSS data.

From the above discussion, we showcase an example of the results in Figure 5.6, focusing on a turning case where the ego-vehicle is affected by miss-alignments, resulting in incorrect heading. The data trained with OXTS align more accurately with the corrected path and ground truth.

5.2.2 GNNs with different quantities of routes

As an additional bonus, the implementations discussed previously provide a slightly clearer differentiation of performance between the IM and IMR configured models, in favour of the IMR model. This is further pronounced by extending the quantity of data points in the data set that contain routes. As shown in Table 4.5, the GNN IMR model with 100% route data shows a significant improvement across all metrics compared to the model with 50% mixed data.

At the same time, when evaluating the model trained with 50% mixed route data and tested with 100% route data, the performance did not show any improvement. We believe that this is caused by the inclusion of what is effectively 50% false negative data points during training, which causes the model to distrust cases where the data is actually true. This distrust then leads to a general performance level that is similar to the level of the IM model, i.e. the case when routes are not used.

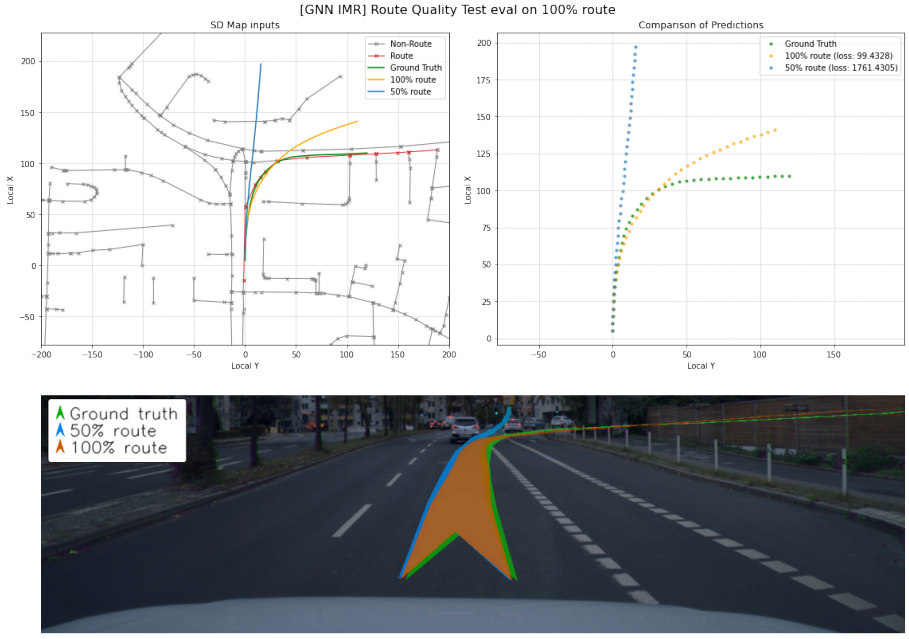
In contrast, having increased quantities of this class of data during both training and testing significantly improves performance. This setup allows the model to learn to trust the route information, as it no longer encounters false negatives. Additionally, when evaluated on the 50% mixed data, the model seems to revert back to performance levels akin to the IM model. As we do not see a strict degradation of the model performance if exposed to a test environment of mixed data, compared to models trained entirely on mixed data, we argue that the model still contextualizes the general flow of the map elements and does not overfit to the route data.

One clear example of how route data improves prediction accuracy is shown in Figure 5.7. The plot compares two different data points evaluated using models trained with varying route data availability but tested on data where route information is fully available. Figures 5.7a and 5.7b illustrate that, for some cases, the image and map information alone are insufficient to provide a correct ego trajectory prediction at the end of the trajectory due to the limitations of image horizon but more importantly due to ambiguities in the map.

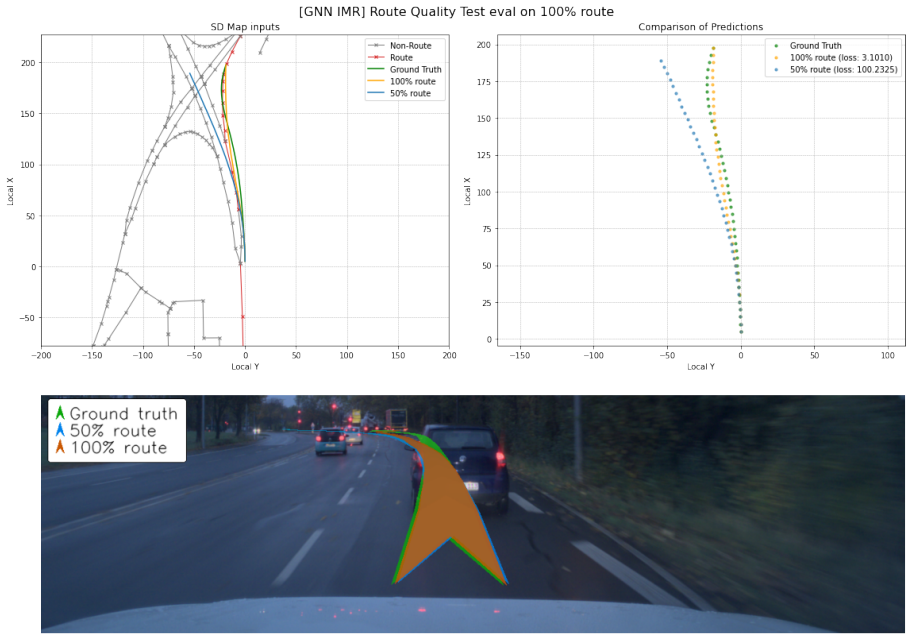
In Figure 5.7a, the image and map context suggest that the ego-vehicle can either go straight or turn right. Similarly, in Figure 5.7b, the ego-vehicle has the option to go straight or turn right onto the off-ramp. The model trained with 50% route data, as hypothesized, does not adequately consider the route information and incorrectly predicts that the vehicle will go straight. Conversely, the model trained with 100% route data accurately predicts the correct path, as evidenced by the alignment of predicted trajectories with the actual route.

Additionally, we can also see that the model learns to handle the false positives introduced by our greedy route generation algorithm in Figure 5.8. In both Figures 5.8a and 5.8b, the map and the image may present ambiguities by providing several possible paths, and even though some elements of the maps are marked incorrectly as routes, the model still predicts the target trajectory quite correctly.

These examples highlight the critical importance of the existence of route data to guide the model to make accurate path predictions, especially in scenarios where the map alone presents multiple possible future paths.

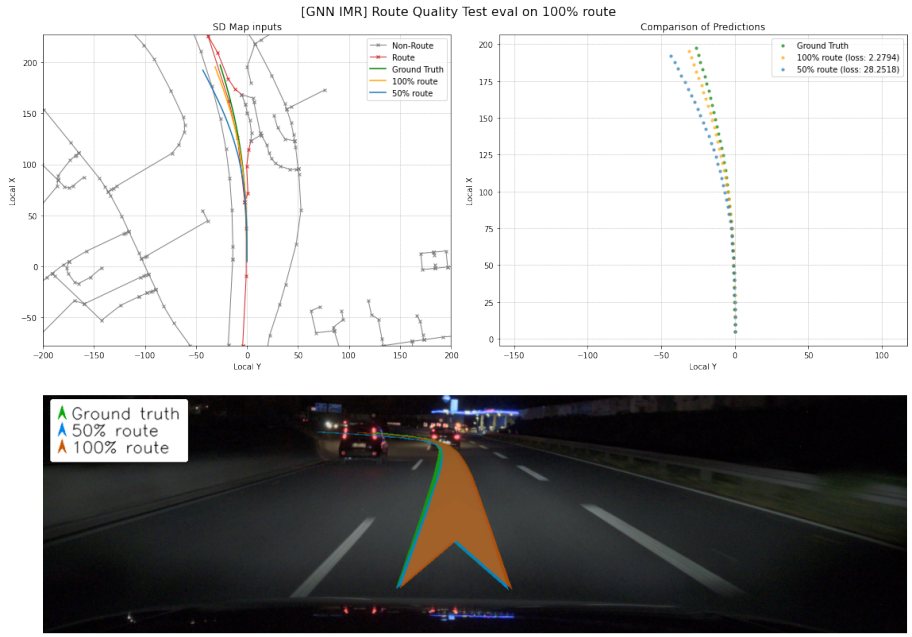


(a)

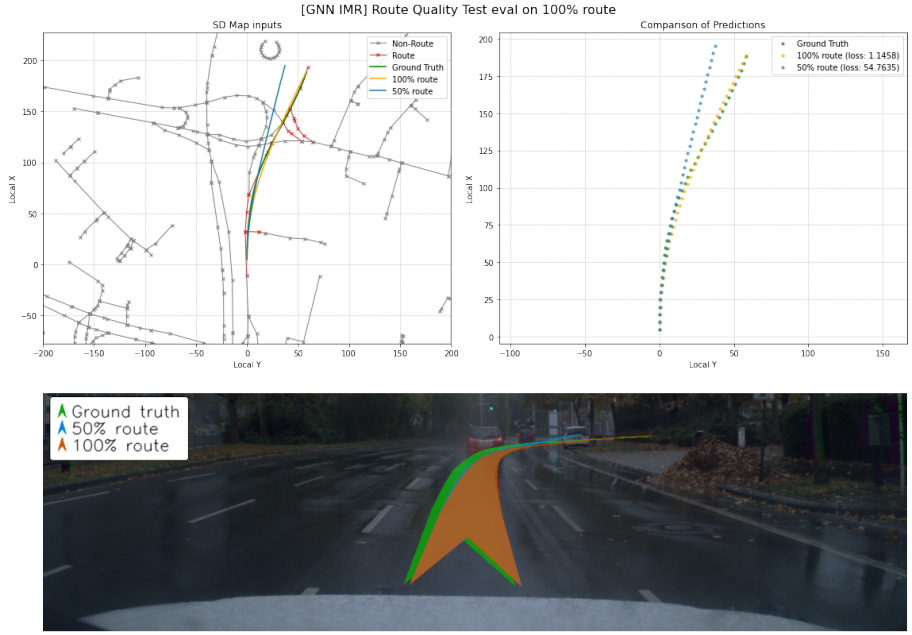


(b)

Figure 5.7: Comparison of the GNN IMR model's predicted trajectories with different training route data percentages, evaluated on 100% route data.



(a)



(b)

Figure 5.8: Comparison of the GNN IMR model's predicted trajectories with different training route data percentages, evaluated on 100% route data. Additionally, these scenarios contain route information with incorrect elements.

5.3 GNNs with the map modality removed

To further understand the impact of route data on model performance, we conducted an experiment where only the route information was provided to the model. This was done to isolate the effect of route data and evaluate its influence independently.

As previously discussed, the GNN IMR model with 100% route data showed significant improvements in performance metrics. When examining the results in Table 4.6, we observed that the GNN IR model improved in terms of L2 loss, showing a slight decrease. However, other metrics did not show similar improvements, instead, the GNN IMR performed better in terms of FDE, ADE, and MR.

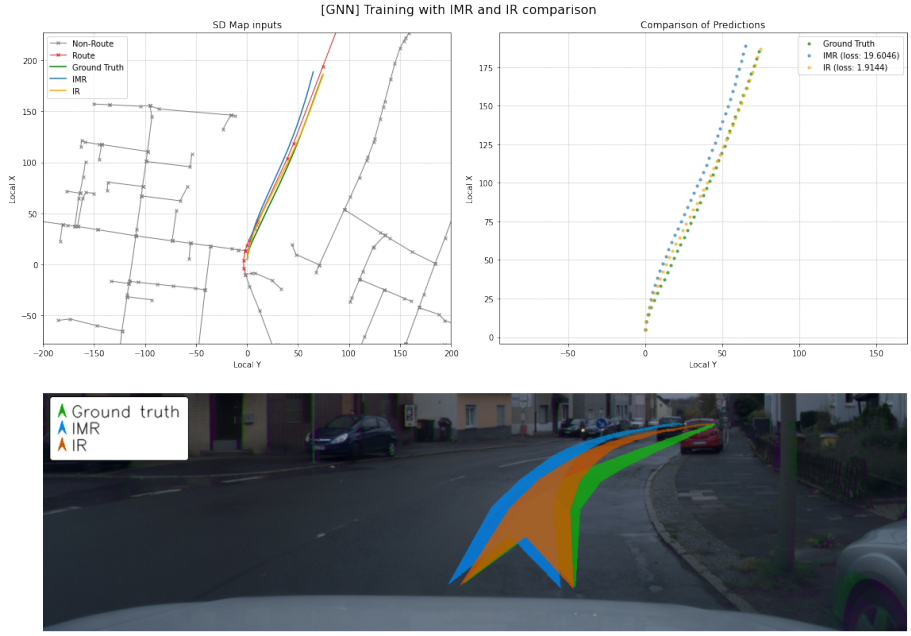
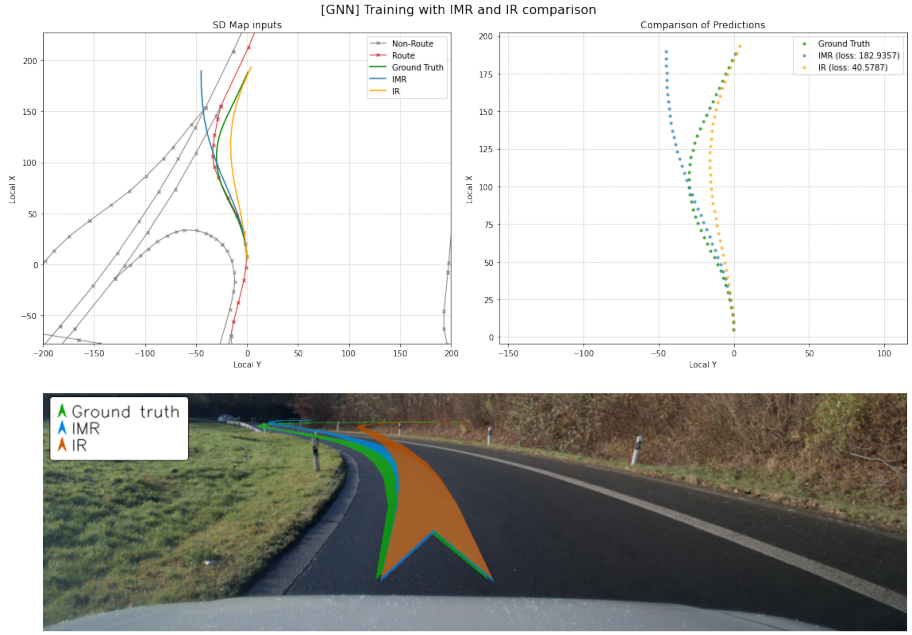
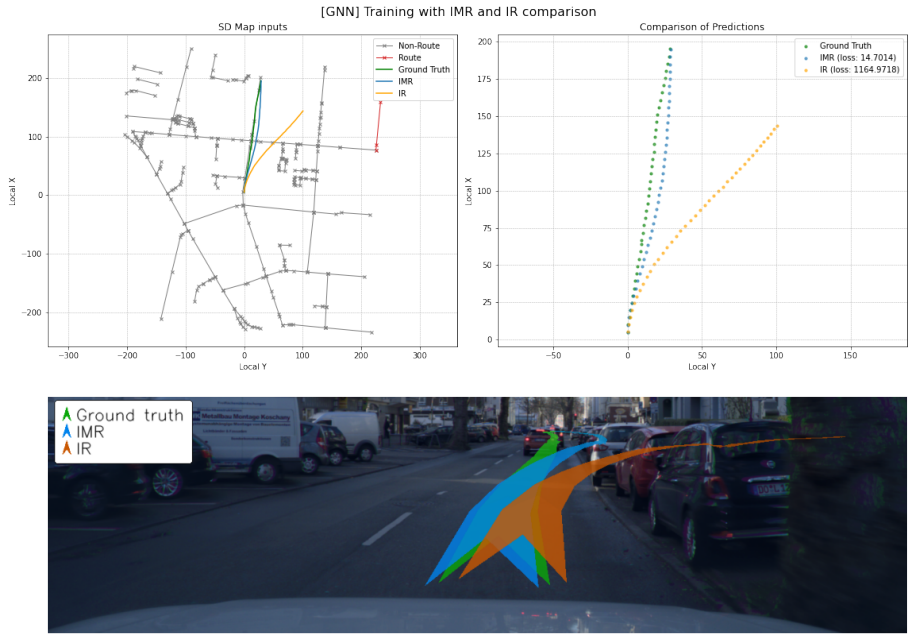


Figure 5.9: Predictions during inference of the GNN IMR model and IR model. Observe that, for the IR model, all map elements shown in the top left image are in fact non-existent (except for the route elements).

As exemplified in Figure 5.9, the route data tends to be part of the subset of the most relevant (closest) elements of the map. Ergo eliminating non-route elements may exclusively eliminate non-relevant information. Simple cases, like the one shown, can easily be handled by both GNN IMR and GNN IR.



(a)



(b)

Figure 5.10: Comparison of the GNN IMR and the GNN IR models in harder cases.

Figure 5.10 illustrates that in more complex cases, some map context might be needed. Starting with Figure 5.10a, an S-curve case with a limited horizon, the GNN IR shows better performance in terms of L2 loss, but its predicted trajectory is not preferable. The GNN IR model, given only the route of the S-curve, tries to take the shortest path, which results in predicting the car to go off the road, compromising safety. On the other hand, the GNN IMR, despite lower overall L2 loss, produces an accurate prediction within the context of information visible in

the image but fails to follow the S-curve trajectory at the end of the prediction.

In the next case, shown in Figure 5.10b, where the provided route is incorrect, the GNN IR model follows the wrong direction due to its high reliance on the route. Although the GNN IR model should be able to extract enough information from the context of the image to correctly produce a straight trajectory, it fails due to erroneous route information. Conversely, the GNN IMR model, although slightly off initially, likely due to the route information being located far away to the right, ultimately follows the correct direction, likely leveraging the map context for better decision-making.

Training with only route information generally provides sufficient context for ego trajectory prediction. However, in cases where the provided route is wrong or the trajectory is complex, important context may be lost. Thus, from our previous discussion in section 5.2.2 about the importance of having routes frequently available, it is clear that while route information can be highly informative, the map context is still necessary for making safe and robust predictions.

5.4 Cross-attention as fusion method

As mentioned in the beginning of this chapter, we discuss the implications of altering the way that the two (or three) input modalities are fused. According to the results in Table 4.7, the concatenation fusion strategy remains the best-performing method.

Employing cross-attention for fusion, where the map features are used as queries and keys and the image features as values, did not improve the model's performance. With support from the qualitative result shown in Figure 5.11, we think that the cross-attention operation helps to affirm the parts of the prediction where the source information of both feature vectors are in agreement. As seen in the aforementioned s-curve scenario the queried features extracted from the map seem to improve the part of the prediction that the image features most likely are quite correct about, i.e. at the start of the trajectory. Or, the perspective is reversed, it is the correctness of the queried map that accounts for the possible improvements of the image features and the seemingly decreasing effects of the usage of the operation at the end of prediction stems from the decrease of the accuracy of the map features at this point.

Figure 5.12, gives a quantified backing to these arguments. Here, one can note a slight, if perhaps minuscule, increase in performance at the earliest stage of the prediction if cross-attention is being utilized compared to when the concatenation operation is used. From this information, the improvement of the prediction in comparison to the baseline is apparent and illustrates that the information extracted from the map does play a part in the tail of the prediction, however not as great as when concatenation is used. Possibly a larger number of layers of cross-attention operations can remedy this issue.

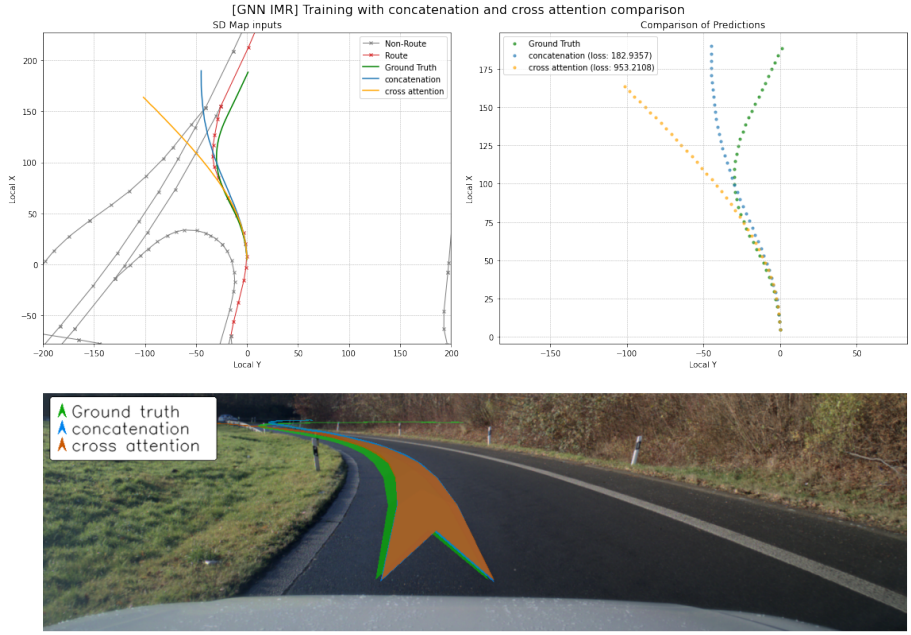


Figure 5.11: The s-curve scenario with the qualitative performance of the cross-attention operation in relation to the concatenation operation. Both models use a GNN IMR model as an extractor of the input data.

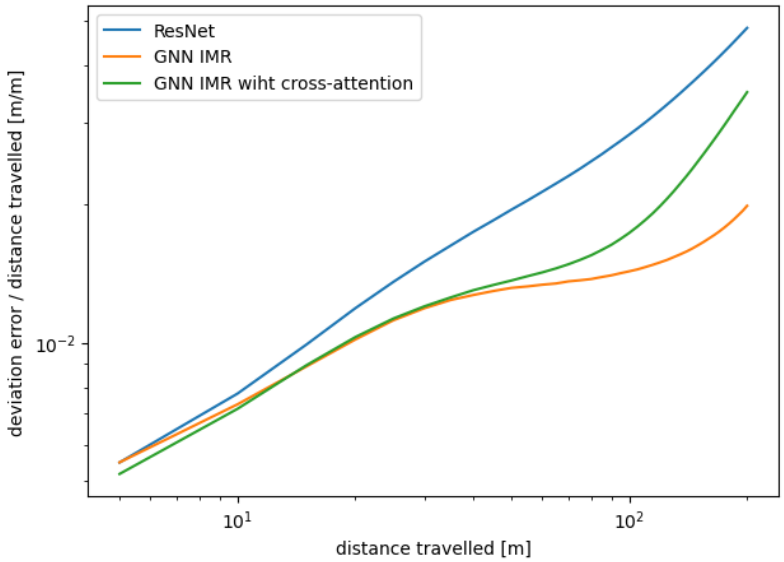


Figure 5.12: The deviation error over distance travelled of the predicted future trajectory of the ego vehicle for different fusion methods, along with the baseline model. Note that the error has been scaled by the same distance travelled in order to further pronounce differences between individual configurations of the models.

5.5 Discussion of the auxiliary task

As discussed in section 4.3, we explored an additional method to enhance our predictions by employing an auxiliary task. This approach aims to force the network to

align more closely with the map links. In the experiment summarized in Table 4.8, we tested our best-performing model, the GNN IMR, by incorporating an auxiliary loss during the backpropagation pass, using data containing 100% routes for testing.

However, the results indicated that the auxiliary loss did not lead to any improvements, i.e. the performance remained similar to that of the current baseline model. Metrics such as FDE and ADE should have decreased if the auxiliary loss had effectively aligned the predictions with the map, but this was not observed. The lack of impact in terms of performance metrics also translates to the qualitative results. In Figure 5.13 one can note that the auxiliary task does not help to remedy miss-alignments between the tail of the prediction and the map elements.

As a basic hypothesis, we think that this means that the manifold of the auxiliary loss shares similar properties with the normal loss. This would imply that in cases where the normal loss shows signs of residing on a plateau or in a local minima, i.e. gradients are small, the auxiliary loss exerts the same state. Essentially, we argue that in cases where it is hard to minimize the normal loss, it is also hard to minimize the particular auxiliary loss.

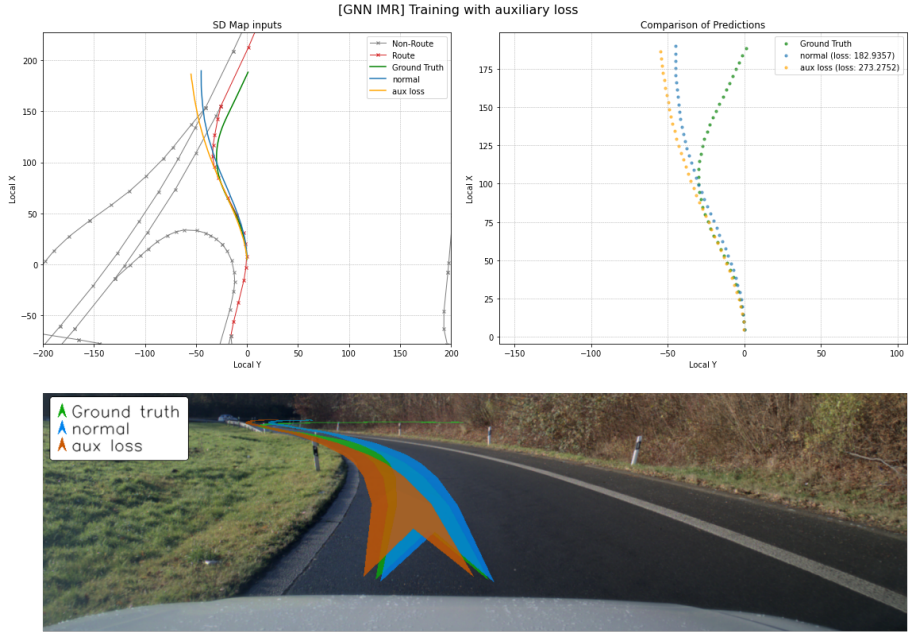


Figure 5.13: The S-curve scenario with the qualitative performance of the addition of the auxiliary task. Both models use a GNN IMR model as the extractor of the input data.

5.6 Discussion of the multimodal method

Based on the quantitative results in Table 4.9, we propose that, using our method of conducting multimodal predictions and evaluating them, more queries seem to increase prediction performance in the naive methods of measuring performance, i.e. absolute minADE, etc. Meanwhile, the confidence in the single best prediction, the

"min" prediction, seems to degrade when the number of queries are increasing, see the increasing difference between minFDE and β-minFDE. In fact, we argue that we're actually not learning to predict confidence at all.

In Figure 5.14, we show three scenarios where a multimodal model of nine queries is utilized. Consolidating the range of confidence values and their region of appearance in the output, we observe that an almost constant set of probabilities is assigned to the output distribution, independent of the input. In particular, we observe that the model is always confident in the trajectories in the immediate straight direction from the vehicle whilst trajectories of increasing curvature are assigned increasingly lower probabilities.

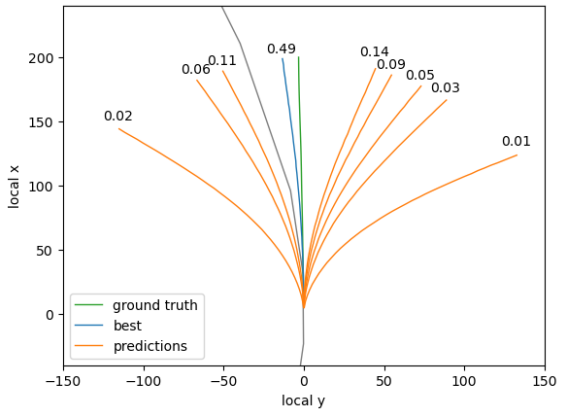
Possibly, the method that we use for rewarding the network for predicting the actual path does not capture the possibility of having multiple feasible trajectories within one scenario. In the scenario of Figure 5.14b, one can imagine at least two feasible trajectories, one taking the off-ramp and one continuing along the current road. However, in its current implementation, the network is encouraged to optimize the prediction towards the actual path in order to minimize its classification error, feasibility is instead implied through the existence of several similar scenarios where the ground truth label is assigned to the other option. Using this argument, confidence in feasible but incorrect trajectories can only occur by essentially tricking the network into believing that it is correct. From this point of view, it is not unsurprising that assigning these probabilities based on the distribution of correct paths in the training data is the easiest, and maybe the only, thing the network can learn.

Similarly, we note that the individual shapes of the predicted trajectories are independent of the available shapes in the input. This is particularly apparent when consolidating the s-curve in Figure 5.14 where, despite the s-shape, curving trajectories arch in a fanned pattern, strictly flowing away from the centred heading of the vehicle.

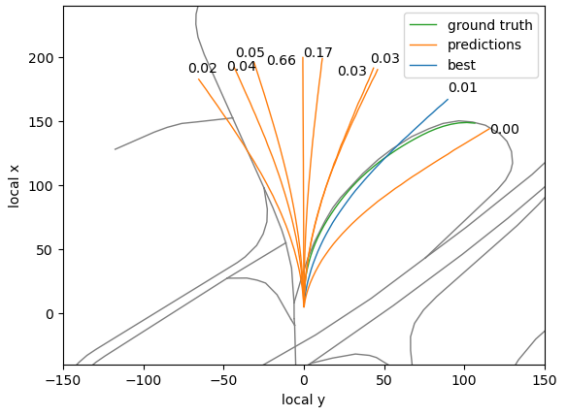
Our initial motivation for modelling multiple predictions through queries came from a hypothesis that each query could represent a class of trajectories within the data. That means that one query would perhaps map to some group of left turns while another would map to a different group of right turns and as such produce distinct proposals for the future trajectory of the vehicle. By examining Figure 5.15, one can observe a somewhat similar distribution of trajectories between each query, implying that our model does not exert this behaviour clearly. This is also true for the distribution of shapes of the outputs, see the same figure. Possibly, higher-order connections, independent of the region of direction of the prediction, are instead utilized to distinguish queries from one another, as queries still seem to map to distinct trajectories within each data point, as seen in Figure 5.14.

We suspect that this regression of higher-order features may be a symptom of an inability of the model to converge deeply. This issue of converging to a local minima is likely the same issue that results in the unspecific assignment of confidence values mentioned earlier. Judging from the fact that the position at where the estimated trajectories are centred seems to vary between inputs, in Figure 5.14b the mean position of the average trajectory is shifted slightly to the right compared to the

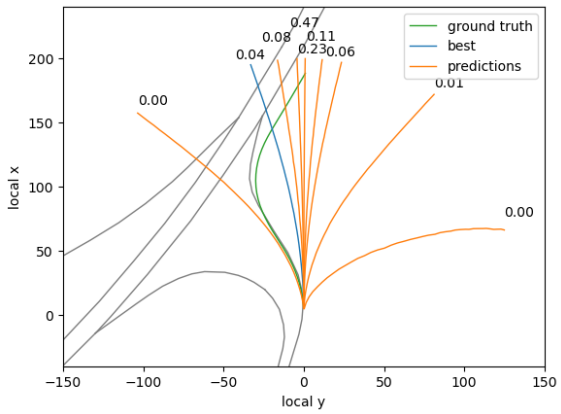
other two examples, we propose that the model, at this local minima, indeed has learnt to map said higher-order features to the output domain. However, due to the lack of further supporting evidence, one must also consider the possibility that this model has converged to a state where it is unable to establish a mapping between the specific input and output, and thus only produces predictions based on the distribution of the output set.



(a)



(b)



(c)

Figure 5.14: Qualitative predictions of multimodal model with 9 queries for three distinct scenarios: (a) a sparse map, (b) the off-ramp shown previously, and (c) the s-curve (also shown previously). Confidences are marked at the endpoint of each prediction.

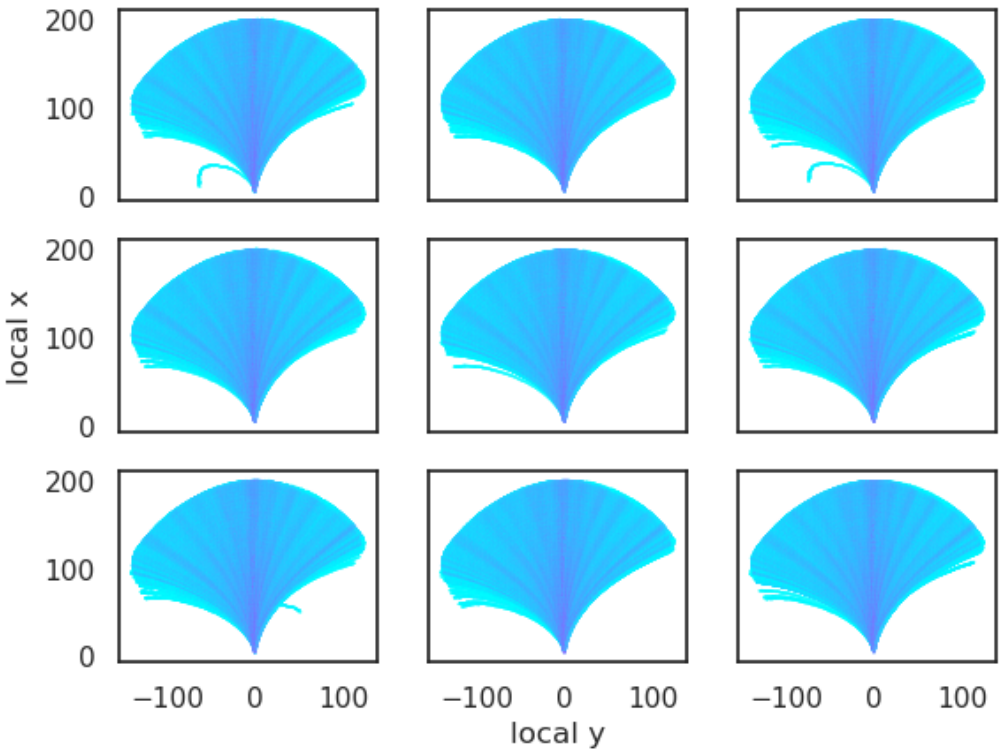


Figure 5.15: The distributions of outputted trajectories sorted per query (or prediction) of the multimodal model.

6

Conclusion

Based on the comprehensive analysis and experimental results presented in this thesis, it is evident that integrating multiple input modalities, such as front-facing camera images and standard definition (SD) maps, significantly enhances the performance of ego trajectory prediction models. GNN-based models consistently outperformed the baseline models such as ResNet and Transformer models across several evaluation metrics. This highlights the effectiveness of GNNs in capturing the spatial relationships inherent in map data and indicates that GNNs are the most effective method for encoding and leveraging map data for trajectory prediction, with IMR being the best configuration.

Different configurations of the GNN models, particularly the GNN IM and GNN IMR, showed substantial performance improvements if the data supplied during training was picked carefully. The accuracy of the alignment of the map plays a crucial role in model performance. Models trained with high-precision heading data (OXTS) demonstrated better performance compared to those trained with commercial GNSS data, underscoring the need for accurate map alignment with ground truth data. Additionally, the availability of route data significantly boosts prediction accuracy. While the GNN IMR model showed significant improvements in most scenarios, its performance in sharp turn scenarios was less pronounced, suggesting a need for further research into handling complex manoeuvres and dynamic changes in trajectory.

Apart from these findings, additional methods such as implementing auxiliary tasks, cross-attention fusion, and multimodal predictions, with the methodology we employed, did not show significant improvements. We, therefore, suggest that future work explore other innovative methods to further enhance model performance within these stages of the IMR model.

Bibliography

[1] National Highway Transportation Safety Administration (NHTSA), *What percentage of car accidents are caused by human error?* www.cbmclaw.com, 2016.

[2] S. Mozaffari, O. Y. Al-Jarrah, M. Dianati, P. Jennings, and A. Mouzakitis, "Deep learning-based vehicle behavior prediction for autonomous driving applications: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 33–47, Jan. 2022, ISSN: 1558-0016. DOI: 10.1109/tits.2020.3012034. [Online]. Available: http://dx.doi.org/10.1109/TITS.2020.3012034.

[3] M. Bojarski, D. D. Testa, D. Dworakowski, *et al.*, "End to end learning for self-driving cars," *arXiv: Computer Vision and Pattern Recognition*, 2016.

[4] T. Pohlen, A. Hermans, M. Mathias, and B. Leibe, "Full-resolution residual networks for semantic segmentation in street scenes," 2017. DOI: 10.1109/CVPR.2017.353.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2016. DOI: 10.1109/CVPR.2016.90.

[6] M. Wang *et al.*, "Ganet: Goal area network for motion forecasting," *arXiv preprint*, 2022.

[7] S. Casas, W. Luo, and R. Urtasun, "Intentnet: Learning to predict intention from raw sensor data," *CoRR*, vol. abs/2101.07907, 2021. arXiv: 2101.07907. [Online]. Available: https://arxiv.org/abs/2101.07907.

[8] J. Gao *et al.*, "Vectornet: Encoding hd maps and agent dynamics from vectorized representation," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[9] M. Liang, B. Yang, R. Hu, *et al.*, "Learning lane graph representations for motion forecasting," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds., Cham: Springer International Publishing, 2020, pp. 541–556, ISBN: 978-3-030-58536-5.

[10] M. Ye, T. Cao, and Q. Chen, "TPCN: temporal point cloud networks for motion forecasting," *CoRR*, vol. abs/2103.03067, 2021. arXiv: 2103.03067. [Online]. Available: https://arxiv.org/abs/2103.03067.

[11] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, "Home: Heatmap output for future motion estimation," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, 2021, pp. 500–507. DOI: 10.1109/ITSC48978.2021.9564944.

[12] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, "GOHOME: graph-oriented heatmap output for future motion estimation," *CoRR*,

vol. abs/2109.01827, 2021. arXiv: 2109.01827. [Online]. Available: https://arxiv.org/abs/2109.01827.

[13] J. Gu, C. Sun, and H. Zhao, "Densetnt: End-to-end trajectory prediction from dense goal sets," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 15 283–15 292. DOI: 10.1109/ICCV48922.2021.01502.

[14] B. Wilson, W. Qi, T. Agarwal, *et al.*, *Argoverse 2: Next generation datasets for self-driving perception and forecasting*, 2023. arXiv: 2301.00493 [cs.CV].

[15] J. Cheng *et al.*, "Forecast-mae: Self-supervised pre-training for motion forecasting with masked autoencoders," *arXiv preprint*, 2023.

[16] H. Chen, J. Wang, K. Shao, *et al.*, *Traj-mae: Masked autoencoders for trajectory prediction*, 2023. arXiv: 2303.06697 [cs.CV].

[17] Z. Lan, Y. Jiang, Y. Mu, *et al.*, *Sept: Towards efficient scene representation learning for motion prediction*, 2023. arXiv: 2309.15289 [cs.CV].

[18] Y. Liu, T. Yuan, Y. Wang, Y. Wang, and H. Zhao, *Vectormapnet: End-to-end vectorized hd map learning*, 2023. arXiv: 2206.08920 [cs.CV].

[19] K. Z. Luo, X. Weng, Y. Wang, *et al.*, *Augmenting lane perception and topology understanding with standard definition navigation maps*, 2023. arXiv: 2311.04079 [cs.CV].

[20] J. Schmidt, J. Jordan, F. Gritschneder, T. Monninger, and K. Dietmayer, "Exploring navigation maps for learning-based motion prediction," 2023. DOI: 10.1109/ICRA48891.2023.10160989.

[21] J.-Y. Liao, P. Doshi, Z. Zhang, D. Paz, and H. I. Christensen, "Osm vs hd maps: Map representations for trajectory prediction," *arXiv.org*, 2023. DOI: 10.48550/ARXIV.2311.02305.

[22] F. Rosenblatt, "The perceptron - a perceiving and recognizing automaton," Cornell Aeronautical Laboratory, Ithaca, New York, Tech. Rep. 85-460-1, Jan. 1957.

[23] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, ISSN: 1476-4687. DOI: 10.1038/nature14539. [Online]. Available: https://doi.org/10.1038/nature14539.

[24] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[25] Z. Zhou, L. Ye, J. Wang, K. Wu, and K. Lu, "Hivt: Hierarchical vector transformer for multi-agent motion prediction," *Computer Vision and Pattern Recognition*, 2022. DOI: 10.1109/CVPR52688.2022.00862.

[26] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009. DOI: 10.1109/TNN.2008.2005605.

[27] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," 2017.

[28] L. Achaji, T. Barry, T. Fouqueray, J. Moreau, F. Aioun, and F. Charpillet, "Pretr: Spatio-temporal non-autoregressive trajectory prediction transformer," 2022. DOI: 10.1109/ITSC55140.2022.9922451.

[29] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," *CoRR*, vol. abs/2005.12872,

2020. arXiv: 2005.12872. [Online]. Available: https://arxiv.org/abs/2005.12872.

[30] P. Nakkiran, G. Kaplun, Y. Bansal, T. Yang, B. Barak, and I. Sutskever, "Deep double descent: Where bigger models and more data hurt," *CoRR*, vol. abs/1912.02292, 2019. arXiv: 1912.02292. [Online]. Available: http://arxiv.org/abs/1912.02292.

[31] R. Huang, H. Xue, M. Pagnucco, F. Salim, and Y. Song, *Multimodal trajectory prediction: A survey*, 2023. arXiv: 2302.10463 [cs.RO].

[32] B. Liao, S. Chen, X. Wang, *et al.*, *Maptr: Structured modeling and learning for online vectorized hd map construction*, 2023. arXiv: 2208.14437 [cs.CV].

[33] H. Cui, V. Radosavljevic, F.-C. Chou, *et al.*, *Multimodal trajectory predictions for autonomous driving using deep convolutional networks*, 2019. arXiv: 1809.10732 [cs.RO].

[34] OpenStreetMap contributors, *Planet dump retrieved from https://planet.osm.org*, https://www.openstreetmap.org, 2021.

A

Appendix 1

A.1 Multimodal Decoder Architecture

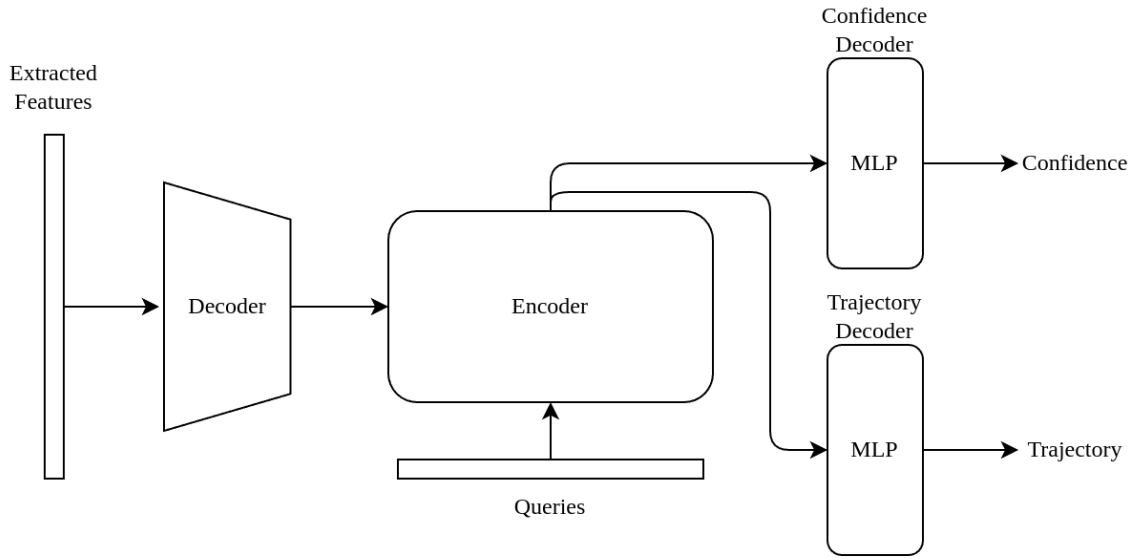


Figure A.1: Schematic describing the architecture of the multimodal decoder proposed in the thesis. The figure depicts an encoder decoder model, implemented after the DETR-model[29]. The specific encoder- and decoder architectures are based of the *Attention is all you need* paper[27]. This module replaces the previous MLP in Figure 3.1.

B

Appendix 2

B.1 Greedy Algorithm for Generating More Routes

In situations where route information is unavailable, a greedy algorithm is employed to generate routes based on the vehicle's projected trajectory and the map data. The algorithm works as follows:

Algorithm 1 Greedy Algorithm for Route Prediction

Require: Map data *map_data*, trajectory data *y_data*
Ensure: Updated map data with routes

Initialize empty lists *ll_points* and *ll_points_link*
for each link in *map_data* **do**
    for index from 0 to length(link.points) - 2 **do**
        Form a line from *link.points*[*index*] and *link.points*[*index* + 1]
        Append line to *ll_points*
        Append *link.attributes.link* to *ll_points_link*
    end for
end for
▷ If Route is available return the current map data
if *ll_points* is empty **then**
    return *map_data*
end if
▷ Find route from traversal link
Find the closest projections of *y_data* on *ll_points*
Initialize empty list *links*
for link in closest_points_index **do**
    if *current_link* is traversable from *previous_link* and *previous_link* != *current_link* **then**
        Append *current_link* to *route_link*
        Update *previous_link* to *current_link*
    end if
end for
▷ Generate Route from the *route_link*
for each link in *map_data* **do**
    if link.attributes is in *route_link* **then**
        Set *link.attributes.is_route* to 1
    end if
end for
return *map_data*