Geometric Deep Learning for Autonomous Driving: Unlocking the Power of Graph Neural Networks With CommonRoad-Geometric

Eivind Meyer, Maurice Brenner, Bowen Zhang, Max Schickert, Bilal Musani, and Matthias Althoff

Abstract— **Heterogeneous graphs offer powerful data representations for traffic, given their ability to model the complex interaction effects among a varying number of traffic participants and the underlying road infrastructure. With the recent advent of graph neural networks (GNNs) as the accompanying deep learning framework, the graph structure can be efficiently leveraged for various machine learning applications such as trajectory prediction. As a first of its kind, our proposed Python framework offers an easy-to-use and fully customizable data processing pipeline to extract standardized graph datasets from traffic scenarios. Providing a platform for GNN-based autonomous driving research, it improves comparability between approaches and allows researchers to focus on model implementation instead of dataset curation.**

I. Introduction

Machine learning agents require an accurate understanding of the surrounding traffic context to make safe and effective decisions [1]. This calls for a descriptive representation not only of the various entities within the traffic environment, but also their complex spatial and temporal relationships. When restricted to a fixed-size feature space—a prerequisite for classical neural networks—this is a particularly challenging prospect given the inherent complexity and variability in road geometries and traffic situations.

An alternative approach is to model the environment state in the form of a heterogeneous graph, encompassing both the road network topology and the traffic participants present in it. This structured representation allows us to capture a wide range of road networks and traffic scenarios with a variable number of discrete elements. Additionally, it enables the explicit modeling of pairwise relationships or interactions between entities through edge connections. Graph neural networks (GNNs) have recently emerged as the principal deep learning framework for processing graph data [2]–[5]. By using GNNs, the graph topology can be exploited as a relational inductive bias [5] during training, aiding generalization.

A. Related work

Graph-structured representations of traffic proposed in existing works differ based on the objectives of the learning task. In [6], the authors let nodes represent individual road segments with edges forming the overarching road network topology for driving speed estimation and road network classification. Alternatively, [7] and [8] present graph-based traffic forecasting approaches in which nodes are used to model traffic participants and edges are used to capture vehicle interactions. Similarly, [9] and [10] adopt vehicle-to-vehicle GNNs as policy networks for reinforcement learning agents. Additionally considering the temporal dimension, [11] employs a spatiotemporal vehicle graph for capturing time-dependent features. Finally, recent works [12]–[20] incorporate both vehicle and map nodes by modelling the environment as a heterogeneous graph, including both inter-vehicle as well as vehicle-road interaction effects.

B. Motivation and contributions

Despite the vast research interest, there is no software framework that offers an interface for extracting custom graph datasets from traffic scenarios. Although there are plenty of autonomous driving datasets, e.g. [21]–[23], researchers have to write considerable amounts of ad-hoc, error-prone conversion code to use them as inputs for GNN models. As evident by the success in other application domains such as bioinformatics and social networks [24], [25], standardized graph datasets would enable autonomous driving researchers to streamline their experiments and ensure comparability and repeatability of their results.

To fill this gap, we propose *CommonRoad-Geometric* (*cr-geo*): a Python library designed to facilitate the extraction of graph data from recorded or simulated traffic scenarios. Our framework extends the *CommonRoad* [26] software platform and uses its standardized interface for a high-level representation of the traffic environment. As illustrated by Fig. 1, our framework unifies the traffic scene into a single graph entity, encompassing both the traffic participants and the underlying road map. The extracted graph representations are based on the `HeteroData` class offered by *PyTorch-Geometric* [27], a popular *PyTorch* [28] extension for deep learning on graph-structured data.

Our paper offers the following contributions:

- we introduce a heterogeneous graph structure for map-aware traffic representations tailored to GNN applications;
- we present and outline the software architecture of *CommonRoad-Geometric* (*cr-geo*), which offers a bridge from the well-established CommonRoad scenario format to *PyTorch-Geometric* (*PyG*);
- as a concrete example of the wide range of GNN-based applications facilitated by *cr-geo*, we train a spatiotemporal trajectory prediction model on a real-world graph dataset extracted from NuPlan [23].

Department of Informatics, Technical University of Munich, Garching, Germany. {eivind.meyer,maurice.brenner,bowen.zhang,max.schickert,bilal.musani,althoff}@tum.de

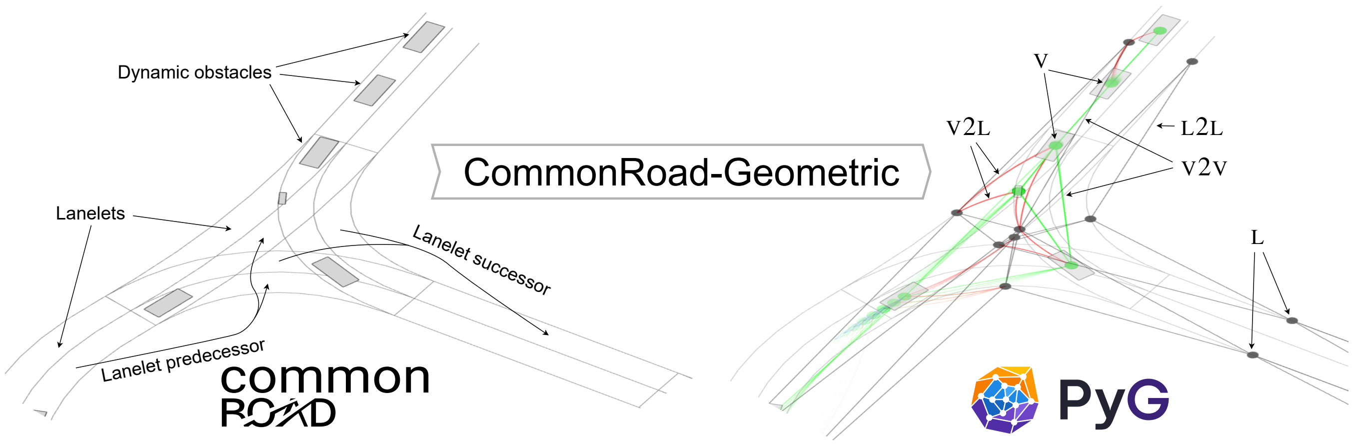


Fig. 1: Our package serves as a bridge from CommonRoad to PyTorch-Geometric. The abbreviated graph labels refer to the heterogeneous node and edge entities of our unified traffic graph structure covered in Section III-A.

II. Background

We first provide some background on heterogeneous graphs and the description of scenarios in CommonRoad.

1) Heterogeneous graphs: A directed heterogeneous graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A}, \mathcal{R}, \mathcal{X}_\mathcal{V}, \mathcal{X}_\mathcal{E})$ is defined as a tuple of a set of nodes $\mathcal{V}$, edges $\mathcal{E}$, node types $\mathcal{A}$, edge types $\mathcal{R}$, and corresponding node and edge features $\mathcal{X}_\mathcal{V}$ and $\mathcal{X}_\mathcal{E}$ [29]. Edges are defined as a 3-tuple $\mathcal{E} \subset \mathcal{V} \times \mathcal{R} \times \mathcal{V}$. Further, each node $v \in \mathcal{V}$ is assigned a node type via the mapping $\tau_V(v) : \mathcal{V} \rightarrow \mathcal{A}$. Analogously, edges $e \in \mathcal{E}$ are associated with an edge type $\tau_\mathcal{E}(e) : \mathcal{E} \rightarrow \mathcal{R}$. Finally, $\mathcal{X}_\mathcal{V}$ and $\mathcal{X}_\mathcal{E}$ contain the feature vectors, which we denote as $\mathbf{x}_v \in \mathbb{R}^{D_\mathcal{V}}$ for node features and $\mathbf{x}_e \in \mathbb{R}^{D_\mathcal{E}}$ for edge features. Fig. 2 illustrates graph features on the node and edge level.

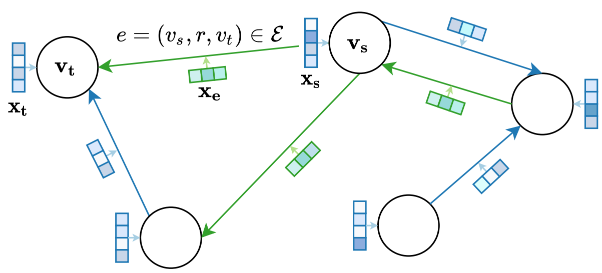


Fig. 2: Illustration of a simple graph structure. Edge types are represented as different colors. Colored squares represent features vectors for nodes (e.g., vehicle speed) and edges (e.g., distance between vehicles).

2) CommonRoad: A CommonRoad *scenario* contains a set $\mathcal{V}^{cr}$ of *dynamic obstacles*. For a vehicle $V \in \mathcal{V}^{cr}$ of rectangular shape (l_V, w_V), we represent its time-dependent state by its x-y center position $\mathbf{p}_V$, its orientation θ_V, as well as their time derivatives in the scenario coordinate frame.

Further, the map-related information of a scenario is described by its *lanelet network*, containing a set $\mathcal{L}^{cr}$ of atomic *lanelets* and their longitudinal and lateral adjacency relations [30]. Lanelets are geometrically defined by their boundary polylines. For a lanelet $L \in \mathcal{L}^{cr}$, we denote its left and right boundary polylines as $S_{L,l}$ and $S_{L,r}$, respectively. Further, we denote the center polyline as $S_{L,c}$. Each polyline is made up of a sequence of N_L waypoints in the scenario coordinate frame. Using the notation $\langle \square_i \rangle_{i \in 1,\ldots,N}$ to define a sequence of values of length N, we let the polylines be denoted as $S_{L,l} = \langle \vec{s}_{l,i} \rangle_{i \in 1,\ldots,N_L}$, $S_{L,r} = \langle \vec{s}_{r,i} \rangle_{i \in 1,\ldots,N_L}$, and $S_{L,c} = \langle \vec{s}_{c,i} \rangle_{i \in 1,\ldots,N_L}$. With $\| \|_2$ denoting the L2-norm, we further define the lanelet length as

$$\|L\|_2 = \sum_{i=1}^{N_L - 1} \sqrt{\|\vec{s}_{c,i+1} - \vec{s}_{c,i}\|_2}, \tag{1}$$

III. Overview

In the following subsections, we define the structure of our traffic graph and outline the software architecture of our framework.

A. Traffic graph structure

Extending *PyG*'s `HeteroData`[1], *cr-geo*'s `CommonRoadData` class represents a heterogeneous traffic graph encapsulating nodes of both the vehicle (V) and lanelet (L) node type, as well as the structural metadata for the specific graph instance. Formally, we have that $\mathcal{A} = \{\text{V}, \text{L}\}$ and $\mathcal{R} = \{\text{L2L}, \text{V2V}, \text{V2L}, \text{L2V}\}$.

In order to capture temporal vehicle interactions with GNN-based message-passing schemes, we further extend the `CommonRoadData` class by the time dimension with `CommonRoadTemporalData`, where $\mathcal{R}$ is augmented by the temporal VTV edge type. The resulting *temporal graph* [31], as shown in Fig. 3, intrinsically encodes the temporal dimension by unrolling the traffic graph over time. Vehicle nodes are repeated to capture vehicle states at past timesteps, and temporal edges encode the time difference between them.

[1]Our source code is available at `https://github.com/CommonRoad/crgeo`, and is provided under the BSD-3-Clause license, allowing free use and distribution.

[1]`https://pytorch-geometric.readthedocs.io/en/latest/modules/data.html#torch_geometric.data.HeteroData`

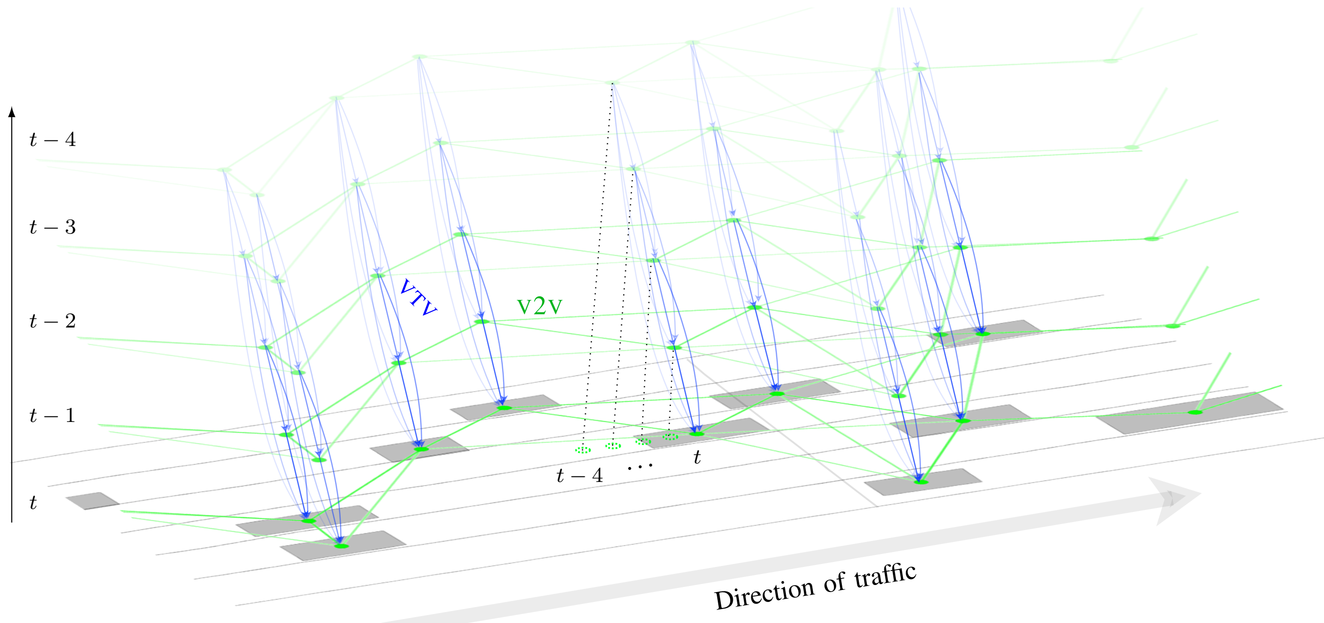


Fig. 3: Visualization of Voronoi-based V2V and causal VTV edges for a highway scenario. Opaque colors denote the most recent edges while more transparent shades represent older ones.

The heterogeneous structure of our graphical data representation is summarized in Tab. II and outlined in the following paragraphs:

1) Lanelet nodes ($\mathcal{V}_L$)*:* Lanelet nodes map to the lanelets in $\mathcal{L}^{cr}$, with each lanelet L being represented as a graph node. The corresponding node features encode the geometric properties of the respective lanelets. Using the general notation ${}^{L}\square$, we denote the lanelet-local transformed polyline coordinates as ${}^{L}S_{L,l}$, ${}^{L}S_{L,c}$, and ${}^{L}S_{L,r}$. Here, the vertex coordinates are transformed to the lanelet-local coordinate frame according to its origin position $\mathbf{p}_L = \vec{s}_{c,1}$ and its orientation $\theta_L = \operatorname{atan2}\left(\vec{s}_{c,2}/\vec{s}_{c,1}\right)$. In the following, we also let the function interpretations $\tilde{S}_{L,\square}(\cdot)$ and $\tilde{\theta}_L(\cdot)$ be defined via orthogonal projections onto the polylines, returning the position and orientation at a given arclength, respectively.

2) Lanelet-to-lanelet edges ($\mathcal{E}_{\text{L2L}}$)*:* Lanelet-to-lanelet edges characterize the lanelet network topology and encode the spatial relationship between adjacent lanelets. As illustrated by Fig. 4, we explicitly differentiate between the heterogeneous L2L adjacency types $\mathcal{R}_{\text{L2L}}$ listed in Tab. I via the edge feature $\tau_{\mathcal{E}}$. For an edge from L to L', we also include their centerline arclength distance at the point of intersection as edge features, which we denote by s_L and $s_{L'}$. This is highlighted in Fig. 4c for two conflicting lanelets.

3) Vehicle nodes ($\mathcal{V}_V$)*:* Vehicle nodes are inserted according to the states of the currently present vehicles and identified by their CommonRoad IDs. As shown in Fig. 3, `CommonRoadTemporalData` additionally includes past vehicle states in the graph representation: here, a vehicle's state history is captured by time-attributed (but otherwise identical) vehicles nodes inserted at each timestep.

4) Vehicle-to-vehicle edges ($\mathcal{E}_{\text{V2V}}$)*:* Vehicle-to-vehicle edges capture the interaction between vehicles at each timestep. The relative pose of connected vehicles is encoded

TABLE I: L2L adjacency types for edges $L \to L'$. The considered adjacency types can be individually selected at the user's preference.

Type	**Interpretation**
predecessor	L continues the driving corridor of L'
successor	L' continues the driving corridor of L
adjacent left	L is left-adjacent to L'
adjacent right	L is right-adjacent to L'
merging	L and L' share a common successor (symmetric)
diverging	L and L' share a common predecessor (symmetric)
conflicting	L and L' cross each other (symmetric)

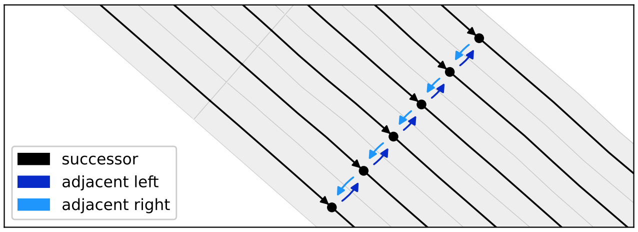
as edge features according to the vehicle-local coordinate frame originating at the center of the source vehicle. Users can provide a specific implementation of our V2V *edge drawer* protocol to encode which V2V relations are relevant for their task. Our framework offers developers a set of standard edge drawer implementations, two of which are depicted in Fig. 5.

5) Vehicle-to-lanelet edges ($\mathcal{E}_{\text{V2L}}$)*:* Vehicle-to-lanelet edges relate vehicles to the underlying road infrastructure. Our framework offers two assignment strategies for drawing the edges:

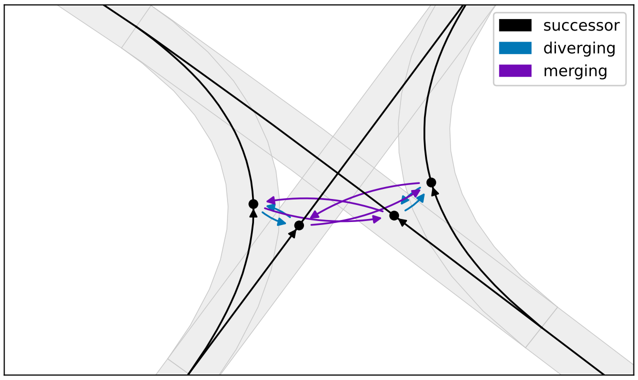
1) **Center**: Each vehicle is connected to all lanelets that contain the vehicle center point.
2) **Shape**: Each vehicle is connected to all lanelets that intersect with the vehicle shape. This constitutes a superset of the edges drawn by the *Center* strategy.

The associated edge features describe the relative pose of the vehicle with respect to the curvilinear lanelet coordinate frame [32]. For a given edge from vehicle V to lanelet L, we denote the orthogonal distance from the lanelet's left and right boundaries to the vehicle center as

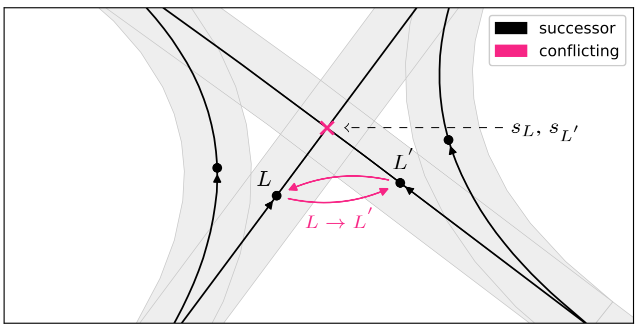
$$\begin{aligned} d^L_{V,l} &= \|\tilde{S}_{L,l}(\mathbf{p}_V) - \mathbf{p}_V\|_2, \\ d^L_{V,r} &= \|\tilde{S}_{L,r}(\mathbf{p}_V) - \mathbf{p}_V\|_2. \end{aligned} \tag{2}$$



(a) Highway scenario containing five parallel lanelets.



(b) Intersection scenario highlighting merging and diverging lanelets.



(c) Same road network as in (b), this time highlighting the conflicting lanelet nodes L and $L^{'}$. The point of intersection is marked with a cross.

Fig. 4: Lanelet graphs highlighting L2L adjacency types.

Further, we define the signed offset from the centerline as

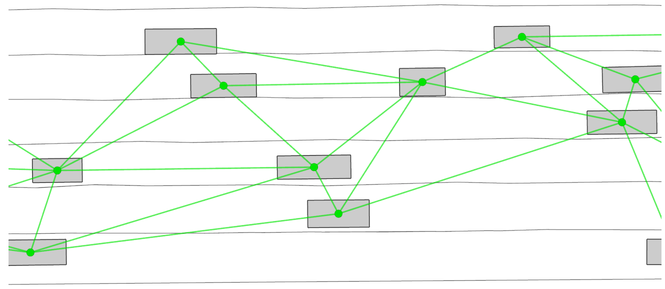
$$d_{V,e}^{L} = \frac{d_{V,l}^{L} - d_{V,r}^{L}}{2} \tag{3}$$

and let $s_V^L \in [0, \|L\|_2]$ denote the centerline arclength from the lanelet origin $\mathbf{p}_L$ to $\tilde{S}_{L,c}(\mathbf{p}_V)$. Finally, the orientation difference is given by

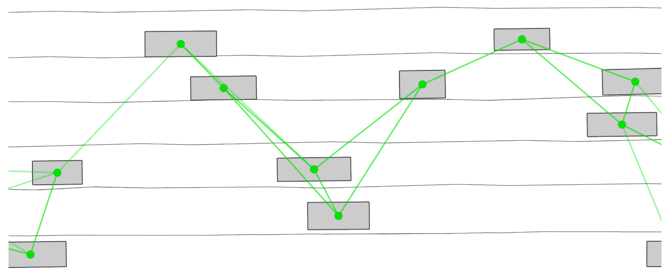
$$\theta_{V,e}^{L} = \tilde{\theta}_L(\mathbf{p}_V) - \theta_V. \tag{4}$$

6) Vehicle-temporal-vehicle edges ($\mathcal{E}_{\text{VTV}}$): `CommonRoadTemporalData` additionally contains temporal vehicle edges for encoding temporal dependencies in the traffic graph: Letting v_t and $v_{t'}$ denote two vehicle nodes emerging from one vehicle instance, a temporal edge $e \in \mathcal{E}_{\text{VTV}}$ connects the vehicle object to itself at two different timesteps, as illustrated by Fig. 3. The temporal separation between the nodes is defined as the elapsed time $\Delta t_e = t - t^{'}$.

As for V2V, *cr-geo* lets users specify a custom *temporal edge drawer* for defining the exact VTV graph structure. The default `CausalEdgeDrawer` inserts directed temporal edges between a historic vehicle node and its future realization at up to T_{max}^{VTV} future time steps. This constrains the flow of the VTV edges to be forward in time.



(a) *VoronoiEdgeDrawer* inserts edges according to a Delaunay triangulation.



(b) *KNearestEdgeDrawer* connects the k closest vehicles (here with $k = 3$).

Fig. 5: V2V edges drawn by two different implementations.

B. Software architecture

Next, we outline the principal components of our software architecture in a bottom-up approach, by detailing the pipeline for collecting graph datasets from CommonRoad scenarios. An architecture overview is given by Fig. 6.

1) Design principles: Conforming to the *single responsibility principle* [34], *cr-geo* is composed of distinct modules with clearly defined roles. To encapsulate volatile processing routines that may require frequent modifications, the *strategy pattern* [35] is employed. This lets the user realize specific behaviors through composition, making it possible to extend the framework to satisfy different requirements and use cases. Users can introduce or substitute implementations without interfering with the rest of the framework, avoiding unintended side effects and minimizing debugging, testing, maintenance, and refactoring efforts.

Next, we introduce the core functionalities of our framework on the basis of these design principles. Tab. III summarizes the options available to customize graph extraction:

TABLE III: Summary of *cr-geo*'s base protocols facilitating user-specified behaviors.

Component	Target	Count	Default
Preprocessors	input scenario	*many*	$\emptyset$
Filters	input scenario	*many*	$\emptyset$
V2V *edge drawer*	$\mathcal{E}_{\text{V2V}}$	*one*	`VoronoiEdgeDrawer`
VTV *edge drawer*	$\mathcal{E}_{\text{VTV}}$	*one*	`CausalEdgeDrawer`
Feature extractors	$\mathcal{X}_{\mathcal{V}}$, $\mathcal{X}_{\mathcal{E}}$	*many*	$\emptyset$
Postprocessors	$\mathcal{G}$	*many*	$\emptyset$

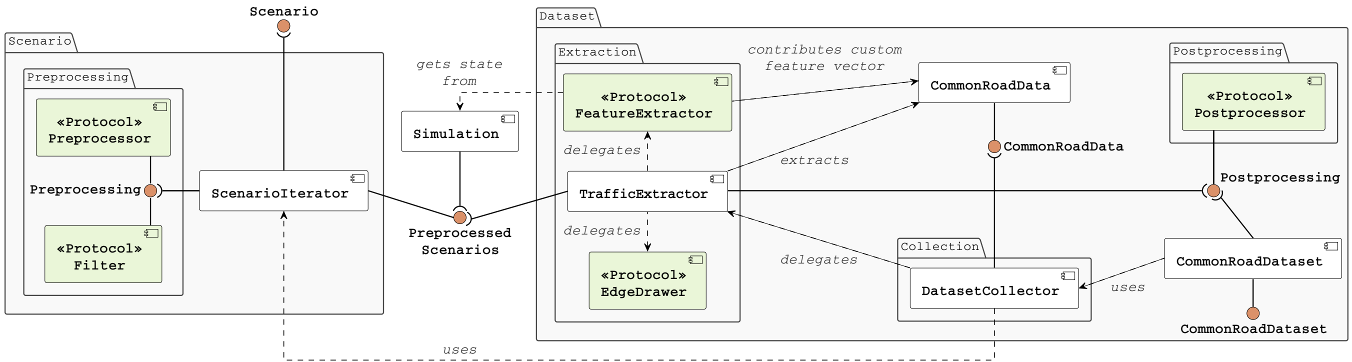


Fig. 6: High-level software architecture for *cr-geo* shown as a UML 2 component diagram [33].

2) Scenario preprocessing and filtering: Our framework supports arbitrary preprocessing and filtering of the input scenarios to ensure that they satisfy the user requirements. As an example, `TrafficFilter(min=10)` lets users exclude scenarios with less than 10 vehicles from the collected graph dataset. Further, suppose that we want a higher-fidelity view of the lanelet network, where no lanelet exceeds 20 meters in length: this can be achieved by using *cr-geo*'s built-in `SegmentLanelets` *preprocessor*, which results in a higher lanelet node density. Using our composition syntax that realizes arbitrary chaining and grouping of preprocessing and filtering operations, these behaviors can be effortlessly combined via
`TrafficFilter(min=10) » SegmentLanelets(size=20)`.

3) Feature extraction: The extraction of the graph features, i.e., $\mathcal{X}_\mathcal{V}$ and $\mathcal{X}_\mathcal{E}$, is carried out through *feature extractors* operating on the scenario objects. The feature extractors receive a *simulation* object, which provides access to inherent and derived state information of the scenario at the current timestep. By maintaining an internal state, feature extractors also support time-dependent features.

In our framework, we distinguish between node- and edge-level feature extractors, with the latter operating on pairs of entities according to $\mathcal{E}$. In accordance with *cr-geo*'s guiding design principles, users can devise their own implementation to augment the extracted graphs by arbitrary custom features without modifying the *cr-geo* source code. To reduce the initial setup time for new users, *cr-geo* also offers users a selection of pre-configured feature extractor implementations designed to meet the most common needs and requirements.

4) Postprocessors: In contrast to the aforementioned node- and edge-level feature extractors, *postprocessors* operate directly on the graph instances in a deferred manner. As such, they are intended to supplement the flexibility we concede in the otherwise rigorous graph extraction procedure. User-devised postprocessing procedures can serve numerous practical purposes, e.g.,

- to complement the functionality of feature extractors by facilitating the computation of graph-global features, e.g., an indicator for whether the current traffic scene corresponds to a traffic jam or not;
- to modify the graph structure, e.g., by the removal or insertion of nodes and edges.

The postprocessors are executed after the initial graph extraction, but prior to dataset generation. Furthermore, they can also be applied while loading an existing dataset.

5) Traffic graph extraction: The execution of edge drawing, feature extraction and postprocessing is orchestrated by a *traffic extractor*, which manages the creation of a `CommonRoadData` instance at a single timestep t. It also handles the declaration of metadata attributes such as node IDs.

The traffic extractor class is complemented by the *temporal traffic extractor* for the purpose of extracting temporal graph representations. Internally relying on a regular traffic extractor, it maintains a cache of the n preceding ordinary `CommonRoadData` instances at all times. When called upon, it returns the `CommonRoadTemporalData` at the current timestep by merging the cached graph sequence into a single entity. The temporal extractor delegates the temporal edge construction to the provided *temporal edge drawer* and the computation of custom temporal features to the VTV feature extractors provided by the user. The regular and temporal extraction procedures are summarized by Alg. 1 and Alg. 2, respectively.

6) Dataset creation: Based on the provided traffic graph extractor, the *dataset collector* offers an interface for generating a chronological sequence of graph instances from a specified scenario. The collector iterates over consecutive timesteps until it reaches the end of the scenario lifetime, dynamically returning the extracted graph objects. Whereas the static simulation mode corresponds to a replay of the recorded vehicle trajectories contained within the CommonRoad scenario, the interactive mode leverages an interactive traffic simulator for on-the-fly generation of realistic vehicle behavior. One such implementation is provided by the traffic simulation tool SUMO [36], which we access via our CommonRoad interface [37].

Finally, the creation and accessing of a persistent graph dataset collected from a set of input scenarios is facilitated by the `CommonRoadDataset`, a full-fledged extension of the powerful `Dataset`[2] class natively offered by *PyG*. As such,

[2] `https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.data.Dataset.html`

TABLE II: Overview of node ($\mathcal{X}_\mathcal{V}$) and edge ($\mathcal{X}_\mathcal{E}$) features.

Type	Feature	Notation	Unit	Size
L (L)	Position	$\mathbf{p}_L$	m	2
	Length	$\lVert L \rVert_2$	m	1
	Orientation	θ_L	rad	1
	Left vertices	${}^L V_l$	m	$2 \cdot N_L$
	Right vertices	${}^L V_r$	m	$2 \cdot N_L$
	Custom feature vector	$\mathbf{x}_v^{(\mathrm{L})}$	—	
V (V)	Position	$\mathbf{p}_V$	m	2
	Orientation	θ_V	rad	1
	Yaw-rate	$\dot{\theta}_V$	rad/s	1
	Velocity	$\dot{\mathbf{p}}_V$	m/s	2
	Acceleration	$\ddot{\mathbf{p}}_V$	m/s^2	2
	Vehicle width	w_V	m	1
	Vehicle length	l_V	m	1
	Custom feature vector	$\mathbf{x}_v^{(\mathrm{V})}$	—	
L2L ($L \to L'$)	Distance[a]	$\lVert \mathbf{p}_{L'} - \mathbf{p}_L \rVert_2$	m	1
	Relative position[a]	${}^L \mathbf{p}_{L'}$	m	2
	Relative orientation[a]	$\theta_{L'} - \theta_L$	rad	1
	Intersection (source)	s_L	m	1
	Intersection (target)	$s_{L'}$	m	1
	Adjacency type	$\tau_\mathcal{E}$	—	1
	Custom feature vector	$\mathbf{x}_e^{(\mathrm{L2L})}$	—	
V2V ($V \to V'$)	Distance	$\lVert \mathbf{p}_{V'} - \mathbf{p}_V \rVert_2$	m	1
	Relative position	${}^V \mathbf{p}_{V'}$	m	2
	Relative orientation	$\theta_{V'} - \theta_V$	rad	1
	Relative velocity	${}^V \dot{\mathbf{p}}_{V'} - {}^V \dot{\mathbf{p}}_V$	m/s	2
	Relative acceleration	${}^V \ddot{\mathbf{p}}_{V'} - {}^V \ddot{\mathbf{p}}_V$	m/s^2	2
	Custom feature vector	$\mathbf{x}_e^{(\mathrm{V2V})}$	—	
VTV[b]	*v2v features*	—— " ——	– " –	– " –
	Time delta	Δt_e	s	1
	Custom feature vector	$\mathbf{x}_e^{(\mathrm{VTV})}$	—	
V2L ($V \to L$)	Left distance	$d_{V,l}^L$	m	1
	Right distance	$d_{V,r}^L$	m	1
	Lateral offset	$d_{V,e}^L$	m	1
	Heading error	$\theta_{V,e}^L$	rad	1
	Projected arclength	s_V^L	m	1
	Normalized arclength	$s_V^L / \lVert L \rVert_2$	—	1
	Custom feature vector	$\mathbf{x}_e^{(\mathrm{V2L})}$	—	

[a]measured between the respective lanelet coordinate frames.
[b]only relevant for *CommonRoadTemporalData*.

our dataset class allows users to easily perform common data operations such as batching, sampling, and parallelization on the collected dataset during model training. As the individual graph instances inherit from the base data representation used by *PyG*, *cr-geo* practitioners can effortlessly adopt their wide selection[3] of state-of-the-art GNN architectures. The dataset creation procedure is summarized by Alg. 3.

[3]`https://pytorch-geometric.readthedocs.io/en/latest/modules/`

Algorithm 1 Extraction of *CommonRoadData* by *TrafficExtractor*.

```
Configuration:
    Scenario scenario
    Edge drawer D^V2V
    Set of feature extractors F
    Sequence of postprocessors P_post
Initialization:
    simulation ← Simulation(scenario)
Input:
    Timestep t
Output:
    CommonRoadData
Procedure: EXTRACT
    state ← simulation(t)                         ▷ Current traffic state
    v2v_edges ← D^V2V(state)
    features ← ∅        ▷ Container for V2V, L2L, V2L, and L2V features
    for each f ∈ F do                              } Feature extraction
        feature ← f(state, v2v_edges)
        features ← features ∪ feature
    data ← CommonRoadData(state, v2v_edges, features)
    for each p ∈ P_post do                         } Postprocessing
        data ← p(data)
    return data
```

Algorithm 2 Extraction of *CommonRoadTemporalData*.

```
Configuration:
    Data cache size n
    Single timestep traffic extractor E_regular        ▷ Ref. Alg. 1
    Temporal edge drawer D^VTV
    Set of temporal feature extractors F^VTV
    Sequence of temporal data postprocessors P^VTV_post
Initialization:
    Empty data sample cache C ← LIST(max_size=n)
Input:
    Timestep t
Output:
    CommonRoadTemporalData
Procedure: EXTRACT
    data ← E_regular(t)
    C.put(data)
    vtv_edges ← D^VTV(C)
    vtv_features ← ∅                       ▷ Container for VTV features
    for each f ∈ F^VTV do                          } Feature extraction
        feature ← f(C, vtv_edges)
        vtv_features ← vtv_features ∪ feature
    t_data ← CommonRoadTemporalData(C, vtv_edges, vtv_features)
    for each p ∈ P^VTV_post do                     } Postprocessing
        t_data ← p(t_data)
    return t_data^t
```

Algorithm 3 Creation of *CommonRoadDataset* from *CommonRoadData*.

```
Configuration:
    Composed preprocessor P_pre       ▷ Chain of preprocessors and filters
    Edge drawers D   ▷ Contains D^V2V and (for temporal datasets) D^VTV
    Set of feature extractors F
    Sequence of postprocessors P_post
Input:
    Set of scenarios S                    ▷ CommonRoad scenario directory
    Timesteps T per scenario
Output:
    CommonRoadDataset
Procedure: CREATE
    dataset ← CommonRoadDataset()
    for each scenario ∈ ScenarioIterator(S, P_pre) do
        simulation ← Simulation(scenario)                 } Delegated to
        E ← Extractor(simulation, D, F, P_post)           } DatasetCollector
        samples ← ∅
        for t ← 1, 2, ..., T do
            data ← E(t)
            samples ← samples ∪ data
        dataset ← dataset ∪ samples
    return dataset
```

IV. Dataset and Experiment

To demonstrate the usage of our framework, we published[4] a graph-converted dataset with a diverse set of real-world road geometries. With the help of the dataset converter[5], we use the NuPlan [23] dataset as our data source, due to the diversity in the incorporated locations and environments. The extracted graph dataset contains *cr-geo*'s default graph features as previously listed in Tab. II.

A. Experiment

As an example use case of our framework, we briefly[6] introduce a spatiotemporal GNN model for vehicle trajectory prediction based on our `CommonRoadTemporalData` environment representation. We implement an end-to-end trainable encoder-decoder architecture based on the message passing framework offered by *PyG*:

1) Encoder component: The GNN encoder, for which we use an adapted version of the Heterogeneous Graph Transformer (HGT) architecture proposed in [38], computes a node-level embedding for each vehicle via message passing-based aggregation. This fixed-sized vector encoding summarizes the social and map-related information required to predict its future movement in the current traffic environment. In order to exploit all properties of the graph-structured data, we implement an edge-enhanced HGT architecture that computes attention weights and messages based on both node and edge features. To accelerate learning, we further carry out three (trainable) encoding steps that are applied before the graph convolution layers:

- For encoding the delta-time attribute Δt_e of the VTV edges, we adopt Time2Vec [39], a learnable vector representation of time that lets us capture periodic and non-periodic time series patterns.
- To get fixed-sized node representations for the lanelet geometries, we use a Gated Recurrent Unit (GRU) [40] which encodes the variable-length waypoint sequences ${}^{L}V_l$ and ${}^{L}V_r$.
- Finally, we adopt a learnable vector embedding to encode the L2L adjacency type $\tau_{\mathcal{E}}$.

2) Decoder component: Based on the encoded vehicle representations, a GRU decoder network generates a fixed-length sequence of local position and orientation deltas for each vehicle, which is aggregated to obtain a sequence of predicted vehicles states. This is done by repeatedly updating the decoded vehicle state by a local transition in the coordinate frame of the previous state. The model is trained to minimize the average displacement error (ADE) between the predicted and ground-truth vehicle trajectories. An example of the resulting predictions with a prediction horizon of 1.0 s and a time interval of 0.2 s is shown in Fig. 8, whereas the quantitative results are summarized in Tab. IV.

[4] `https://commonroad.in.tum.de/datasets`
[5] `https://commonroad.in.tum.de/tools/dataset-converters`
[6] The model implementation can be found in the *cr-geo* repository.

TABLE IV: Experimental results from the cities included in the Nuplan dataset. Each experiment is trained and validated on the datasets collected in the same city. We use average displacement error and final displacement error (FDE) as evaluation metrics.

Dataset split	**Number of scenarios**	**ADE** [m]	**FDE** [m]
Singapore	2372	0.106	0.227
Boston	938	0.138	0.316
Pittsburgh	1560	0.215	0.454

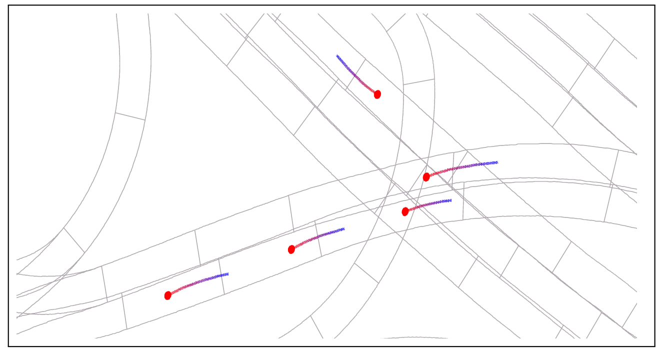


Fig. 8: Predicted trajectories from NuPlan's Singapore dataset.

V. Conclusion

This paper presents *cr-geo*, an open-source Python package offering a standardized interface for map-aware graph extraction from traffic scenarios. As a pioneering effort, it serves as a flexible framework that lets users collect custom graphical PyTorch datasets tailored to their research needs. Exemplified by our trajectory prediction implementation, it minimizes the time spent by researchers on writing boilerplate code for dataset collection. *cr-geo* complements the CommonRoad software platform, which offers a converter tool for popular traffic datasets. With ease of use and extension being our core design goals, *cr-geo*'s flexible interface is achieved by delegating and encapsulating all steps of the traffic graph creation. We invite researchers to contribute to *cr-geo* to further enhance its capabilities.

Acknowledgements

This research was funded by the German Research Foundation grant AL 1185/7-1 and the Federal Ministry for Digital and Transport through the project KoSi.

References

[1] S. Lefèvre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," vol. 1, no. 1, pp. 1–14, 2014.
[2] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.
[3] I. Chami, S. Abu-El-Haija, B. Perozzi, C. Ré, and K. Murphy, "Machine learning on graphs: A model and comprehensive taxonomy," vol. 23, no. 89, pp. 1–64, 2022.
[4] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *Proc. of the The World Wide Web Conference*, 2019, pp. 2022–2032.
[5] P. W. Battaglia, J. B. Hamrick, V. Bapst, *et al.*, "Relational inductive biases, deep learning, and graph networks," 2018. arXiv: `1806.01261`.

[6] T. S. Jepsen, C. S. Jensen, and T. D. Nielsen, "Graph convolutional networks for road networks," in *Proc. of the International Conference on Advances in Geographic Information Systems*, 2019, pp. 460–463.

[7] F. Diehl, T. Brunner, M. T. Le, and A. Knoll, "Graph neural networks for modelling traffic participant interaction," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2019, pp. 695–701.

[8] H. Jeon, J. Choi, and D. Kum, "SCALE-Net: Scalable vehicle trajectory prediction network under random number of interacting vehicles via edge-enhanced graph convolutional neural network," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020, pp. 2095–2102.

[9] M. Huegle, G. Kalweit, M. Werling, and J. Boedecker, "Dynamic interaction-aware scene understanding for reinforcement learning in autonomous driving," in *Proc. of the IEEE International Conference on Robotics and Automation*, 2020, pp. 4329–4335.

[10] P. Hart and A. Knoll, "Graph neural networks and reinforcement learning for behavior generation in semantic environments," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2020, pp. 1589–1594.

[11] X. Li, X. Ying, and M. C. Chuah, "GRIP: Graph-based Interaction-aware Trajectory Prediction," in *Proc. of the IEEE Intelligent Transportation Systems Conference*, 2019, pp. 3960–3966.

[12] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun, "Learning lane graph representations for motion forecasting," in *Proc. of the European Conference on Computer Vision (ECCV)*, 2020, pp. 541–556.

[13] W. Zeng, M. Liang, R. Liao, and R. Urtasun, "LaneRCNN: Distributed representations for graph-centric motion forecasting," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021, pp. 532–539.

[14] H. Zhao, J. Gao, T. Lan, *et al.*, "TNT: Target-driveN trajectory prediction," in *Conference on Robot Learning*, 2020.

[15] B. Kim, S. Park, S. S. Lee, E. Khoshimjonov, D. Kum, J. Kim, J. S. Kim, and J. W. Choi, "LaPred: Lane-aware prediction of multi-modal future trajectories of dynamic agents," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 14 631–14 640.

[16] L. Zhang, P. Li, J. Chen, and S. Shen, "Trajectory prediction with graph-based dual-scale context fusion," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022, pp. 11 374–11 381.

[17] F. Janjos, M. Dolgov, and J. M. Zöllner, "StarNet: Joint action-space prediction with star graphs and implicit global-frame self-attention," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2022, pp. 280–286.

[18] J. Ngiam, V. Vasudevan, B. Caine, *et al.*, "Scene Transformer: A unified architecture for predicting future trajectories of multiple agents," in *International Conference on Learning Representations*, 2022.

[19] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, "GOHOME: Graph-oriented heatmap output for future motion estimation," in *Proc. of the International Conference on Robotics and Automation*, 2022, pp. 9107–9114.

[20] X. Mo, Z. Huang, Y. Xing, and C. Lv, "Multi-agent trajectory prediction with heterogeneous edge-enhanced graph attention network," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 9554–9567, 2022.

[21] R. Krajewski, J. Bock, L. Kloeker, and L. Eckstein, "The highD Dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems," in *Proc. of the International Conference on Intelligent Transportation Systems*, 2018, pp. 2118–2125.

[22] W. Zhan, L. Sun, D. Wang, *et al.*, "INTERACTION Dataset: An INTERnational, Adversarial and Cooperative moTION Dataset in Interactive Driving Scenarios with Semantic Maps," 2019. arXiv: `1910.03088`.

[23] H. Caesar, J. Kabzan, K. S. Tan, W. K. Fong, E. Wolff, A. Lang, L. Fletcher, O. Beijbom, and S. Omari, "NuPlan: A closed-loop ml-based planning benchmark for autonomous vehicles," 2021. arXiv: `2106.11810`.

[24] C. Morris, N. M. Kriege, F. Bause, K. Kersting, P. Mutzel, and M. Neumann, "TUDataset: A collection of benchmark datasets for learning with graphs," in *ICML Workshop on Graph Representation Learning and Beyond*, 2020.

[25] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford large network dataset collection*, `http://snap.stanford.edu/data`, 2014.

[26] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017, pp. 719–726.

[27] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[28] A. Paszke, S. Gross, F. Massa, *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. of Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

[29] Y. Sun and J. Han, "Mining heterogeneous information networks: A structural analysis approach," *ACM SIGKDD Explorations Newsletter*, vol. 14, no. 2, pp. 20–28, 2013.

[30] P. Bender, J. Ziegler, and C. Stiller, "Lanelets: Efficient map representation for autonomous driving," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2014, pp. 420–425.

[31] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-RNN: Deep learning on spatio-temporal graphs," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition,*, 2016, pp. 5308–5317.

[32] E. Héry, S. Masi, P. Xu, and P. Bonnifait, "Map-based curvilinear coordinates for autonomous vehicles," in *Proc. of the IEEE International Conference on Intelligent Transportation Systems*, 2017, pp. 1–7.

[33] G. Booch, I. Jacobson, and J. Rumbaugh, *Unified Modeling Language Reference Manual, The Second Edition*. Addison-Wesley Professional, 2004.

[34] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

[35] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, 1994.

[36] P. A. Lopez, E. Wiessner, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flotterod, R. Hilbrich, L. Lucken, J. Rummel, and P. Wagner, "Microscopic Traffic Simulation using SUMO," in *Proc. of the International Conference on Intelligent Transportation Systems*, 2018, pp. 2575–2582.

[37] M. Klischat, O. Dragoi, M. Eissa, and M. Althoff, "Coupling SUMO with a motion planning framework for automated vehicles," in *Simulating Connected Urban Mobility*, 2019, pp. 1–9.

[38] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," in *Proc. of The Web Conference*, 2020, pp. 2704–2710.

[39] P. Diniz, D. A. D. Junior, J. O. B. Diniz, A. C. de Paiva, A. C. da Silva, M. Gattass, R. Quevedo, D. Michelon, C. Siedschlag, and R. Ribeiro, "Time2Vec transformer: A time series approach for gas detection in seismic data," in *Proc. of the Symposium on Applied Computing*, 2022, pp. 66–72.

[40] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *Proc. of the Workshop on Syntax, Semantics and Structure in Statistical Translation*, 2014, pp. 103–111.