OPEN ACCESS
International Journal of Semantic Computing
Vol. 17, No. 3 (2023) 349–370
© The Author(s)
DOI: 10.1142/S1793351X23620040

Maneuver Prediction Using Traffic Scene Graphs via Graph Neural Networks and Recurrent Neural Networks

Petrit Rama* and Naim Bajcinca†
Department of Mechanical and Process Engineering
Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau (RPTU)
Gottileb-Daimler-Straße 42, 67663 Kaiserslautern, Germany
*petrit.rama@mv.uni-kl.de
†naim.bajcinca@mv.uni-kl.de

Received 25 October 2022
Revised 1 April 2023
Accepted 26 May 2023
Published 9 August 2023

The driving process involves many layers of planning and navigation, in order to enable tractable solutions for the otherwise highly complex problem of autonomous driving. One such layer involves an inherent discrete layer of decision-making corresponding to tactical maneuvers. Inspired by this, the focus of this work is predicting high-level maneuvers for the ego-vehicle. As maneuver prediction is fundamentally feedback-structured, it requires modeling techniques that take into consideration the interaction awareness of the traffic agents involved. This work addresses this challenge by modeling the traffic scenario as an interaction graph and proposing three deep learning architectures for interaction-aware tactical maneuver prediction of the ego-vehicle. These architectures are based on graph neural networks (GNNs) for extracting spatial features among traffic agents and recurrent neural networks (RNNs) for extracting dynamic motion patterns of surrounding agents. These proposed architectures have been trained and evaluated using BLVD dataset. Moreover, this dataset is expanded using data augmentation, data oversampling and data undersampling approaches, to strengthen model's resilience and enhance the learning process. Lastly, we compare proposed learning architectures for ego-vehicle maneuver prediction in various driving circumstances with various numbers of surrounding traffic agents in order to effectively verify the proposed architectures.

Keywords: Maneuver prediction; decision-making; autonomous driving; interaction graphs; graph neural networks (GNNs); recurrent neural networks (RNNs).

1. Introduction

The process of driving a vehicle is highly interactive experience, constantly influenced by both static and dynamic entities and features of the traffic scenario. Human driving involves numerous layers of tasks to navigate the road network,

*Corresponding author.

starting from finding the optimal global and local path, observing the behavior of surrounding traffic agents, navigating while engaging with involved agents and executing low-level driving actions. One such task of human driving involves an inherent discrete layer in decision-making, corresponding to specific tactical maneuvers such as right or left turns, lane changing, straight driving, stopping, etc. It becomes sensible to inherit this at a higher level of a hierarchical assembly in machine driving as well, in order to provide tractable solutions for the otherwise extremely challenging problem of autonomous driving. Autonomous driving is one of the most active research domains nowadays, given that it crosses numerous research disciplines, including environment perception, localization, behavior recognition, situational awareness, planning, navigation and control algorithms.

For autonomous driving, motion prediction is highly important and critical. Lefèvre *et al.* in the survey [1], categorized motion modeling and prediction approaches of traffic agents into physics-based, maneuver-based and interaction-aware. Physics-based approach considers laws of physics for motion prediction, which makes this approach the simplest one. Maneuver-based approach predicts the independent high-level movement of a traffic agent, restricted by static features of the road (i.e. number of lanes, width of the lane or maximum speed), and movement capabilities of the traffic agent, without considering nearby agents. And lastly, the approach which is gaining more attention recently, the interaction-aware approach considers the interactive relationship between traffic participants to predict the future motion of ego-vehicle. This approach accounts for the interactive nature of the driving process, anticipating the influence of involved traffic agents, which leads to a better understanding of traffic situations and a more reliable motion prediction.

Based on the output of the models, motion prediction models are categorized into maneuver prediction, trajectory prediction or maneuver-based trajectory prediction. A lot of research and progress on trajectory prediction has been reported in literature, using classical methods or data-driven methods recently [2–5]. Many studies and developments have gone into maneuver-based trajectory prediction as well, which involves first predicting a maneuver and then fitting a trajectory to that particular maneuver, as stated in the literature [6, 7]. However, fewer results related to the problem of maneuver prediction have been reported in literature. This is explained by the lack of datasets and labels with explicit driving maneuvers, as it requires more effort and expertise to annotate different traffic scenarios for maneuver planning. One such dataset, published relatively recently is BLVD dataset [8], which is used as the main dataset for this work.

Ranney in his work [9] categorized maneuver planning in three levels of timescale. Strategic maneuver planning is a long-term planning problem, concerned with planning maneuvers for reaching the destination from the origin, accounting also for other important aspects of planning. On the other hand, tactical maneuvers are in the timescale of seconds, and include a set of operations to achieve a short-term goal, such as lane change, turns or stopping. Lastly, operational maneuvers, which are in the timescale of milliseconds, include critical operations while driving, to maintain safety and follow traffic rules. This work presents three architectures for predicting tactical maneuvers for the ego-vehicle in divers traffic scenarios.

Our everyday driving practice demonstrates that a driving decision of a traffic agent can heavily impact decisions of other agents. Consequently, while modeling the problem of maneuver prediction, one has to consider the interaction between traffic agents. Due to their flexibility, scalability and high-level representation, graph-based techniques have recently been widely used for interaction modeling in many domains such system biology, social networks, recommendation algorithms, also in autonomous driving [10, 11]. Considering the crucial aspect of spatial modeling, this works also follows a graph-based modeling approach, utilizing graphs as data structures to model the interactive behavior between surrounding traffic agents, regardless of the locality. Whereby, nodes represent detected and tracked traffic agents, while edges represent the relative spatial interaction between them. Such graphs serve as inputs to graph neural networks (GNNs) for learning and extracting spatial features. Movement patterns of surrounding traffic agents and maneuver changes of the ego-vehicle are tracked over time via recurrent neural networks (RNNs). This work is an extension of the previous work done in NIAR [12], with the main focus to follow an interaction-aware approach for modeling diverse multi-agent traffic scenarios as interaction graphs and predict the best tactical maneuver for the ego-vehicle. The main contributions of this work are as follows:

- Representation of traffic scenarios as dynamic and interaction graphs, and modeling the impact of interactions between traffic agents for maneuver prediction;
- Proposing three different architectures based on GNNs and RNNs for solving the problem of tactical maneuver prediction for ego-vehicle in diverse traffic scenarios;
- Data resampling techniques, such as data augmentation, data oversampling, data undersampling or combination of them, to train and validate the proposed architectures, and also stabilize the learning process;
- Conducting ablative experiments to evaluate the impact of maneuvers, number of layers in GNNs and time-window in RNNs on the model's performance.

2. Related Works

The research landscape in motion prediction and decision-making for autonomous driving is mainly separated into three main categories. The first category predicts a trajectory for the ego-vehicle based on the previous states, solved as a regression task. The second category is concerned with predicting high-level movement in form of maneuver prediction, solved as a classification task. The third category models and solves the problem of motion prediction in two phases, determining the maneuver first, based on which a trajectory is predicted. These methods are powerful, but limited in case of wrong maneuver classification.

2.1. *Decision-making using classical methods*

Decision-making has been the subject of research for some time now. Hermes *et al.* [13] used velocity and yaw angle components of the trajectory as input to a classifier to determine the best maneuver and then implement a probabilistic tracking framework to predict the motion state in the future. Hidden Markov Models were used in [14, 15] as a data-driven approach to classify a maneuver first, generating a probability distribution for each state over all possible vehicle's trajectory. Libener *et al.* [16] used Bayes net to develop a model to classify the maneuver of vehicles in complex intersection scenarios, based on vehicle's velocity profile. Houenou *et al.* [17] proposed a new approach based on heuristic classifiers to recognize a high-level maneuver of lane keeping or lane changing using lane marking distance, then using polynomial fitting for trajectory prediction. Schreier *et al.* [18] proposed a two-stage motion prediction model, first using Bayesian network to predict the state of all vehicles forward in time based on maneuver motion models, then predicting a joint probability distribution function for all possible maneuver-based trajectories.

2.2. *Decision-making using deep learning*

Data-driven techniques, machine learning specifically, have also been used to address the challenging problem of decision-making. These methods gained prominence since many research institutes and commercial companies around the world are producing huge volumes of data and making them accessible to the general public as traffic datasets. CoverNet [2] proposed a multimodal trajectory prediction model, framing the problem of high-level decision-making as a classification task, over all feasible maneuvers and generate the best trajectory based on the state space. Mo *et al.* [19] proposed an interaction-aware encoder–decoder trajectory prediction model for the ego-vehicle, based on long short-term memory (LSTM) for tracking dynamic features of surrounding vehicles and convolutional neural network (CNN) to extract spatial interactive features between traffic agents.

Due to the multimodal nature of driving, there are numerous options available when faced with a particular traffic situation. The biggest drawback of using only trajectory prediction is that the result frequently represents an average of several different options. A solution would be to predict the maneuver or predict a probability for each maneuver, over which specific trajectories are generated. Deo *et al.* [6, 7] proposed a model based on convolutional social pooling for extracting spatial inter-dependencies among agents, and LSTM encoder–decoder to preserve the time-context, predicting the probability distribution over possible maneuver-based trajectories.

Meanwhile, maneuver-based prediction methods are focused on modeling and solving the problem as a classification task. Schlechtriemen *et al.* [20] proposed a classification algorithm based on random forest, focused on predicting lane change or lane keeping maneuvers in highway scenarios, considering that the dataset is highly unbalanced and affected by noise and outliers. Khosroshahi in the work [21] proposed an architecture based on LSTMs, using ground plane coordinates of vehicles position and angular changes for maneuver classification at intersections.

2.3. *Decision-making using GNNs*

GNN is a relatively new branch of deep learning algorithms, which operate on graph-structured data, proposed from Scarselli *et al.* [22]. This work served as the basis for many other concepts and variants of GNNs, notably graph convolutional networks (GCNs), which generalized the convolution operator for graphs [23]. These models and other variants work using message passing framework (MPF) [24], with the main idea to aggregate information from neighboring nodes and update the representation of the central node. Research interest in GNNs is growing fast, already being used successfully in many domains, such as computer vision, natural language processing (NLP) or system biology. Recently, GNNs are also being used in decision-making systems of autonomous driving, where spatial dependencies and interactive behavior among traffic agents are modeled as interaction graphs.

Diehl *et al.* [3] analyzed empirically different aspects of modeling a traffic scene and utilizing GNNs for predicting the trajectory of vehicles, proving that adopting GNN models increased the prediction accuracy. Ciu *et al.* [25] used spectral convolutional graph neural network combined with LSTMs, to learn and extract localized spatial features from the traffic network, in order to predict network-wide traffic states. GRIP [4] represents the spatial interaction between traffic agents in form of a graph, proposing an architecture based on convolution layers to capture spatial features and LSTM encoder–decoder model to predict simultaneously the future trajectories for all observed traffic agents. Mo *et al.* [5] combined GNN to capture interaction features and RNN to capture dynamic features, in order to predict an interaction-aware trajectory, proving that the dynamic features of the target vehicle and the interaction with other vehicles affects the trajectory prediction accuracy. VectorNet [26] represents traffic components in form of vectors, modeling their interaction using a hierarchical GNN to learn the context features and their high-order interaction, in order to predict the trajectory of vehicles.

On the other hand, GNNs are also used for maneuver planning and prediction in autonomous driving. Chandra *et al.* [27] used spectral graph analysis and LSTMs to predict maneuvers and trajectories of surrounding traffic participants in urban traffic scenarios, representing the proximity among them as a weighted graph. Pan *et al.* [28] proposed a framework based on GNNs, LSTMs and attention mechanisms to utilize spatio-temporal graphs to learn the driver's intention of lane change and predict the trajectory of vehicles. Li *et al.* [10, 11] modeled the spatial interaction between heterogeneous road agents in form of a graph, using GNNs for spatial feature extraction and LSTMs for temporal features extraction, to recognize interactive behavior events and predict the trajectory of all heterogeneous road agents.

3. Methodology

Making decisions in complex traffic scenarios is a difficult and highly interactive process that is affected by all nearby traffic agents. Such traffic agents (i.e. vehicles, buses, motorcycles, pedestrians, etc.) move through a shared traffic area while navigating and interacting with other agents. A traffic scenario is defined by key elements that must be taken into account before modeling the problem and implementing a deep learning architecture for maneuver prediction. The first element is concerned with the dynamic nature of traffic scenarios, where traffic agents can move inside or outside the field of influence of other agents. Next element, agents which are inside this field of influence, have characteristic features which should be part of the problem modeling. The last element is to understand the scene context, by determining the driving action of nearby agents influencing the decision of ego-vehicle, in order to anticipate the movement of these surrounding agents for improving the accuracy of maneuver planning and prediction.

Following the work done in [12], traffic scenarios are represented as interaction graphs. Moving traffic agents are represented as nodes in the graph, while edges represent the interaction among traffic agents. An interaction graph, which abstracts a traffic scenario as a fluid and dynamic graph data structure, naturally represents the spatial interaction between traffic agents in different environments. Hence, a graph fulfills all requirements from the aforementioned elements, used in this work to model the problem. Agents move inside or outside of field of influence, reflected in the structure of the graph, adding or removing nodes, modeling it as a dynamic system. Characteristic features of agents (i.e. class, position or relative distance) are part of the interaction graph in form of node features and edge features, respectively. Lastly, inferred driving actions of agents from the scenario context are also modeled as node features, enriching the scenario graph. To solve the problem of decision-making, the traffic scenario needs to be considered as a whole, which is an advantage when representing the problem as a graph data structure.

Two aspect to consider for modeling and predicting maneuvers are the spatial interaction and temporal movement pattern. Therefore, graph structures are build to model such interactions among traffic agents, using a GNN module to extract such spatial feature. Leveraging GNNs for capturing spatial and interactive features between traffic agents has already been proven to increase the predictive power, compared to models that do not take into account this interaction [3]. On the other hand, temporal features of movement pattern are tracked in time using a RNN module. These two modules, GNN and RNN, are combined in three different ways in three different architecture proposed in this work. These different combinations give us an idea of how important are spatial and temporal features, hyperparameters of the architectures, and time complexity for training and evaluating them.

3.1. *Problem formulation*

Each frame of a traffic scenario is passed through an environment perception algorithm, where n traffic agents are detected and tracked. For each agent i at time t, a feature vector $f_i^{(t)}$ is detected, represented as:

$$f_i^{(t)} = [c_i^{(t)}, a_i^{(t)}, s_i^{(t)}, x_i^{(t)}, y_i^{(t)}, l_i^{(t)}, w_i^{(t)}, h_i^{(t)}, o_i^{(t)}], \tag{1}$$

where c is the class of traffic agent, a is the driving action, s is the description of the traffic scenario, x and y are positional coordinates, l is the length, w is the width and h is the height of the bounding box, and lastly o is the heading angle.

Such feature vectors $f_i^{(t)}$ are tracked for $\mathcal{T}$ timesteps, ordered in time in form of historical feature $x_i^{(t)}$ of agent i at time t, represented as:

$$x_i^{(t)} = \{f_i^{(t-\mathcal{T})}, f_i^{(t-\mathcal{T}+1)}, f_i^{(t-\mathcal{T}+2)}, f_i^{(t-\mathcal{T}+3)}, f_i^{(t-\mathcal{T}+4)}, \dots, f_i^{(t)}\}. \tag{2}$$

Such ordered historical feature $x_i^{(t)}$ are build for n detected and tracked traffic agent, stacked together in form of a feature matrix $\mathcal{X}^{(t)}$ at time t, represented as:

$$\mathcal{X}^{(t)} = \{x_1^{(t)}, x_2^{(t)}, x_3^{(t)}, x_4^{(t)}, \dots, x_n^{(t)}\}. \tag{3}$$

Given this feature matrix $\mathcal{X}^{(t)}$ at time t, a graph $g^{(t)}$ is build for every timestep, used as an input the model. The aim is to learn a mathematical function $F(\cdot)$ to map past $\mathcal{T}$ timesteps of n feature vectors represented as graph signals into one of many possible maneuvers for the ego-vehicle. The output of the model is the predicted maneuver for ego-vehicle at time $t+1$.

3.2. *Graph notation and construction*

A separate interaction graph $\mathcal{G}^{(t)}$ is constructed from every frame t of a traffic scenario. Such graphs are modeled as directed graphs, defined as $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$, where $\mathcal{V}^{(t)} = \{v_1, v_2, v_3, v_4, \dots, v_n\}$ is the set of n observed agents as nodes, and $\mathcal{E}^{(t)} \subset \mathcal{V}^{(t)} \times \mathcal{V}^{(t)}$ is the set of edges. The adjacency matrix $\mathcal{A}^{(t)} \in \mathbb{R}^{n\times n}$ of graph $\mathcal{G}^{(t)}$ shows which nodes are connected to each other. Every node v_i in the graph has its own feature vector f_i. The feature vector of the whole graph is defined as the feature matrix $\mathcal{X}^{n\times d}$, where d is the feature dimension.

The graph data structure models the interaction between n detected traffic agents represented as nodes, with $n+1$ total nodes in the graph including the ego-vehicle as an additional node. Moreover, edges represent interactive relationship among traffic agents. Every agent interacts with other nearby agents, therefore connecting every node with every other node in the graph, forming a complete graph in the process. In our work, we enrich the edge features with normalized weights, encoding the inverse Euclidean distance between the centerpoint of ego-vehicle and the centerpoint of the detected bounding box of other traffic agents in the image space. This indicates that traffic agents which are closer to the ego-vehicle will have a bigger impact on decision-making of the ego-vehicle, and traffic agents which are further away will have less impact on decision-making.

3.3. *Model architectures*

In the framework of this work, three different architectures based on GNNs and RNNs are proposed to model and predict tactical maneuvers for the ego-vehicle, shown in Fig. 1. The main building modules are GNNs for spatial feature extraction,

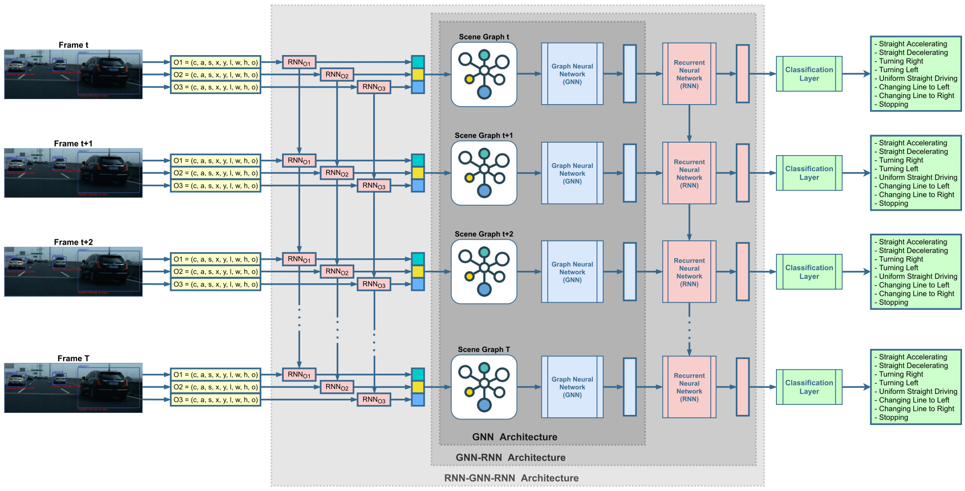


Fig. 1. Three proposed architectures with GNNs and RNNs for maneuver prediction.

and RNNs for temporal feature extraction. The first step is a shared step for all architectures, which is extracting objects and features for constructing interaction graphs. Such graphs serve as input to the GNN architecture. For the other two architectures, which include the RNN module, a further processing step is required for ordering traffic scene graphs in time, before giving it as input for the network. The last layer is also a shared layer for all architectures, which includes a softmax classification layer. Implementing three different architectures, different graph representations, hyperparameters, time windows and dataset variants illustrates how different methods or aspects can impact the prediction accuracy of maneuver planning. Three proposed architectures are explained in detail as follows:

(1) **GNN architecture:** This architecture consists of a multi-layer GNN module, which serves also as a baseline model. This architecture gives the initial overview of the best network hyperparameters, tuning them for the best accuracy and impact evaluation of spatial features for maneuver prediction. The architecture is shown in Fig. 1, including only the innermost part, colored in dark gray. Detected and tracked traffic agents which are inside the field of influence from the ego-vehicle are used as nodes in the graph. Every node is connected to every other node, constructing a directed complete graph $\mathcal{G}^{(t)}$ for timestep t. Extracted features are used as node and edge features, enriching the graph with traffic scenario information. Such graphs are used as inputs for the GNN module:

$$g^{(t)} = \mathrm{GNN}(\mathcal{G}^{(t)}), \tag{4}$$

where $\mathrm{GNN}(\cdot)$ is a multi-layer GNN function. The input graph passes through the message passing function and the node update function, as described in [24]. Node features are updated multiple time based on its own previous node feature, and neighboring node features and edge features, transformed into so-called node embeddings. The output of the module is a transformed graph, with a latent spatial representation $g^{(t)}$ for every node in the graph. A layer-wise GNN operation for node i is defined as:

$$g_i^{(l+1)} = \varphi_{\text{upd}}(x_i^{(l)}, \varphi_{\text{agg}}(x_i^{(l)}, x_j^{(l)}, e_{ij}^{(l)})), \tag{5}$$

where $x_i^{(l)}$ is the vector representation of node i in layer l, $x_j^{(l)}$ is the vector representation of neighboring node j of node i and $e_{ij}^{(l)}$ is the normalized weight between node i and j. Function $\varphi_{\text{agg}}(\cdot)$ aggregates feature information from neighboring nodes into the central node, weighted by edge features. Function $\varphi_{\text{upd}}(\cdot)$ updates the feature representation of each node at layer l into embeddings for every node at layer $l+1$ in the graph by using the vector representation of neighboring nodes. After applying multiple times these two functions, the embedding of ego-node at time t, denoted as $g_0^{(t)}$, is extracted from the graph and used for maneuver classification of the ego-vehicle at time t.

(2) **GNN–RNN architecture:** In complex traffic scenarios, it is important to make decisions based on driving actions of nearby traffic agents which are observed for some time. Preserving the temporal driving context is of profound importance. Therefore, compared to the previous architecture, the second proposed architecture, GNN–RNN architecture has an additional RNN module deployed after the GNN module, for tracking the temporal context of maneuver prediction. The GNN–RNN architecture is shown in Fig. 1, which includes the inner and middle parts of the architecture, colored in dark gray and gray, respectively.

Graph construction is done identically as in the previous model, described in Sec. 3.2, with one additional and important preprocessing step. Input graphs are ordered in time as a sequence of graphs $\{\mathcal{G}^{(t)} | t \in [t-\mathcal{T}, t]\}$ of a fixed time-window of length $\mathcal{T}$. This is a necessary step for using RNN module, where previous hidden representations are important for predicting the representation for next timestep. Such graphs are passed through the multi-layer GNN module first, as defined in Eq. (4), transforming node features into node embeddings. Ego-node embedding $g_0^{(t)}$ is extracted from graph $\mathcal{G}^{(t)}$ for each timestep $[t-\mathcal{T}:t]$, where a sequence of length $\mathcal{T}$ of ego-node embeddings is created $\{g_0^{(t)} | t \in [t-\mathcal{T}, t]\}$. This sequence $g_0^{(t-\mathcal{T}:t)}$ is passed as an input to the RNN module, tracking ego-embeddings in time, extracting temporal patterns of maneuver change based on the graph structure and features:

$$h^{(t-\mathcal{T}:t)} = \text{RNN}(g_0^{(t-\mathcal{T}:t)}), \tag{6}$$

where $\text{RNN}(\cdot)$ is a multi-layer RNN network, implemented as many-to-many, and $h^{(t-\mathcal{T}:t)}$ is a sequence of hidden representations for every timestep. Hidden representations $h^{(t-\mathcal{T}:t)}$ are passed through a multi-layer perceptron (MLP) to classify and predict the best ego-vehicle maneuver for every timestep $[t-\mathcal{T}:t]$:

$$z^{(t-\mathcal{T}:t)} = \text{MLP}(h^{(t-\mathcal{T}:t)}), \tag{7}$$

where $z^{(t)}$ is the final latent representation from the network. This representation is passed through softmax function for training and validating the network.

(3) **RNN–GNN–RNN architecture:** The last proposed architecture, RNN–GNN–RNN architecture is based on GNN module and RNN encoder–decoder framework [29], including all gray modules of the architecture in Fig. 1.
Extracted feature vectors from the detected traffic agents are used as inputs to the RNN encoder module, colored in light gray. Historical features $x_i^{(t)}$ are order in time, as defined in Eq. (2), from $t - \mathcal{T}$ to t, for every detected traffic agent i in the traffic scenario. This sequence of feature vectors $\{f_i^{(t)} | t \in [t - \mathcal{T}, t]\}$ serves as the input to the RNN encoder:

$$\hat{h}_i^{(t-\mathcal{T}:t)} = \mathrm{RNN}_{\mathrm{enc}}(f_i^{(t-\mathcal{T}:t)}), \tag{8}$$

where $\hat{h}_i^{(t-\mathcal{T}:t)}$ are the hidden representations of an individual traffic agent i from $t - \mathcal{T}$ to t, where $\mathcal{T}$ is a fixed time window. Hidden representation $\hat{h}_i^{(t)}$ from the $\mathrm{RNN}_{\mathrm{enc}}$ is used as a node feature for node i in graph $\mathcal{G}^{(t)}$, instead of the feature vector $f_i^{(t)}$ used in the two previous architectures. Graphs with new node features are passed through a multi-layer GNN module, as defined in Eq. (4) for message passing and node update. Ego-node embedding $g_0^{(t)}$ is extracted from the graph, encompassing temporal dependencies from individual traffic agents for $\mathcal{T}$ past timesteps and spatial features from the current timestep. The RNN decoder tracks dynamic features of ego-vehicle, such as movement changes:

$$h_0^{(t-\mathcal{T}:t)} = \mathrm{RNN}_{\mathrm{dec}}(g_0^{(t-\mathcal{T}:t)}). \tag{9}$$

The hidden representation $h_0^{(t)}$ from each timestep $[t - \mathcal{T} : t]$ is passed through an MLP network for classification, similar as defined in Eq. (7), mapping temporal features and spatial features of surrounding traffic agents. This representation is passed lastly through a softmax function, to train the deep learning network and predict the next best maneuver for ego-vehicle in different traffic scenarios.

4. Experiments

All experiments were conducted on a desktop with Ubuntu 18.04 with 2.2GHz Intel (R) Xeon(R) CPU E5-2698 v4, 256 GB RAM, and Tesla V100-DGXS-32GB for training.

4.1. *Dataset*

Proposed architectures are data-driven approaches for solving the problem of maneuver planning. Therefore, for these models to learn in a supervised fashion, a large amount of labeled data is required, with variety of complex traffic scenarios. In this work, Building a Large-Scale 5D Semantics Benchmark for Autonomous Driving (BLVD) [8] is used as the main dataset to train and validate the proposed architectures. BLVD provides 3D detection and tracking of heterogeneous traffic agents (i.e. vehicles, riders and pedestrians), the driving action of surrounding agents relative to the ego-vehicle (i.e. parallel driving in right/left, overtaking from right/left, straight accelerating/decelerating, stopping, etc.) and scenario description (i.e. highway or urban road, low or high density of participants or intersection), among others. But the main information provided is the driving maneuver of ego-vehicle, which is used as the label for supervised learning. Such maneuvers include "straight decelerating", "straight accelerating", "turning left", "turning right", "uniformly straight driving", "changing line to left", "changing line to right" and "stopping".

BLVD dataset, similar to most dataset for autonomous driving, suffers from the class imbalance problem, where the majority of datapoints are just driving straight or stopping, dominating the overall class distribution. This is a known issue in machine learning, therefore a further processing step was required to prepare the dataset. This step is necessary to stabilize the learning process and tackle the problem of class imbalance. This step includes adding datapoints in minority classes or removing datapoints from majority classes. In this work, data resampling techniques such as data augmentation, data oversampling and data undersampling techniques, were used to generate variants of BLVD dataset. But, since we are dealing here with graphs, these resampling techniques are important to evaluate the impact they have in the learning process. These three techniques are explained as follows:

(1) **Data augmentation** is a technique used extensively in computer vision, to increase the number of frame datapoints from existing frame datapoints. But, this process is different when working with graphs, especially in traffic scenarios. The only augmentation technique used here is a lateral inversion, in which traffic scenarios are flipped along the lateral axis. This means that the feature matrix of the graph had to be adjusted accordingly, such as the x, y coordinate position in image space, the heading angle, driving action "overtaking from left" would be "overtaking from right", "parallel driving in right" would be "parallel driving in left", and the maneuver of ego-vehicle "turning right" would be "turning left" or "changing line to left" would now be "changing line to right".
(2) **Data oversampling** is a technique used when there is a class imbalance in the dataset and consists of choosing randomly datapoints from the underrepresented classes of the dataset and adding them over in the training set. In our work, scenario graphs of the underrepresented classes are chosen, copying them over and adding noise in the node and edge features of the graph. Node feature include x, y position or length, width, height of the bounding box, which are chosen randomly, adding noise with a standard deviation of 0.01%, 0.02% or 0.03%. After this processing step, class distribution of the dataset is more balanced, and the added noise to node features and edge features makes the learning process more robust.

(3) **Data undersampling** is also a technique used in case there is a class imbalance problem in the dataset. The difference to the previous technique is that it removes datapoints from the overrepresented classes, without adding new datapoints from the underrepresented classes. BLVD dataset contains a lot of scenarios where the ego-vehicle is just driving straight or stopping, therefore these two classes are two main classes undersampled for experiments.

Using the aforementioned techniques, apart from the original BLVD dataset, six additional dataset variants are created, testing the impact of data augmentation, data oversampling and undersampling in the accuracy of all three proposed architectures. The first dataset variant is BLVD dataset [8], referred to as "BLVD" in the following, constructing the interaction graph as described in Sec. 3.2, with no additional datapoints or biases added or removed. The second variant is generated using data augmentation technique applied to all maneuver classes in BLVD, referred to as "B-AUG". Because of class imbalance in BLVD, data augmentation was applied to only underrepresented classes, such as "Turning Left", "Turning Right", "Changing Line to Left" and "Changing Line to Right", to generate the next variant called "B-AUG2". Next, BLVD dataset was oversampled with classes that are underrepresented, introducing randomly biases to the node or edge features in the process, generating the "B-DOS" variant. BLVD was undersampled with classes that are overrepresented, selecting randomly only 25% of "Uniform Straight Driving" class and 50% of "Stopping" class, generating the "B-DUS" dataset variant. Two techniques, data oversampling and undersampling were used to balance the class distribution and create the variant called "B-DOUS". Lastly, the variant referred to as "B-AUDO", was generated using first data augmentation to generate new possible datapoints, and then balancing the class distribution using data oversampling. The class distribution of these dataset variants is shown in Table 1.

Table 1. Class distribution of all seven dataset variants.

	BLVD	B-AUG	B-AUG2	B-DOS	B-DUS	B-DOUS	B-AUDO
Straight Decelerating	4.46%	3.88%	4.13%	14.27%	10.78%	12.93%	13.03%
Straight Accelerating	2.34%	2.36%	2.17%	13.26%	5.66%	12.84%	12.10%
Turning Left	3.53%	2.65%	4.84%	13.02%	8.53%	12.51%	12.00%
Turning Right	1.69%	1.65%	4.84%	11.26%	4.10%	12.56%	12.32%
Uniform Straight Driving	63.53%	67.14%	58.87%	15.64%	38.53%	15.17%	13.96%
Changing Line to Left	1.11%	1.23%	2.50%	7.40%	2.69%	8.26%	10.87%
Changing Line to Right	1.58%	1.30%	2.50%	10.49%	3.82%	11.71%	12.39%
Stopping	21.75%	19.79%	20.16%	14.65%	25.90%	14.02%	13.32%
Total	**69,040**	**102,877**	**74,504**	**280,481**	**28,562**	**214,172**	**314,118**

4.2. *Model implementation*

Models are implemented in Python, PyTorch [30] and Deep Graph Library (DGL) [31] for the GNN module. Three proposed architectures in this work can have one

GNN module or two modules in GNN and RNN, different number of layers, normalization, dropout values, hidden units or activation functions. But, just the most important hyperparameters and implementation details of each architecture, which showed the best results from the experiments, are explained as follows:

(1) **GNN architecture:** The input to the architecture is an interaction graph, with a feature matrix of dimension $b \times n \times d$, where b is the batch size, n represents the number of nodes and d is the dimensionality of the feature vector f_i. Batch size used for training and validating this architecture was 32. The best GNN module consists of three GNN layers with 32-dimensional hidden units. Each GNN layer, apart from the last layer, is followed by a layer normalization [32], activation function of LeakyReLU with a negative slope of 0.1, and dropout value of 0.1 to avoid overfitting. After three layers of GNN, the node embedding of ego-vehicle is extracted from the transformed graph. This node embedding from the last layer is converted in class probabilities using softmax function. The output dimension is eight, representing eight possible maneuvers the ego-vehicle can make, as noted in BLVD dataset. The architecture was trained for 150 epochs.

(2) **GNN–RNN architecture:** The input to the architecture is again an interaction graph, with a feature matrix of dimension $b \times n \times d$, with batch size 32. The interaction graph are ordered in time as a sequence, with a time widow of 20. The GNN module consists of three GNN layers with 16-dimensional hidden units. Each layer is passed through layer normalization, activation function of LeakyReLU with 0.1 negative slope, and no dropout. The ego-node embedding is extracted from the GNN module and used as an input to the RNN module. This module is used as the decoder, implemented using a two-layer LSTM network [33], with 16-dimensional hidden units, 0.2 dropout value and of type many-to-many. Lastly, an MLP is used as classification layer, with 16 hidden units and 0.2 dropout value. The final wights are passed through softmax function, outputting eight probabilities to classify the maneuver of ego-vehicle. This architecture converged faster, therefore was trained for 75 epochs.

(3) **RNN–GNN–RNN architecture:** This architecture deploys an LSTM encoder–decoder [29] for maneuver prediction. The input to the LSTM encoder is a feature matrix of size $b \times n \times d$, with batch size of 32. These extracted feature matrices are ordered in time, with a time window of 20. Encoder is implemented using a two-layer LSTM with 16-dimensional hidden units and a dropout value of 0.2. The hidden representation from every timestep is used as a node feature, while the structure of the graph is build from the input feature matrix. Such graphs serve as input to the GNN module, which consists of two GNN layers, with 16-dimensional hidden units, layer normalization, and LeakyReLU (0.1) activation function. The node embedding of ego-vehicle is extracted and given as an input to the LSTM decoder, implemented as a two-layer LSTM network, with 16-dimensional hidden units, 0.2 dropout value and many-to-many type. The

last layer of this architecture is an MLP classification layer, with 16-dimensional hidden units and a dropout value of 0.2. This architecture was trained for 50 epochs, but training time per epoch was longer then the previous architectures.

The dataset is split randomly in training set and testing set, with a ratio of 70% and 30%, respectively. Adam Optimizer [34] is used for training the proposed architectures, with a learning rate of 0.001 and L2 regularization of 0.0001. All models are trained as supervised classification problems, minimizing the cross-entropy loss between the predicted probabilities from the network and maneuver labels for ego-vehicle from the BLVD dataset:

$$\text{loss} = -\sum_{i}^{C=8} y_i \log(\text{softmax}(z_i)) \tag{10}$$

where z_i are the weight from the last layer of the network, passed through softmax function and y_i is the label from the dataset. These values are minimized over C classes, in this work eight maneuver classes for the ego-vehicles.

4.3. *Evaluation metrics*

Three proposed architectures are trained and evaluated using classification accuracy over eight maneuver labels, shown in Table 1. The problem of tactical maneuver prediction in this work is modeled as a multi-class classification problem. Since BLVD dataset is highly unbalanced, and since we experimented with different dataset variants, just accuracy and loss value is not enough to report the performance of the model. Apart from accuracy, F1 score and ROC score are also reported for all experiments to have a better understanding of the performance of proposed architectures.

5. Results

5.1. *Main results*

This section presents the main results from experiments, based on the implementation details provided in Sec. 4.2. The best accuracy, F1 score and ROC score of all three proposed architectures, for all dataset variants are shown in Table 2.

GNN architecture scored the best performance in B-DOS dataset variant, with 92.88% accuracy, 92.81% F1 score and 95.92% ROC score. Considering that time aspect was not part of the modeling, GNN architecture performed generally worse on the same dataset variants, compared to architectures with RNN module. GNN–RNN architecture has shown very good performance in B-DOS as well, with an accuracy value of 96.81%, F1 score of 96.79% and an ROC score of 98.15%. Lastly, RNN–GNN–RNN architecture has performed slightly better in the overall experiments. The best performance is reported again in the B-DOS dataset variant, with 97.01% accuracy, 97.01% F1 score and 98.27% ROC score. Figure 2 shows the training

Table 2. **Main Results:** Accuracy (Acc), F1 score (F1-S) and ROC score (ROC-S) are reported for all three proposed architectures. **Conclusion:** RNN–GNN–RNN architecture showed slightly better overall performance compared to the other two architectures, while all architectures saw the best improvement from B-DOS dataset variant.

Dataset	GNN			GNN–RNN			RNN–GNN–RNN		
variants	Acc	F1-S	ROC-S	Acc	F1-S	ROC-S	Acc	F1-S	ROC-S
BLVD	91.04%	90.88%	92.97%	92.61%	92.52%	93.93%	91.74%	91.50%	93.28%
B-AUG	*92.59%*	*92.27%*	*93.05%*	88.23%	87.74%	92.09%	87.34%	86.29%	91.46%
B-AUG2	91.07%	90.93%	92.98%	86.68%	86.19%	90.29%	86.06%	85.44%	90.11%
B-DOS	**92.88%**	**92.81%**	**95.92%**	**96.81%**	**96.79%**	**98.15%**	**97.01%**	**97.01%**	**98.27%**
B-DUS	86.58%	86.49%	91.32%	86.54%	86.42%	91.02%	84.70%	84.60%	89.99%
B-DOUS	92.76%	92.68%	95.85%	95.98%	95.96%	97.69%	96.49%	96.47%	97.98%
B-AUDO	90.96%	90.88%	94.83%	*94.74%*	*94.74%*	*96.96%*	*94.76%*	*94.76%*	*97.00%*

Notes: **Bold values** show the best result for an architecture, underline values are second best results and *italic values* are the third best overall results.

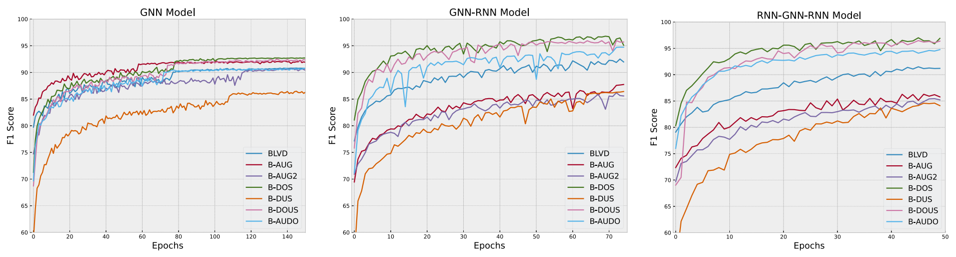


Fig. 2. Training graphs of F1 score evaluation metric for all dataset variants.

graphs of the F1 score for all three architectures, revealing that training in B-DOS dataset variant converged faster, compared to other dataset variants.

Results can also be looked from the perspective of BLVD dataset variants and the impact of data resampling techniques in the performance. Data oversampling had the biggest impact, improving the F1 score of the GNN architecture by 2.12%, GNN–RNN architecture by 4.62% and RNN–GNN–RNN architecture by 6.02%, compared with the original BLVD dataset. Dataset variant B-DOUS also improved the performance, since the dataset was increased with more traffic scenarios, added bias and noise, the learning architectures were able to differentiate from more movement patterns and features. While data undersampling technique in combination with data oversampling improved the results, data undersampling technique on the other hand did not improve the results. By removing datapoints from the overrepresented classes, B-DUS variant decreased the F1 score of GNN architecture by 4.83%, of GNN–RNN architecture 6.59% and RNN–GNN–RNN architecture by 7.54%. Also data augmentation did not improve the results generally. We argue that the reason that this technique did not improve the results is that by inverting a traffic scenario laterally, the model deals with a new scenario, hence a different interaction graph, with also different driving maneuvers and rules, often times generating scenarios which are not valid. This is especially relevant in architecture with RNN, where a movement patterns is tracked in time, but is not in compliance with traffic rules.

5.2. *Comparison results*

5.2.1. *Impact of maneuver class distribution*

As mentioned in Sec. 4.1 describing the BLVD dataset, there is a huge class imbalance, with "straight driving" and "stopping" classes that dominate the class distribution. Most of the time, in these traffic scenarios, not much happens and therefore data-driven models can't learn to differentiate the main features responsible for maneuver prediction. On the other hand, there are classes that are underrepresented such as turns or lane change maneuvers. But, what if we remove overrepresented or underrepresented classes from the dataset and train the architectures?

Experiments conducted in this subsection shows the performance results of GNN architecture in case traffic scenarios with certain ego-vehicle maneuvers are not used for training. Experiments are split in four groups of maneuvers:

- Maneuvers#1: All classes included from BLVD, eight classes in total;
- Maneuvers#2: Not included are overrepresented classes "uniform straight driving" and "stopping" in training, including all other classes, six classes in total;
- Maneuvers#3: Not included in training are underrepresented classes "changing line to left" and "changing line to right", six classes in total for training;
- Maneuvers#4: Not included are classes from the previous group, additionally maneuvers "turning left" and "turning right", four classes in total for training.

As shown in Fig. 3, removing maneuvers from training can impact the performance of the GNN architecture. Removing overrepresented maneuvers decreased the

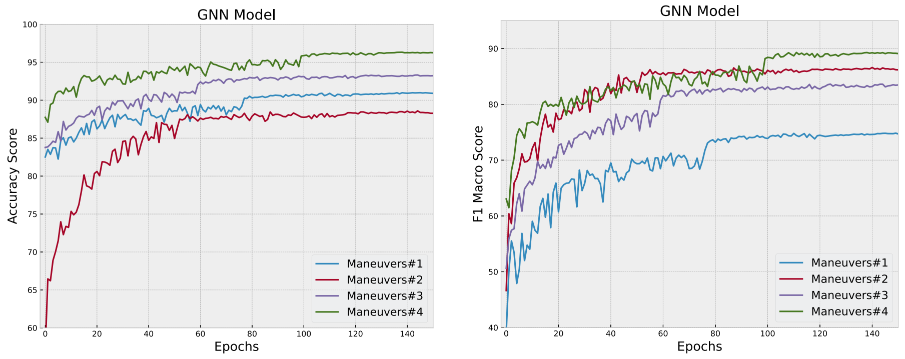


Fig. 3. Accuracy and F1 macro score graphs for the proposed GNN architecture, showing the impact of not including certain maneuvers in training and evaluation.

accuracy, confirmed also from the main results, where data undersampling showed the worst results. On the other hand, removing underrepresented classes increased the accuracy of training. Yet, such models thrive on larger datasets, hence we believe data-driven models are more accurate and complete if they utilize all datapoints, enlarged with data resampling techniques used in this work.

5.2.2. *Impact of GNN layers*

The GNN module is utilized in this work to extract spatial features from traffic graphs. Spatial features, together with temporal features have profound impact for decision-making. GNN architecture deals only with spatial features extraction, while GNN–RNN architecture combines spatial and temporal features extraction. Both architectures are implemented with three GNN layers, based on best empirical results from experiments. What if we change the number of GNN layers and what is the impact on spatial (GNN) or spatio-temporal (GNN–RNN) feature extraction?

In this subsection, we compared the performance of GNN architecture and GNN–RNN architecture if we change the number of GNN layers. Table 3 shows the impact of number of GNN layers on the best accuracy values, F1 score and ROC score.

Table 3. **Results:** Experiments ran on B-DOS dataset variant, reporting accuracy (Acc), F1 score (F1-S) and ROC score (ROC-S) for different number of GNN layers. **Conclusion:** Three-layer GNN module showed the best results for GNN architecture, while the performance change was minimal for GNN–RNN architecture.

No. of GNN layers	GNN			GNN–RNN		
	Acc	F1-S	ROC-S	Acc	F1-S	ROC-S
1	59.31%	59.47%	76.57%	96.58%	96.57%	98.03%
2	85.53%	85.45%	91.67%	96.65%	96.63%	98.05%
3	**92.88%**	**92.81%**	**95.92%**	96.81%	96.79%	98.15%
4	91.96%	91.87%	95.39%	**97.45%**	**97.45%**	**98.52%**
5	90.62%	90.50%	94.61%	97.32%	97.31%	98.45%

In the case of GNN architecture, the performance increases until the third GNN layer, at which point the performance decreases with each additional layer. We argue that, since we are dealing with small and complete graphs, three rounds of message passing are enough to capture spatial features from the scenario. Adding more layers can lead to feature over-smoothing, whereby the GNN model loses the expressive and discriminative power to learn from data.

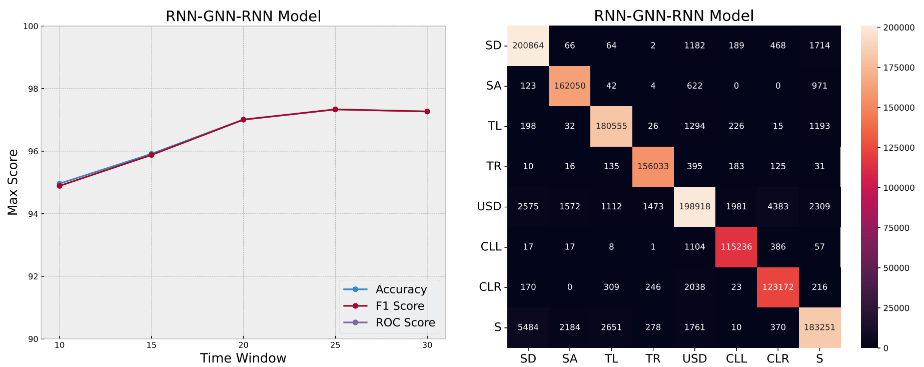
However, GNN–RNN architecture showed slightly better performance with four GNN layers, yet the overall difference is very small. We argue that extracted spatial features are not so dominant, compared with temporal features. Hence, in the GNN–RNN architecture, tracking temporal features in time with RNN is more important.

5.2.3. *Impact of RNN time-window*

For the main experiments, the value of time window for GNN–RNN and RNN–GNN–RNN architectures is empirically set to 20 timesteps, as the best trade-off value between prediction results and computation time. In this subsection, we experimented with the time-window in the RNN–GNN–RNN architecture and show the impact of this hyperparameter in the performance, again in the B-DOS variant.

Figure 4(a) visualizes the impact of time window in RNN encoder and decoder on the performance of the RNN–GNN–RNN architecture. Increasing the time window, improves the performance results. In short, more information from the past provides a better understanding of the traffic scenario, hence better prediction, until a threshold value is reached, after which the prediction does not improve much.

Figure 4(b) shows the confusion matrix of the RNN–GNN–RNN architecture, with a time window of 20 timesteps, trained on B-DOS variant. The confusion matrix reflects the good accuracy the model showed for maneuver prediction. Yet, it also shows the main classes the model struggles to predict and confuses with. Under-represented classes are confused and wrongly predicted for overrepresented classes of "uniform straight driving" and "stopping", maneuvers dominating the dataset.



(a) Performance impact of time window

(b) Confusion matrix for time window 20

Fig. 4. Time window analysis for RNN–GNN–RNN architecture.

5.3. *Visualization of maneuver prediction*

A typical traffic scenario, namely "M_L_m23_4_188" from BLVD dataset, is taken and visualized in Fig. 5 for model inference. The top part of the figure shows the camera image and extracted features used as input for the architecture, in this case RNN–GNN–RNN. On the bottom left, the interaction graph is constructed from extracted features, whereby nodes are colored based on the object class, red are "vehicles", blue are "riders" and green are "pedestrians". This graph is passed as an input to the model for inference, outputting a probability distribution over eight possible maneuvers for the ego-vehicle, shown on the bottom right part of the figure.

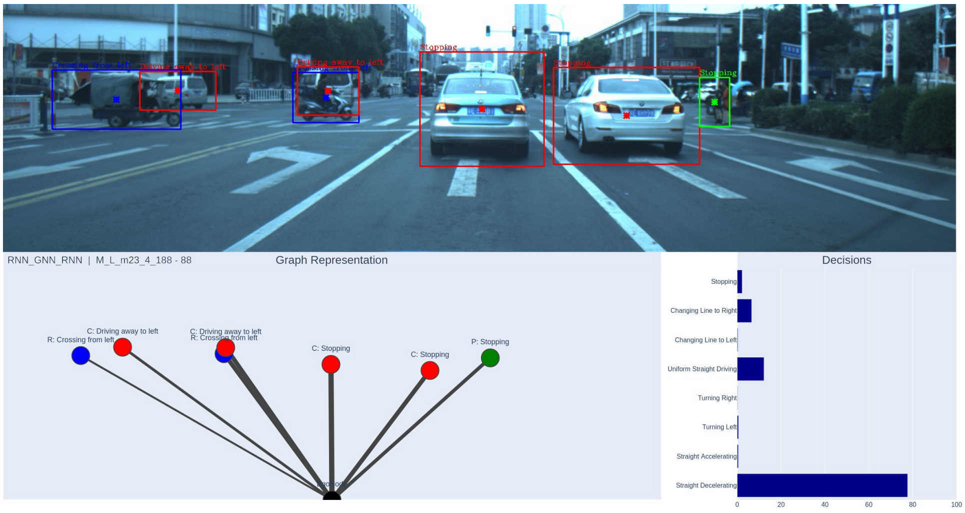


Fig. 5. Visualization of model inference for maneuver prediction.

In this urban traffic scenario, two cars and one pedestrian in front have stopped, while traffic participant are moving from left to right in an intersection. Based on extracted features, the model predicts the maneuver “straight decelerating” for the ego-vehicle.

6. Conclusion and Future Work

This work proposed modeling a traffic scenario as an interaction graph and proposed three deep learning architectures for interaction-aware tactical maneuver prediction based on GNNs and RNNs. The interaction graph is a flexible data structure, whereby nodes are modeled from detected traffic participants, while edges represent spatial interaction between them. Such graphs are utilized by GNNs to learn spatial and interactive behavior of surrounding traffic agents. On the other hand, graphs are order in time, modeling the problem of maneuver prediction as a sequence of graphs, utilizing RNNs for temporal feature extraction. BLVD was used as the main dataset, extended in form of dataset variants, with techniques of data augmentation, data oversampling and data undersampling. Three proposed architectures were trained and validated against all seven dataset variant, to stabilize the learning process and improve the robustness of models. The experiments showed different aspects of each architecture, but architectures which combined GNN and RNN showed superior results. Moreover, data manipulation techniques generally improved the results, especially data oversampling technique. Based on all experiments, we showed empirically that a traffic scenario can be modeled as a spatial interaction graph. The proposed data-driven architectures were validated in diverse driving scenarios for maneuver prediction in autonomous driving.

For future work, modeling the traffic graph with additional static features of the road network, such as lane markings, traffic signs or free space information, can improve the GNN module for spatial feature extraction. Moreover, including ego-vehicle odometry data must be part of input set too, as profound important information for maneuver planning. Lastly, experimenting with more data manipulation techniques, especially for enriching datapoints in form of interaction graphs, for tackling the unbalanced class distribution in autonomous driving datasets and improving deep learning motion prediction models.

Acknowledgment

This work is supported by the Federal Ministry for Digital and Transport (BMDV) of Germany within the scope of research project AORTA (FKZ: 01MM20002).

References

[1] S. Lefèvre, D. Vasquez and C. Laugier, A survey on motion prediction and risk assessment for intelligent vehicles, *ROBOMECH J.* **1** (2014) 1.

[2] T. Phan-Minh, E. C. Grigore, F. A. Boulton, O. Beijbom and E. M. Wolff, Covernet: Multimodal behavior prediction using trajectory sets, in *2020 IEEE/CVF Conf. Computer Vision and Pattern Recognition*, 2020, pp. 14062–14071.

[3] F. Diehl, T. Brunner, M. Truong-Le and A. C. Knoll, Graph neural networks for modelling traffic participant interaction, in *2019 IEEE Intelligent Vehicles Symposium*, 2019, pp. 695–701.

[4] X. Li, X. Ying and M. C. Chuah, GRIP: Graph-based interaction-aware trajectory prediction, in *2019 IEEE Intelligent Transportation Systems Conference*, 2019, pp. 3960–3966.

[5] X. Mo, Y. Xing and C. Lv, Graph and recurrent neural network-based vehicle trajectory prediction for highway driving, in *24th IEEE Int. Intelligent Transportation Systems Conference*, 2021, pp. 1934–1939.

[6] N. Deo and M. M. Trivedi, Convolutional social pooling for vehicle trajectory prediction, in *2018 IEEE Conf. Computer Vision and Pattern Recognition Workshops*, 2018, pp. 1468–1476.

[7] N. Deo and M. M. Trivedi, Multi-modal trajectory prediction of surrounding vehicles with maneuver based lstms, in *2018 IEEE Intelligent Vehicles Symposium*, 2018, pp. 1179–1184.

[8] J. Xue, J. Fang, T. Li, B. Zhang, P. Zhang, Z. Ye and J. Dou, BLVD: Building a large-scale 5D semantics benchmark for autonomous driving, in *Int. Conf. Robotics and Automation*, 2019, pp. 6685–6691.

[9] T. A. Ranney, Models of driving behavior: A review of their evolution, *Accid. Anal. Prev.* **26**(6) (1994) 733–750.

[10] Z. Li, C. Lu, Y. Yi and J. Gong, A hierarchical framework for interactive behaviour prediction of heterogeneous traffic participants based on graph neural network, *IEEE Trans. Intell. Transp. Syst.* **23**(7) (2021) 1–13.

[11] Z. Li, J. Gong, C. Lu and Y. Yi, Interactive behavior prediction for heterogeneous traffic participants in the urban road: A graph-neural-network-based multitask learning framework, *IEEE/ASME Trans. Mechatron.* **26**(3) (2021) 1339–1349.

[12] P. Rama and N. Bajcinca, NIAR: Interaction-aware maneuver prediction using graph neural networks and recurrent neural networks for autonomous driving, in *2022 Sixth IEEE Int. Conf. Robotic Computing*, 2022, pp. 368–375.

[13] C. Hermes, C. Wohler, K. Schenk and F. Kummert, Long-term vehicle motion prediction, in *2009 IEEE Intelligent Vehicles Symposium*, 2009, pp. 652–657.

[14] C. Laugier, I. E. Paromtchik, M. Perrollaz, Y. Mao, J. Yoder, C. Tay, K. Mekhnacha and A. Nègre, Probabilistic analysis of dynamic scenes and collision risks assessment to improve driving safety, *IEEE Intell. Transp. Syst. Mag.* **3**(4) (2011) 4–19.

[15] N. Deo, A. Rangesh and M. M. Trivedi, How would surround vehicles move? A unified framework for maneuver classification and motion prediction, *IEEE Trans. Intell. Veh.* **3**(2) (2018) 129–140.

[16] M. Liebner, M. Baumann, F. Klanner and C. Stiller, Driver intent inference at urban intersections using the intelligent driver model, in *2012 IEEE Intelligent Vehicles Symposium*, 2012, pp. 1162–1167.

[17] A. Houenou, P. Bonnifait, V. Cherfaoui and W. Yao, Vehicle trajectory prediction based on motion model and maneuver recognition, in *2013 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2013, pp. 4363–4369.

[18] M. Schreier, V. Willert and J. Adamy, Bayesian, maneuver-based, long-term trajectory prediction and criticality assessment for driver assistance systems, in *17th Int. IEEE Conf. Intelligent Transportation Systems*, 2014, pp. 334–341.

[19] X. Mo, Y. Xing and C. Lv, Interaction-aware trajectory prediction of connected vehicles using CNN-LSTM networks, in *The 46th Annual Conference of the IEEE Industrial Electronics Society*, 2020, pp. 5057–5062.

[20] J. Schlechtriemen, F. Wirthmueller, A. Wedel, G. Breuel and K. Kuhnert, When will it change the lane? A probabilistic regression approach for rarely occurring events, in *2015 IEEE Intelligent Vehicles Symposium*, 2015, pp. 1373–1379.

[21] A. Khosroshahi, Learning, Classification and Prediction of Maneuvers of Surround Vehicles at Intersections Using LSTMs, University of California, San Diego (2017).

[22] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner and G. Monfardini, The graph neural network model, *IEEE Trans. Neural Networks* **20**(1) (2009) 61–80.

[23] T. N. Kipf and M. Welling, Semi-supervised classification with graph convolutional networks, in *5th Int. Conf. Learning Representations, Conference Track Proceedings*, 2017.

[24] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals and G. E. Dahl, Neural message passing for quantum chemistry, in *Proc. 34th Int. Conf. Machine Learning*, eds. D. Precup and Y. W. Teh, Proceedings of Machine Learning Research, Vol. 70, 2017, pp. 1263–1272.

[25] Z. Cui, K. Henrickson, R. Ke and Y. Wang, Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting, *IEEE Trans. Intell. Transp. Syst.* **21**(11) (2020) 4883–4894.

[26] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li and C. Schmid, Vectornet: Encoding HD maps and agent dynamics from vectorized representation, in *2020 IEEE/CVF Conf. Computer Vision and Pattern Recognition*, 2020, pp. 11522–11530.

[27] R. Chandra, T. Guan, S. Panuganti, T. Mittal, U. Bhattacharya, A. Bera and D. Manocha, Forecasting trajectory and behavior of road-agents using spectral clustering in graph-LSTMs, *IEEE Rob. Autom. Lett.* **5**(3) (2020) 4882–4890.

[28] J. Pan, H. Sun, K. Xu, Y. Jiang, X. Xiao, J. Hu and J. Miao, Lane-attention: Predicting vehicles' moving trajectories by learning their attention over lanes, in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2021, pp. 7949–7956.

[29] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, Learning phrase representations using RNN encoder–decoder for statistical machine translation, in *Proc. 2014 Conf. Empirical Methods in Natural Language Processing*, a meeting of SIGDAT, a Special Interest Group of the ACL, eds. A. Moschitti, B. Pang and W. Daelemans, 2014, pp. 1724–1734.

[30] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, Pytorch: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems 32: Annual Conf. Neural Information Processing Systems 2019*, eds. H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox and R. Garnett, 2019, pp. 8024–8035.

[31] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu and Y. Gai, Deep graph library: A graph-centric, highly-performant package for graph neural networks, preprint (2019), arXiv:1909.01315.

[32] L. J. Ba, J. R. Kiros and G. E. Hinton, Layer normalization, *CoRR* abs/1607.06450 (2016), http://arxiv.org/abs/1607.06450.

[33] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Comput.* **9**(8) (1997) 1735–1780.

[34] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, in *3rd Int. Conf. Learning Representations*, 2015.