A Graph-based Framework for Coverage Analysis in Autonomous Driving

Thomas Mühlenstädt and Marius Bause

No Institute Given

1 Abstract

tbd

2 Introduction

In autonomous driving, coverage analysis is a crucial step to ensure the safety and reliability of the system. In most situations, coverage arguments are collected either per coverage factor, or maybe up to 2 or 3 factor interactions. See for example [**?**] for an production grade implementation of state of the art coverage analysis. In contrast to existing approaches, this paper proposes a graph-based framework for coverage analysis. There are already other graph-based approaches for analysing and representing traffic scenes. However, the work in that paper is not specifically focused on coverage analysis. Hence in this paper, graph based traffic scene representations are utilized for coverage analysis.

This paper is structured as follows: In chapter 3, existing coverage and analysis approaches are discussed. The basis for the following chapters is layed in chapter 4. Afterwards, two coverage approaches using the traffic scene graphs are discussed in chapter 5 and chapter 6. Finally, the developed methods are applied to Carla and Argoverse 2.0 data in chapter 8 followed by a short summary and outlook.

3 Existing coverage and analysis approaches

add Master thesis Johannes

There exist a large amount of literature on coverage analysis in the context of autonomous driving. In the following, some of the most relevant approaches are discussed.

The PEGASUS project ([**?**]) introduces a systematic, scenario-based methodology for the verification and validation of highly automated driving functions, addressing the impracticality of traditional distance-based testing approaches. The core of the method is a six-layer model used to systematically describe and structure the driving environment, encompassing factors from road geometry to weather conditions. This framework is particularly relevant for coverage analysis, as it facilitates a structured decomposition of the vast, continuous test space into

discrete, manageable "logical scenarios". By systematically parameterizing and exploring these logical scenarios across simulation, proving ground, and field tests, the methodology aims to ensure the completeness of relevant test runs and provides a structured foundation for arguing that the automated system has been adequately tested across its entire operational design domain (ODD). This shift from random sampling to a structured, scenario-based approach allows for a more efficient and comprehensive assessment to generate evidence for a final safety argumentation

In a similar direction, [**?**] focus on defining ontoligies for automated vehicles in an object oriented manner, with the intention to have a clearly defined and implementable structures for scenarios, delivering a clear linkage to coverage arguments.

The authors of ([**?**]) address the challenge of validating automated driving systems in complex urban environments by proposing a trajectory-based clustering method for real-world driving data. They extract motion trajectories and associated features from urban driving recordings, apply unsupervised clustering to group similar driving behaviours/scenes, then analyze the resulting clusters to reveal common scenario types and redundancies. Their approach enables structuring the enormous scenario space, supporting more efficient test-set generation and scenario selection for automated vehicle verification. While not explicitly focused on coverage analysis, their approach is a good starting point for coverage analysis.

In [**?**], the authors identify that key concepts in automated driving—namely scene, situation, and scenario—are inconsistently defined in the literature, which complicates the development, testing and validation of driving-automation modules. They propose clear definitions: a scene is a snapshot of the environment including dynamic and static elements plus actors' self-representations; a situation is the set of all circumstances relevant for behaviour decision at a given moment, derived from the scene but reflecting the actor's goals and values; and a scenario is a temporal development of several scenes in sequence, involving actions/events and goals/values. The paper also provides example implementations of these definitions in the context of automated-vehicle systems and how they interface with perception, planning and control, and testing.

Another important reference is the SAE International recommended practice [**?**], which provides many foundational terms in the context of autonomous driving. It provides a functional taxonomy and clear definitions for key terms related to driving automation systems (DAS) used in on-road motor vehicles. The document defines six levels of driving automation (Levels 0 through 5) based on the role of three primary actors — the human driver, the driving automation system (DAS), and other vehicle systems/components. It introduces important related terms such as the Dynamic Driving Task (DDT), Operational Design Domain (ODD), Automated Driving System (ADS), and Vehicle Motion Control, among others. The 2021-04 revision (superseding the 2018 version) was developed in collaboration with ISO/TC 204/WG14 to harmonise global terminology and improve clarity for multi-discipline audiences (engineering, legal,

media). The document emphasises that it is descriptive (not normative); it does not prescribe specifications or impose performance requirements for DAS.

The paper [**?**] proposes a scenario-based framework for developing and validating automated-driving systems across different development phases. It introduces three abstraction levels of scenarios—functional, logical, and concrete—and discusses how these can be transformed for use in testing. The authors note that existing parameter-selection methods, such as equivalence-class or combinatorial testing, lack a systematic way to determine meaningful test coverage, highlighting coverage analysis as an open challenge in scenario-based validation.

In [**?**], a standard textbook on software testing although not specifically focused on autonomous driving, the authors provide an introduction to software testing, which is a good starting point for coverage analysis. They introduce the concept of coverage and discuss the different types of coverage, such as statement coverage, branch coverage, condition coverage, and decision coverage. They also discuss the different techniques for measuring coverage, such as branch coverage, condition coverage, and decision coverage.

A much more focussed example of coverage analysis in the context of autonomous driving is provided by [**?**] in a blog post, discussing topics like items, parameters and coverage buckets and performance metrics like time to collision and error collections.

The authors of ([**?**]) investigate the significant challenge of safety validation and production release for fully autonomous vehicles, positing that established testing concepts are insufficient as they fundamentally rely on the human driver's ability to intervene as a safety backup. The authors introduce the "approval-trap," a statistical argument demonstrating the unfeasibility of proving superior safety through real-world driving, which would necessitate billions of test kilometers. This validation gap is presented as a fundamental problem of coverage analysis, where the technical system must now demonstrably cover the vast operational domain previously managed by the human. The paper concludes that overcoming this challenge requires a paradigm shift toward new test case generation methodologies, such as critical scenario identification, and the extensive use of validated simulation-based tools to achieve sufficient test coverage efficiently.

Well known standards for coverage analysis in the context of autonomous driving are the ISO 21448 ([**?**]) and the UL 4600 ([**?**]). The ISO 21448 is a standard for the safety of the intended functionality of road vehicles, including automated driving systems. It defines a framework for coverage analysis, including the definition of coverage criteria and the measurement of coverage. The UL 4600 is a standard for the safety of autonomous products, including automated driving systems. It defines a framework for coverage analysis, including the definition of coverage criteria and the measurement of coverage.

4 Defining a traffic scene graph

still to be added: `https://sagroups.ieee.org/adwg/wp-content/uploads/sites/661/2024/10/ADWG_STV2_whitepaper.pdf`

still to be added: https://www.vvm-projekt.de/securedl/sdl-eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1Ni
eyJpYXQiOjE3NDc2OTIxNjgsImV4cCI6MTc0Nzc4MjE2OCwidXNlciI6MCwiZ3JvdXBzIjpbMCwtMV0sImZpbGUiOiJ
f9d402f084002b57/VVMethoden_Zusammenfassender_Abschlussbericht_v1.0_
zur_Veroeffentlichung.pdf

[?]
[?]
[?]

4.1 Map Graph Construction

The map graph serves as the foundational structure for traffic scene graph construction, encoding the spatial relationships between lanes in the road network. It is a directed multigraph $G_{\text{map}} = (V_{\text{map}}, E_{\text{map}})$ where nodes represent individual lanes and edges encode three fundamental spatial relationships between lanes.

Each node $v \in V_{\text{map}}$ represents a lane segment and stores geometric and semantic information including:

- Lane boundaries (left and right boundaries as polylines)
- Lane length
- Lane centerline
- Intersection status (whether the lane is within an intersection)
- Road type and lane type information

The map graph defines three edge types that capture different spatial relationships:

1. **Following edges** ($e_{\text{following}}$): Connect lanes that form a continuous path in the same direction.
2. **Neighbor edges** (e_{neighbor}): Connect adjacent lanes traveling in the same direction.
3. **Opposite edges** (e_{opposite}): Connect lanes traveling in opposite directions.

The map graph construction algorithm processes the original map data from Argoverse or CARLA to extract these relationships. For intersection detection, lanes are analyzed for geometric overlap, and all overlapping lanes are marked as intersection lanes to ensure proper handling of complex road geometries.

4.2 Actor Graph Construction

The actor graph $G_{\text{actor}} = (V_{\text{actor}}, E_{\text{actor}})$ represents the dynamic relationships between vehicles and other actors in a traffic scene at a specific timestep. This graph is constructed using a two-phase algorithm that separates relation discovery from graph construction. The second step serves to reduce the number of relations in the graph for computational tractability, and enables the representation of relationships between actors through indirect paths involving multiple intermediate nodes.

Each node $v \in V_{\text{actor}}$ represents an actor (vehicle, pedestrian, etc.) and stores the following attributes:

- **Primary lane ID**: The lane on which the actor is primarily located
- **Lane IDs**: List of all lanes the actor occupies (for actors spanning multiple lanes)
- **Longitudinal position** (s): Position along the lane's centerline
- **3D position** (x, y, z): Cartesian coordinates
- **Longitudinal speed**: Speed along the lane direction
- **Actor type**: Classification (vehicle, pedestrian, etc.)
- **Lane change indicator**: Boolean flag indicating if the actor changed lanes from the previous timestep

Note that the node structure is extensible—users can add additional attributes such as acceleration, heading angle, or vehicle dimensions as needed for their coverage model.

Each edge $e \in E_{\text{actor}}$ represents a relationship between two actors and contains:

- **Edge type**: One of four relation types between actors (leading, following, neighbor, opposite)
- **Path length**: Distance in meters along the lane network

Similar to nodes, edge attributes can be extended with additional information such as relative velocities, time-to-collision, or interaction probabilities.

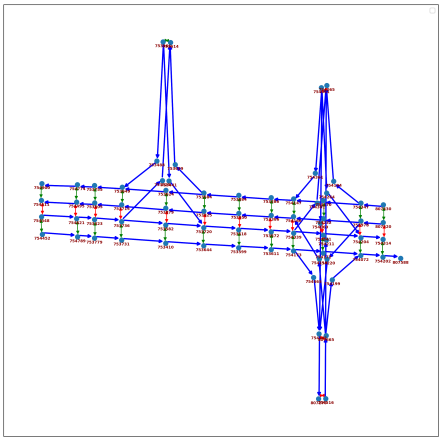
The actor graph defines four types of relationships between actors, ordered by semantic hierarchy:

1. **Following/Leading** (`following_lead`, `leading_vehicle`): Represents longitudinal relationships where one actor follows another in the same direction and lane.
2. **Neighbor** (`neighbor_vehicle`): Represents lateral relationships between actors on lanes traveling in the same direction. The actors need not be on immediately adjacent lanes.
3. **Opposite** (`opposite_vehicle`): Represents relationships between actors on lanes traveling in opposite directions. Similar to neighbor relationships, the actors need not be on immediately opposite lanes.

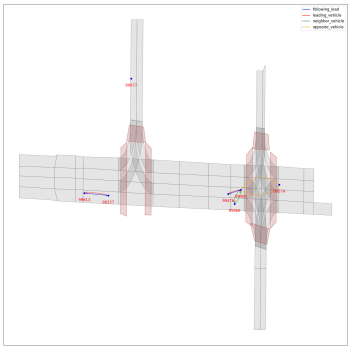
4.3 Hierarchical Graph Construction Algorithm

The actor graph construction follows a two-phase approach that separates relation discovery from graph building, enabling hierarchical processing and preventing redundant edge creation.

Phase 1: Relation Discovery The discovery phase identifies all potential relationships between actor pairs based on distance constraints and map graph structure. For each pair of actors (A, B) at timestep t, the algorithm determines their primary lanes and checks if a path exists in the map graph connecting them. The path structure determines the relationship type:



(a) Example graph representation of a map.



(b) Example traffic graph.

Fig. 1: The map graph (left) serves as the foundational structure for determining actor relations, encoding spatial relationships between lanes. The traffic scene (right) shows actors and their relationships at a specific timestep. Note that actors can be disconnected if they are far enough apart, as the graph construction algorithm only creates edges between actors within the specified distance thresholds.

- **Following/Leading**: If both actors are on the same lane, or if the path consists entirely of following edges.
- **Neighbor**: If the path contains exactly one neighbor edge and the remaining edges are following edges. Note that this does not require the actors to be on immediately adjacent lanes—the path may contain multiple following edges before and after the single neighbor edge.
- **Opposite**: If the path contains exactly one opposite edge and the remaining edges are following edges. Similar to neighbor relationships, this does not require the actors to be on immediately opposite lanes.

Each relationship is validated using two distance checks: the lane-based path length ensures that curved roads are properly accounted for, while the Euclidean distance ensures that only actors in close proximity are considered. All discovered relationships are stored in a relations dictionary for the construction phase.

Phase 2: Hierarchical Graph Construction The construction phase builds the actor graph by adding edges in a hierarchical order, ensuring that higher-priority relationships are established first and redundant edges are prevented.

Construction Order Edges are added in three stages, following the semantic hierarchy:

1. **Leading/Following edges** (highest priority): All following/leading relationships are processed first, sorted by path length (shortest first).
2. **Neighbor edges** (medium priority): Neighbor relationships are processed second, sorted by absolute path length.
3. **Opposite edges** (lowest priority): Opposite relationships are processed last, sorted by absolute path length.

Redundancy Prevention Before adding an edge between actors A and B, the algorithm checks if a path already exists in the current actor graph with length $\leq$ the maximum node distance for that relation type. This prevents redundant edges that would create triangles or longer indirect paths when shorter direct paths exist.

For leading/following and neighbor relationships, each direction ($A \to B$ and $B \to A$) is checked independently, as these relationships can be asymmetric. For opposite relationships, if either direction has an existing path, both directions are rejected, as opposite relationships are inherently symmetric.

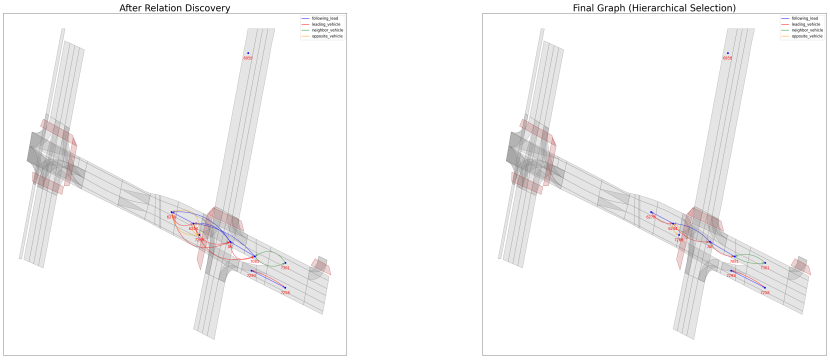
The path checking uses breadth-first search to find the shortest path in the actor graph, counting the number of edges (not nodes) in the path. This ensures that if a path exists with k edges where $k \leq \text{max\_node\_distance}$, the direct edge is not added.

The graph is updated immediately after each edge addition, ensuring that path checks for subsequent edges use the current graph state. This incremental approach guarantees that the hierarchical construction works correctly and prevents race conditions.

Figure **??** illustrates the effect of redundancy prevention by comparing the graph after relation discovery with the final graph after hierarchical selection. The discovery graph respects the maximum distances chosen for discovery, but still contains redundant relations. For example, three vehicles in a row can be described by pairwise lead-follow relations, but the relation of the last vehicle to the first vehicle is already encoded in the graph through the intermediate vehicle. Similarly, for a row of vehicles to a neighboring or opposite vehicle, we only need the opposite/following relation to the closest vehicle in the row; the other actor relations are encoded in the graph. This redundancy prevention reduces the number of edges in the graph significantly, making it more efficient while preserving all necessary connectivity information.

Input Parameters Table 1 summarizes all input parameters for the actor graph construction algorithm, their descriptions, and the values used in our experiments.

The distance limits in the discovery phase determine which potential relationships are identified, while the node distance limits in the construction phase control redundancy prevention. The asymmetric limits for opposite relationships (100m forward vs. 10m backward) reflect that vehicles traveling in opposite directions are more relevant when they are ahead rather than behind.



(a) After relation discovery (b) Final graph after hierarchical selection

Fig. 2: Comparison of the actor graph after relation discovery and after hierarchical selection with redundancy prevention. The discovery graph contains all relations found within the distance limits, while the final graph removes redundant edges that can be represented through existing paths, significantly reducing the number of edges while preserving connectivity.

All parameters are configurable and can be adjusted based on the specific application requirements, road network characteristics, or desired graph density. The values shown in Table 1 represent default values that work well for urban traffic scenarios, but may need adjustment for highway scenarios (longer distances) or dense urban intersections (shorter distances).

4.4 Time based graph representations

The graph creation strategy described above is assuming to create a graph for a single timestep. One possible extension is to create a graph which contains information for multiple timesteps. In order to create a time indexed graph, first the actor graphs for all timesteps are created. The joint set of all nodes across all timesteps is then determined. Node attributes of single timestep graphs are combined into a tensor of node attributes, introducing a new dimension for the time index. While the nodes are somehow time independent by this strategy, the edges are modeled as time dependent. Here it is utilized, that there can be multiple edges between the same pair of nodes, keeping potential edge attributes and adding an additional edge attribute to indicate the time stamp of the edge.

This extension allows the graph to easily represent changing states, e.g. the relationship between two vehicles changes from neighbor to leading vehicle, i.e. a cut in happened. One important special case for this type of processing is to handle actors, which are not present in all time steps, i.e. they appear and disappear during the time window. Especially if node attributes are used in

Table 1: Input parameters for actor graph construction

Parameter	Type	Description	Value
Distance limits (discovery phase)			
`max_distance_lead_veh_m`	float	Maximum distance in meters for leading/following relationships	100
`max_distance_neighbor_forward_m`	float	Maximum distance in meters for forward neighbor relationships	50
`max_distance_neighbor_backward_m`	float	Maximum distance in meters for backward neighbor relationships	50
`max_distance_opposite_forward_m`	float	Maximum distance in meters for forward opposite relationships	100
`max_distance_opposite_backward_m`	float	Maximum distance in meters for backward opposite relationships	10
Node distance limits (construction phase)			
`max_node_distance_leading`	int	Maximum number of edges in actor graph path for leading/following	3
`max_node_distance_neighbor`	int	Maximum number of edges in actor graph path for neighbor	2
`max_node_distance_opposite`	int	Maximum number of edges in actor graph path for opposite	2
Timestep configuration			
`delta_timestep_s`	float	Time step increment in seconds for temporal graph creation	1.0

the processing, a suitable imputation strategy is necessary to avoid non-equal dimensions of tensors for node attributes.

The methods derived in the following chapters can be applied to this extended graph representation as well, even though the focus in this paper will be on single timestep graphs.

5 Create subgraphs for coverage analysis

There is a lot of knowledge in the literature on how to define archetypes of traffic scenes. Once an archetype is defined, a special property of graphs can be used. Two graphs are isomorphic if they have the same structure, regardless of the node and edge labels. As the archetypes are not necessarily involving a lot of actors, these are more like subsets of actual traffic scenes. A very simple example might be 2 vehicles on the same lane, driving in the same direction and another vehicle driving on a neighboring lane. This situation can be represented by a graph with 3 nodes and 2 edges. In most real traffic situations however, there will be additional actors present, so that we are not searching for isomorphic graphs, but rather want to check if any subgraph of G is isomorphic to the archetype graph A. This is an example of a subgraph isomorphism problem. While this problem is NP-hard, the graphs considered here are rather small, so the computational time is reasonable. One such algorithm is the VF2 algorithm,

which is implemented in the NetworkX library (see [**?**]). The strategy we are then applying is the following:

1. Define a set of subgraphs S that are considered to be archetypes of traffic scenes, e.g. unprotected left turns with opposite traffic or lead vehicle following situations.
2. Define which node and edge attributes are considered for the isomorphism check.
3. Create an empty dataframe C with a column for each subgraph in S
4. Define the set of traffice scenes (e.g. from Carla or Argoverse) defined as graphs G
5. For each graph G, check if any subgraph of G is isomorphic to any subgraph in S and note the result in a new row in table C

This strategy can be described to some degree as a bottom up approach: Starting from a detail level, individual situations are defined. Then going upwards to different datasets, it is checked, if the archetype is present. Also, follow up analysis of the created coverage dataframe can be performed. For example,

- The distribution of numeric attributes like speed and and distance to other actors can be visualized for the subset of all traffic scenes which are subgraph isomorphic to an archetype.
- It can be cross tabulated, which combinations of archetypes are jointly present in a traffic scene.
- Pass Fail rates or other AV performance metrics can be calculated for the subset of all traffic scenes which are subgraph isomorphic to an archetype.

In our case, we defined 18 subgraph archetypes covering common traffic scenarios:

- Simple 2-actor patterns (only used if 2 actors are isolated): following, opposite traffic, and neighboring vehicles
- Complex 3-actor patterns: lead vehicle with neighbor, platoon formations, opposite traffic configurations, and lane change scenarios (cut-in), with variants at intersections
- Complex 4-actor patterns: cut-out scenarios, multi-vehicle platoons, and lead-following with opposite traffic, with intersection variants
- Complex 5-actor patterns: combined lead, neighbor, and opposite vehicle scenarios with intersection variants

The node coverage analysis across Argoverse and CARLA datasets achieved 62% overall coverage. It is posible to define more archetypes or detect them automatically, but this is out of scope for this project.

6 Graph Embeddings for Traffic Scene Analysis

This section describes the concepts, implementation and application of graph embeddings to traffic scene graphs. The implementation follows a comprehensive

approach to learning graph representations through self-supervised contrastive learning.

Embeddings are a widely used method to translate raw data like images or text into an embedding space in order to be able to perform machine learning tasks on them. One well known example of this is the Word2Vec model, which is used to translate words into a vector space, where the distance between vectors can be used to measure the similarity between words ([**?**]).

In the context of traffic scene graphs, embeddings are used to translate the graph structure into a vector space, where the distance between vectors can be used to measure the similarity between traffic scenes. This is useful for coverage analysis, as it allows to compare traffic scenes among each other. For example, two traffic scenes can be considered similar if the distance between their embeddings is small. This enables to search for a most similar simulation scenario given a real world scenario, to identify areas with near duplicates or to easily visualize structures in the embedding space, which in the original space of all possible traffic scenes would not be possible.

Graph neural networks (GNNs) are a class of neural networks that are designed to process graph-structured data and have gained a lot of popularity in the last years, see for example (add references).

In this paper, a network architecture using a Graph Isomorphism Network with Edge features (GINE) as described in [**?**] is used to generate embeddings for traffic scene graphs as implemented in the pytorch geometric library ([**?**]). Main reason for using this specific architecture is that is capable of learning embedding representations not only on the graph structure itself but on both node and edge attributes. Other network architectures like GraphSAGE or GAT are not capable of this. (TODO: check if this is true)

The exact architecture of the model is shown in Figure 2. The features used are the actor type (as a one hot encoding), the actor speed (float), if the actor is on an intersection (boolean) and if the actor changed its lane since the last timestep (boolean) for the nodes. For the edges, the edge type (as a one hot encoding) and the path length (float) between the two nodes are used.

The model has been trained on the CARLA and Argoverse 2.0 datasets using self-supervised contrastive learning.

Training employed a self-supervised contrastive objective on mini-batches of 128 graphs. For each batch, two correlated views of every graph were created by perturbing continuous attributes with zero-mean Gaussian noise ($\sigma = 0.1$) applied to node longitudinal speed and edge path length. A four-layer GINE encoder (hidden width 96) produced graph-level representations via the concatenation of mean, max, and sum pooling, followed by an embedding MLP (yielding 256-dimensional embeddings) and a projection head. The contrastive loss was a temperature-scaled cross-entropy over in-batch similarities (cosine similarity of ℓ_2-normalized projections, temperature $\tau = 0.1$), maximizing agreement of the two views of the same graph while contrasting against other graphs in the batch. Optimization used Adam with an exponentially decaying learning rate, initialized at 0.02 and multiplied by 0.75 across successive stages. This setup follows

established practice in contrastive pretraining for GNNs [**?**] and is implemented using PyTorch Geometric [**?**].

7 Analysing a traffic scene with a graph

Having defined a graph-based traffic scene representation, we can now analyse the coverage of the system. Two methodologies are proposed for this purpose: One is to define archetypes of traffice scenes, and to compare graphs from observed traffic scenes to these archetypes. The second one is to translate graphs to graph embeddings, and then to compare the embeddings of different sets of traffic scenes.

8 Application

8.1 Argoverse 2.0

[**?**]

TODO MARIUS: Describe the Argoverse 2.0 dataset.

8.2 Carla

CARLA (Car Learning to Act, [**?**]) is an open-source simulator specifically designed for autonomous driving research and development. It provides a highly realistic urban driving environment with diverse road layouts, weather conditions, and traffic scenarios. The simulator features a comprehensive sensor suite simulation, flexible API for scenario creation, and supports both learning-based and traditional autonomous driving approaches. CARLA enables researchers to test and validate autonomous vehicle systems in a safe, controllable environment before real-world deployment.

The simulator has gained widespread adoption across both academic and industrial settings. In research, CARLA serves as a standard platform for developing and benchmarking autonomous driving algorithms, including reinforcement learning approaches for vehicle control and sensor fusion techniques [**?**]. Industry applications include virtual testing of production autonomous vehicle systems, scenario-based validation pipelines, and integration with hardware-in-the-loop testing frameworks [**?**]. CARLA is also extensively used in autonomous driving competitions and challenges, providing a common evaluation environment for comparing different approaches across research groups worldwide.

Here, Carla version 0.9.15 is used. The CARLA version 0.10.0 is not used, because it had only 2 maps and Mine_1 (which is not really normal roads) at the start of this project. Specifically, the following maps were used: Town01, Town02, Town03, Town04, Town05 and Town07. Plots of these maps are shown in Figure 3.

The data generation script implements sophisticated behavior control mechanisms to create diverse and realistic traffic scenarios. Multiple vehicle types

including trucks, motorcycles, and regular cars are spawned with varying probabilities, each exhibiting different behavioral characteristics such as speed preferences, following distances, and lane-changing tendencies. The script incorporates dynamic behavior modifications during simulation, including random slowdowns, periodic behavior changes, and adaptive responses to traffic conditions, resulting in rich and varied traffic scene data across multiple CARLA maps and simulation iterations. The simulation runs have between 20 and 60 vehicles each.

The resulting data consists of xxx scenes with 11 seconds of simulation time each, in order to have a similar data size as the Argoverse 2.0 dataset.

8.3 Subgraph Isomorphism Coverage Analysis

We apply the subgraph isomorphism approach to both the CARLA and the Argoverse data to identify coverage scenarios. The archetypes used here are defined manually and described in table xxx. The chosen archetypes are typical traffic situations like e.g. lead vehicle situations, lane change situations or different combinations of opposite direction vehicles. Also, information like which traffic actors are on an intersection are incorporated into the archetypes. The results are shown in Figure 4 to Figure 7. In Figure 4 there is a clear signal that for both datasets the coverage is not uniform per the different archetypes. Also, the distribution for the Carla dataset is not close to the Argoverse distribution, indicating that the CARLA data is not representative for the Argoverse data. Even worse, the Carla dataset is nearly completely missing out e.g. on the cut_out_intersection archetype, clearly indicating a coverage gap in the Carla dataset. A next step of analysis is to check, which archetypes are occuring simultaneously in a traffic scene. Figure 5 shows the agreement matrix for the manually defined coverage scenarios for CARLA and Argoverse. The heatmaps show the percentage of agreement between the manually defined coverage scenarios for CARLA and Argoverse.

TODO: Write a sentence about the agreement matrices once using the actual actor graph defintion.

A last example of how to use the assignment of archetypes to traffic scenes is to check the parameter distribution for the speed and path length of the traffic scenes. Figure 6 and Figure 7 show the parameter distribution for the speed and path length of the traffic scenes for CARLA and Argoverse. The plots show that the distribution for the CARLA data is not close to the Argoverse distribution, indicating that the CARLA data is not representative for the Argoverse data. Given the assignment of archetypes to traffic scenes based on the actors can be used in further analysis not done here, e.g. to check more divers parameters and distributions.

8.4 Graph Embeddings Coverage Analysis

The resulting embeddings have been analysed in a number of ways. For plausibility checks, for a number of randomly samples scenarios, the scenario with closest embedding vector (euclidean distance) has been visualized. This is shown

in Figure 10. This includes comparison between CARLA and Argoverse scenarios.

As a next step, the embedding space has been analysed using PCA and t-SNE. This is shown in Figure 9. This can be done in a number of different flavors, like for example:

- distinguishing between CARLA and Argoverse scenarios by a color coding,
- visualiizing just the CARLA scenarios and color code them by the map, in order to see if the maps deliver different types of scenarios,
- by calculating a density distribution of the embedding space for the Argoverse scenarios, and to check if for a regions with a density about a certain threshold, there exist CARLA scenarios, i.e. do a coverage analysis in the embedded space
- do the same vice versa, i.e. to check for relevance of CARLA scenarios.

As the embedding space is a normal metric space, representing scenarios across a large range of different driving situations, these embeddings can be used also for many other tasks, like for example clustering, anomaly detection, similarity search, etc.

And, as the main task considered here is coverage analysis, the embeddings can be used to check if a target distribution is met by a test distribution in the embedded space. Specifically, considering the Argoverse 2.0 scenarios as the target distribution, and the CARLA scenarios as the test distribution, the embeddings can be used to check if the CARLA scenarios cover the Argoverse 2.0 scenarios in the embedded space.

9 Summary

- A method for using traffic scenes represented as graphs for coverage analysis has been developed in this paper.
- Starting from defining rules for the actor graph construction, the resulting graphs are then used by 2 different approaches for coverage analysis.
- Firstly, a non-AI approach is used where a set of archetypes is defined (which is done here manually but in principle could be done automatically as well).
- Then it is checked, which archetypes are present in a traffic scene and how often.
- Secondly, a graph embedding approach is used to create a vector space of the traffic scenes and to check if the target distribution is met by the test distribution in the embedded space.
- The results for this approach are shown in Section 8, being applied to Carla and Argoverse 2.0 data.
- The main advantage of this approach over traditional coverage analysis approaches is that it very easily scales to a large number of traffic scenarios archetypes and different driving situations.
- Besides the definition of the actor graph, no specific handling of different types of traffic situations is necessary.

- In other approaches (cite TNO Streetwise) it is necessary to manually define for each scenario type how it is defined. Here the only thing necessary is to define the actor graph construction rules as well as potentially the archetypes as graphs.
- Also, the presented approach easily allows for different number of actors in the traffic scene without excluding actors or only focussing on a subset of actors.
- Next steps in the same line of this research will be to further optimize the actor graph to include several time steps into one graph.
- Also, automatically extracting archetypes from real world traffic observation data will be investigated.
- The source code for this research is available at `https://github.com/tmuehlen80/graph_coverage`.

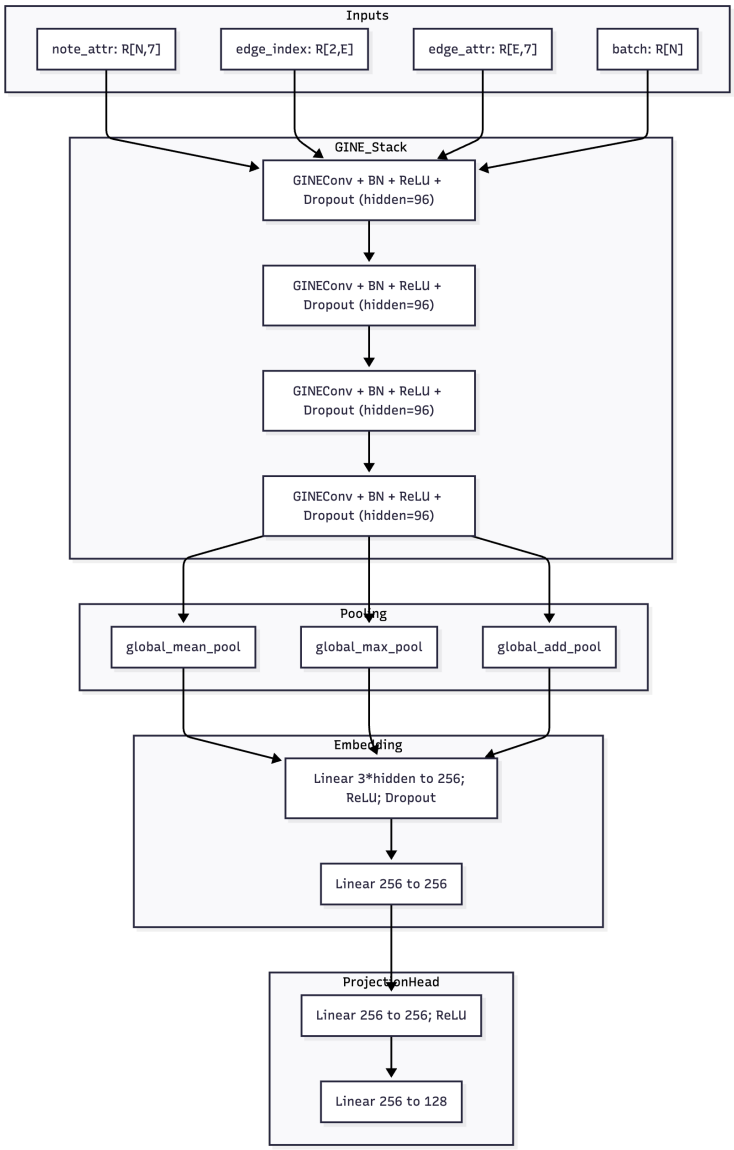


Fig. 3: Model architecture for the Graph Isomorphism Network with Edge features (GINE).

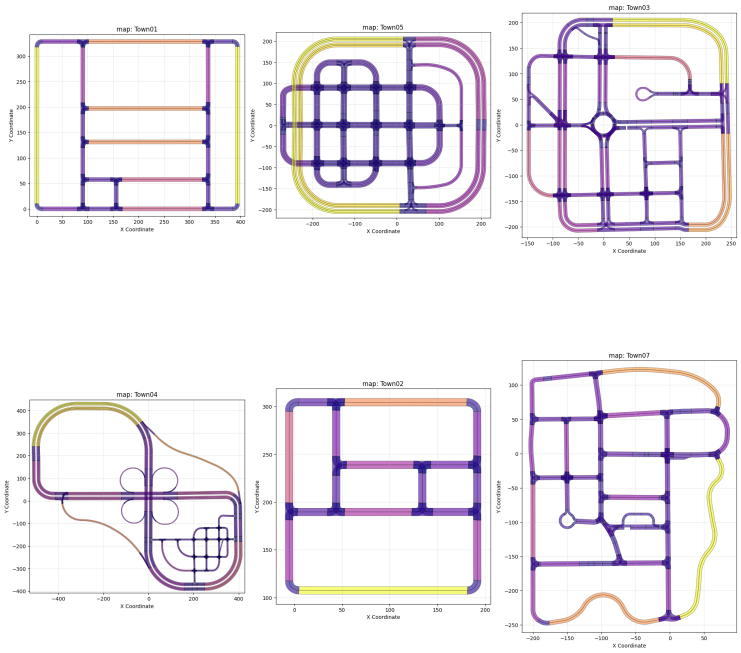
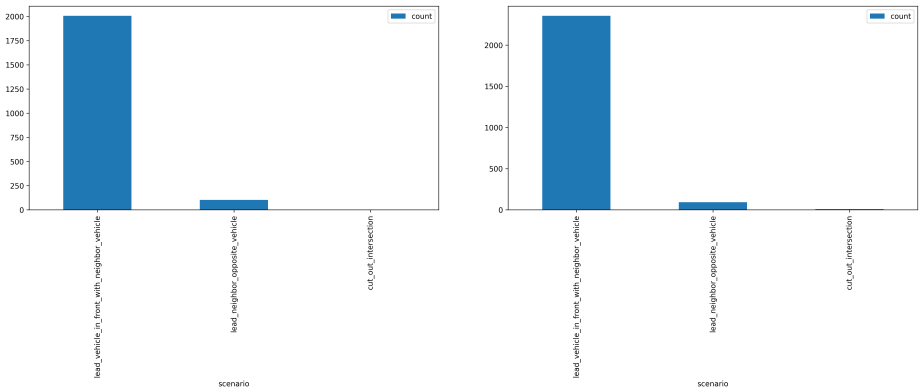
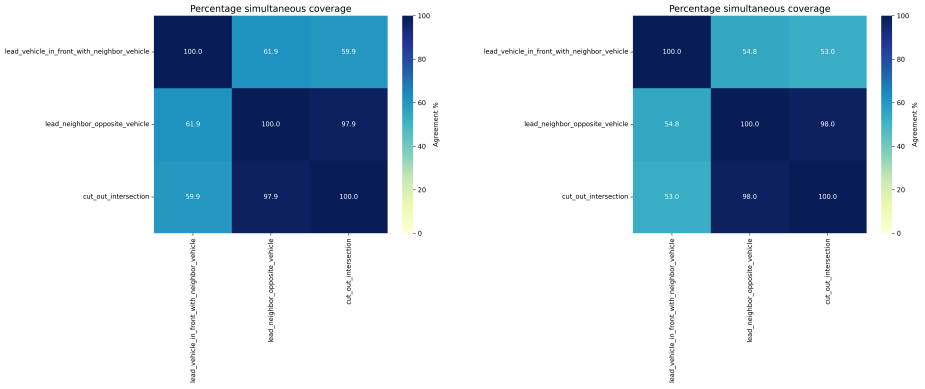


Fig. 4: Overview of CARLA maps used in the simulation study: Town01, Town02, Town03, Town04, Town05, and Town07. These maps provide diverse urban driving environments with varying road layouts, intersections, and traffic patterns.



(a) CARLA coverage scenarios (b) Argoverse coverage scenarios

Fig. 5: Coverage barcharts for the manually defined coverage scenarios for CARLA and Argoverse.



(a) Agreement matrix for CARLA.

(b) Agreement matrix for Argoverse.

Fig. 6: Agreement matrix for manually defined coverage scenarios for CARLA and Argoverse.

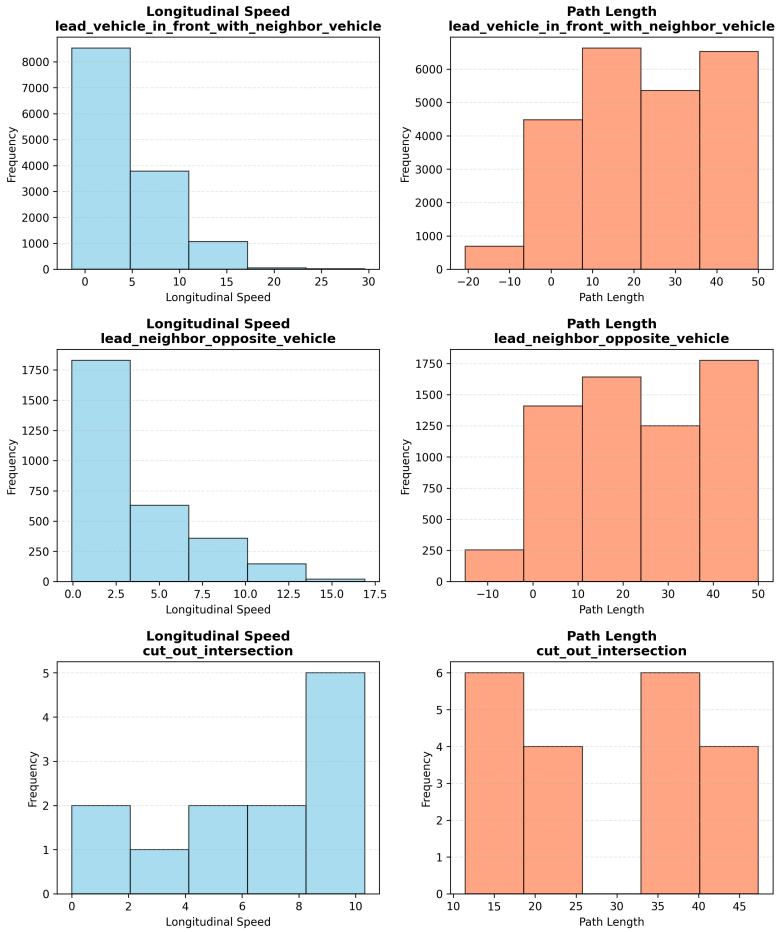


Fig. 7: Parameter distribution for speed and path length for CARLA.

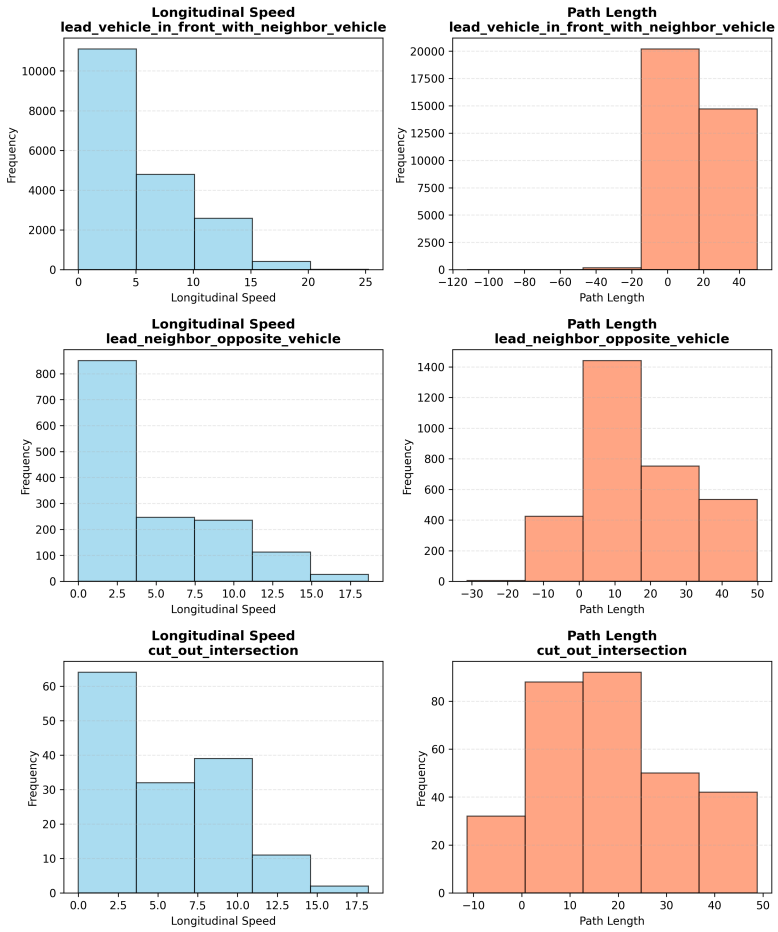


Fig. 8: Parameter distribution for speed and path length for Argoverse.

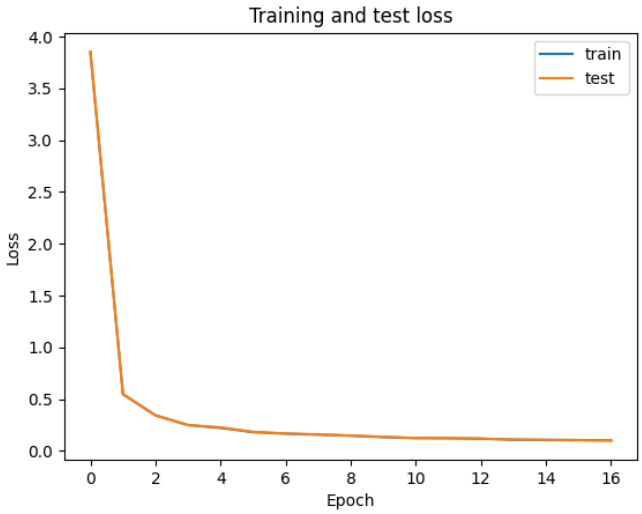


Fig. 9: Training and test loss for the graph embeddings model trained jointly on CARLA and Argoverse 2.0 data.

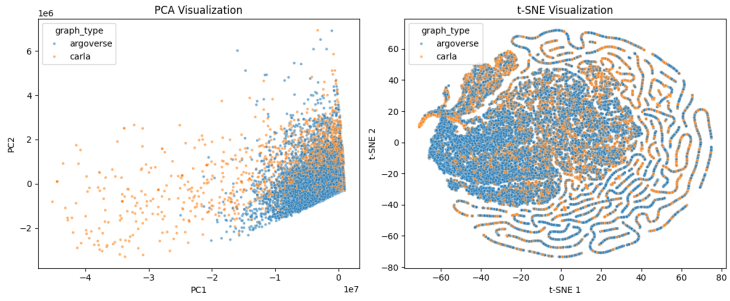
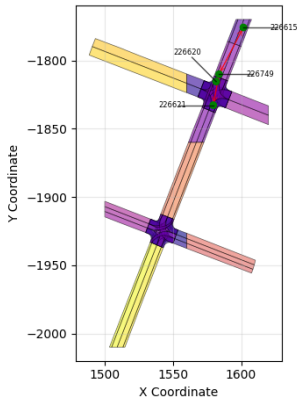
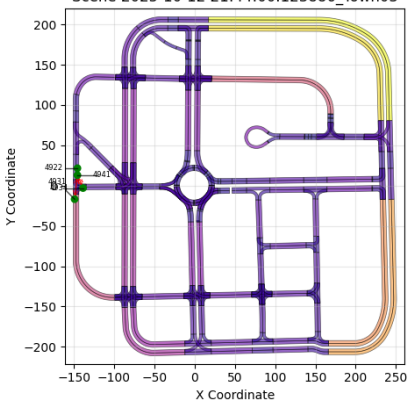


Fig. 10: PCA and t-SNE visualization of the embedding space for Carla and Argoverse 2.0 scenarios.

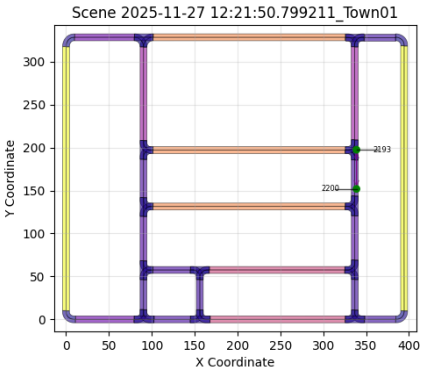
Scene 49a33c17-1937-4b6b-a5ec-3f2d9cdb32d8



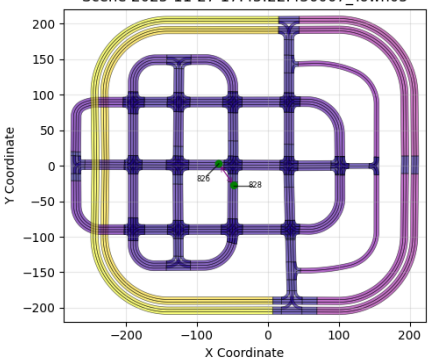
Scene 2025-10-12 21:44:00.123886_Town03



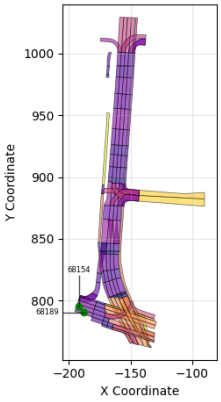
Scene 2025-11-27 12:21:50.799211_Town01



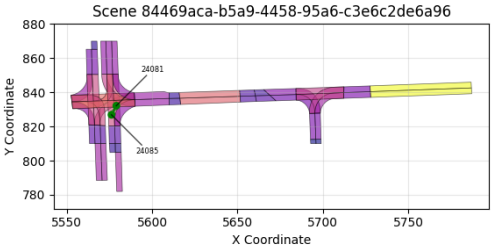
Scene 2025-11-27 17:45:22.436067_Town05



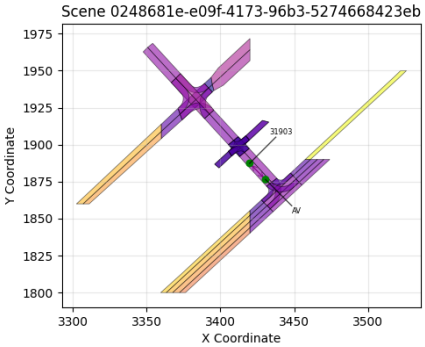
Scene 22cb2942-d35b-4815-86b1-1eb2d522cf67



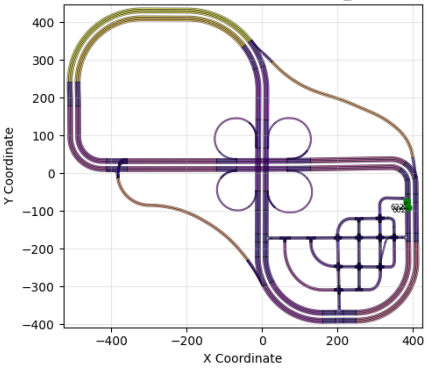
Scene 84469aca-b5a9-4458-95a6-c3e6c2de6a96



Scene 0248681e-e09f-4173-96b3-5274668423eb



Scene 2025-11-27 16:57:55.062665_Town04



Scene eb266466-4d41-40f5-ae8c-92afe7fcab90