A Graph-based Framework for Coverage Analysis in Autonomous Driving

Thomas Mühlenstädt and Marius Bause

October 20, 2025

1 Abstract

test4

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 [9] 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 6 and chapter 7. 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 larger The PEGASUS project ([18]) 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

The authors of ([19]) 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 [20], 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 [17], 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 [14] 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 [1], 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.

[9]
[5]
[21]
[8]
[13]

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...eyJpYXQiOjE3NDc2OTIxNjgsImV4cCI6MTc0Nzc4MjE2OCwidXNlciI6MCwiZ3JvdXBzIjpbMCwtMV0sImZpbGUiOiJm.../f9d402f084002b57/VVMethoden_Zusammenfassender_Abschlussbericht_v1.0_zur_Veroeffentlichung.pdf`

[16]
[2]
[10]

4.1 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.

- create a graph for multiple timesteps
- create a joint set of nodes.
- For each time step, determine the corresponding edges.
- Add all edges to the graph, resulting in a di-graph, which can contain for a pair of nodes as many edges as there are time steps.

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 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 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.

6 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 are 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 [4]). 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.

7 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 ([15]).

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 [11] is used to generate embeddings for traffic scene graphs as implemented in the pytorch geometric library ([7]).

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)

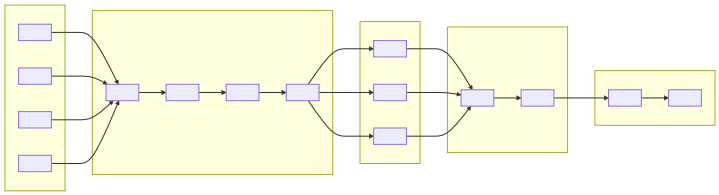


Figure 1: Model architecture for the Graph Isomorphism Network with Edge features (GINE).

The exact architecture of the model is shown in Figure 1. 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 [11] and is implemented using PyTorch Geometric [7].

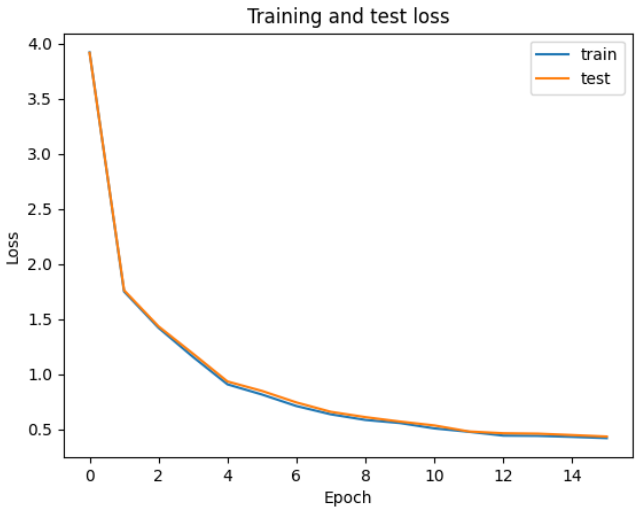


Figure 2: Training and test loss for the graph embeddings model trained jointly on CARLA and Argoverse 2.0 data.

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 as well as cosine distance) has been visualized. This is shown in Figure 8. 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 3. This can be done in a number of different flavors, like for example:

- distinguishing between CARLA and Argoverse scenarios by a color coding,

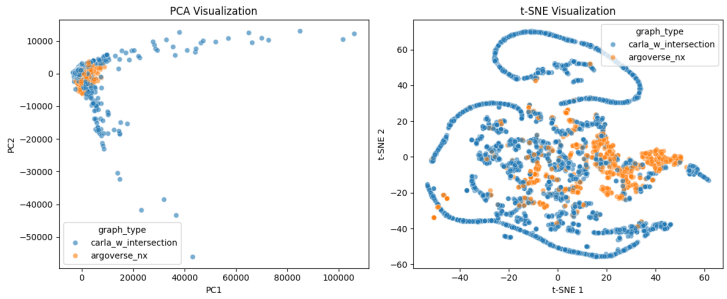


Figure 3: PCA and t-SNE visualization of the embedding space for Carla and Argoverse 2.0 scenarios.

- 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.

The results for this approach are shown in Section xxx.

8 Application

8.1 Argoverse 2.0

[22]

8.2 Carla

CARLA (Car Learning to Act, [6]) 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 [3]. Industry applications include virtual testing of production autonomous vehicle systems, scenario-based validation pipelines, and integration with hardware-in-the-loop testing frameworks [12]. 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 4.

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.

Table 1: My Table

scn 1	scn 2	scn 3	scn 4	size
False	False	False	False	613
False	False	True	False	3
True	True	False	True	377
True	True	True	True	7

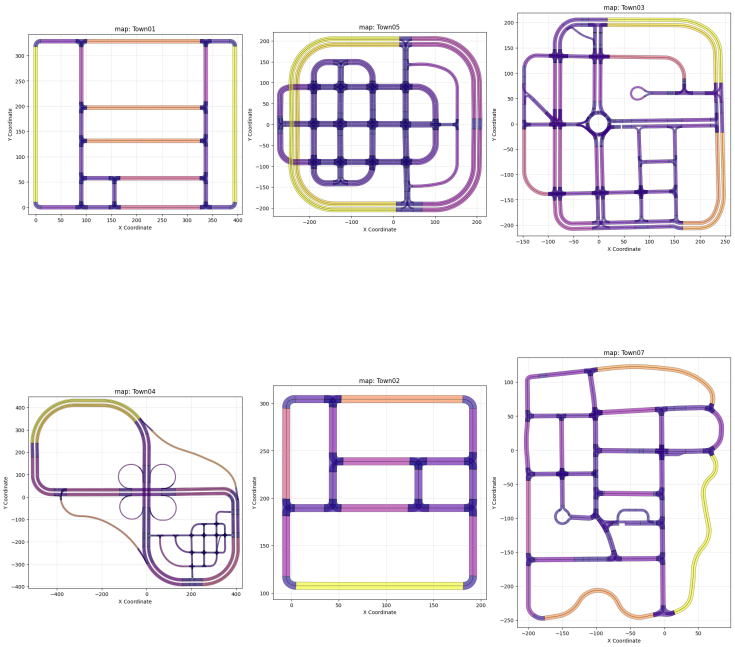


Figure 4: Overview of CARLA maps used in the simulation study: Town01, Town02, Town03, Town04, Town05, and Town07. These maps provide diverse urban driving environments with varying road layouts, intersections, and traffic patterns.

9 Summary

References

[1] Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, 2008.

[2] Gerrit Bagschik, Till Menzel, and Markus Maurer. Ontology based scene creation for the development of automated vehicles, 2018.

[3] Felipe Codevilla, Eder Santana, Antonio M. López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving, 2019.

[4] Luigi P. Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, oct 2004.

[5] Erwin de Gelder, Jan-Pieter Paardekooper, Arash Khabbaz Saberi, Hala Elrofai, Olaf Op den Camp, Steven Kraines, Jeroen Ploeg, and Bart

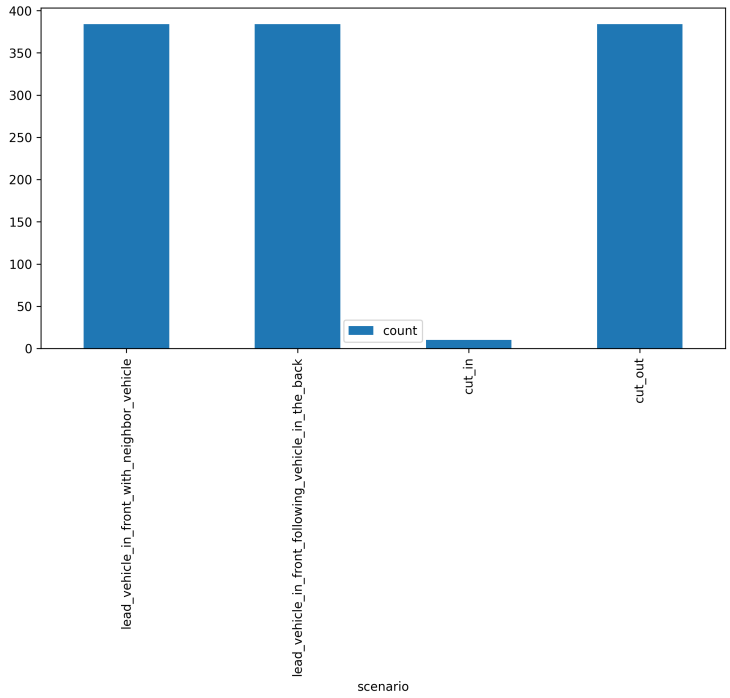


Figure 5: Coverage barchart for the manually defined coverage scenarios for CARLA.

De Schutter. Towards an ontology for scenario definition for the assessment of automated vehicles: An object-oriented framework. *IEEE Transactions on Intelligent Vehicles*, 7(2):300–314, 2022.

[6] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[7] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[8] International Organization for Standardization. ISO 21448:2022 – Road Vehicles – Safety of the Intended Functionality, 2022. Standard document, ISO/TC 22/SC 32.

[9] Foretellix. Av coverage and performance metrics. `https://blog.foretellix.com/2019/09/13/av-coverage-and-performance-metrics/`, September 2019. Accessed: 2025-04-04.

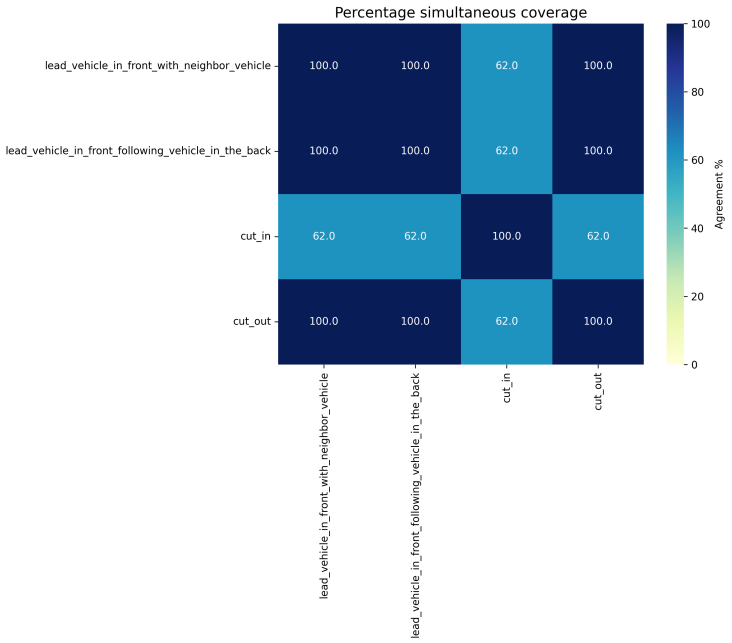


Figure 6: Agreement matrix for manually defined coverage scenarios for CARLA.

[10] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: a language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, page 63–78, New York, NY, USA, 2019. Association for Computing Machinery.

[11] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks, 2020.

[12] Bernhard Jaeger, Kashyap Chitta, and Andreas Geiger. Hidden biases of end-to-end driving models, 2023.

[13] Underwriters Laboratories. *UL 4600: Standard for Safety for the Evaluation of Autonomous Products*. UL Standards, 2020. Standard for autonomous vehicle safety and validation.

[14] Till Menzel, Gerrit Bagschik, and Markus Maurer. Scenarios for development, test and validation of automated vehicles. *CoRR*, abs/1801.08598, 2018.

[15] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[16] M.E.J. Newman. *Networks, An Introduction*. Oxford University Press, 2010.

[17] On-Road Automated Driving (ORAD) Committee. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Standard J3016, SAE International, April 2021.

[18] PEGASUS. Pegasus method. Available online, 2019. Accessed: October 20, 2025.

[19] Lennart Ries, Philipp Rigoll, Thilo Braun, Thomas Schulik, Johannes Daube, and Eric Sax. Trajectory-based clustering of real-world urban driving sequences with multiple traffic objects. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 1251–1258, 2021.

[20] Simon Ulbrich, Till Menzel, Andreas Reschka, Fabian Schuldt, and Markus Maurer. Defining and substantiating the terms scene, situation, and scenario for automated driving. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 982–988, 2015.

[21] Walther Wachenfeld and Hermann Winner. *The Release of Autonomous Vehicles*, pages 425–449. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[22] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, Deva Ramanan, Peter Carr, and James Hays. Argoverse 2: Next generation datasets for self-driving perception and forecasting. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS Datasets and Benchmarks 2021)*, 2021.

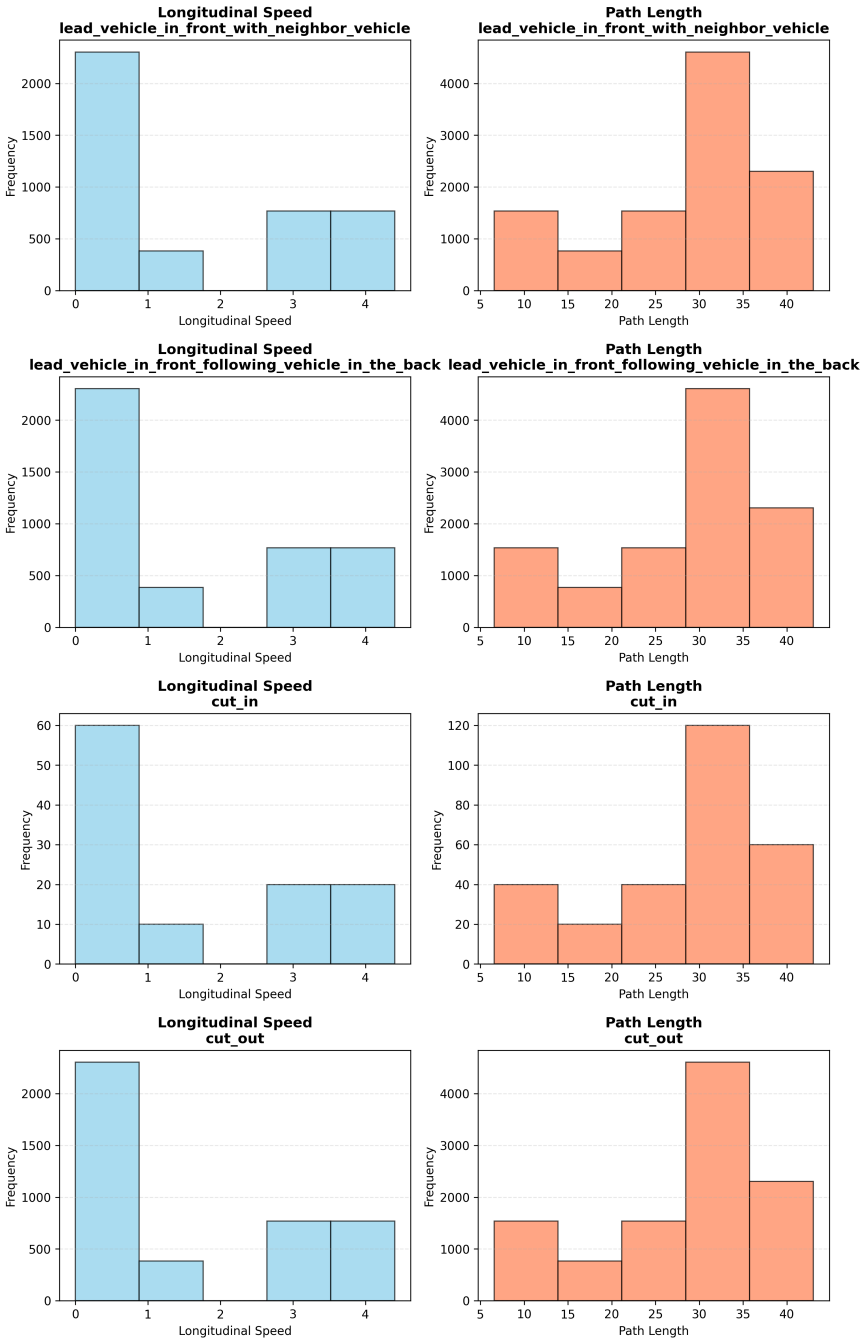
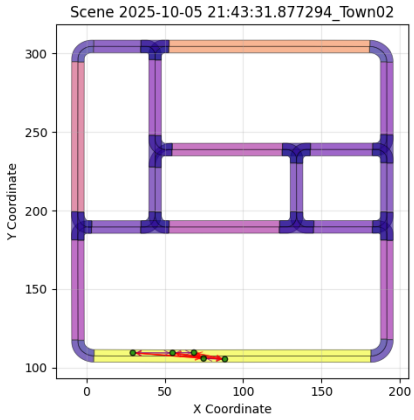
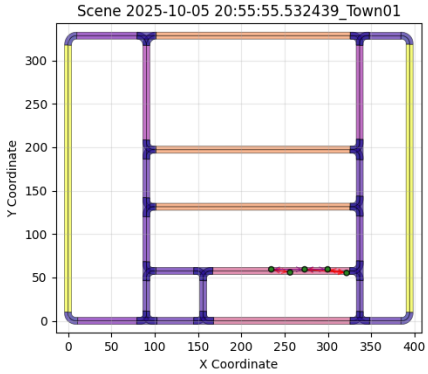
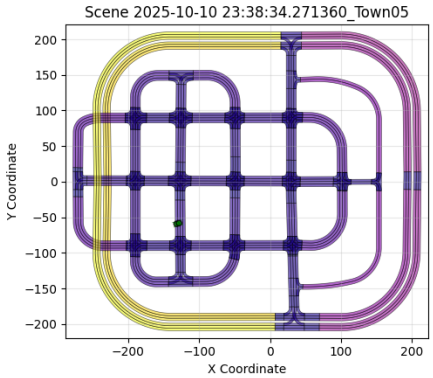
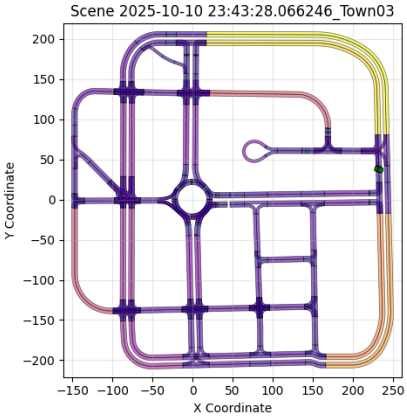


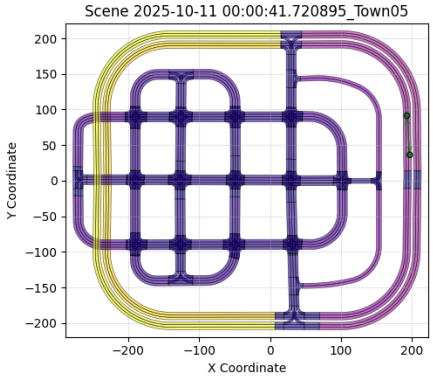
Figure 7: Agreement matrix for manually defined coverage scenarios for CARLA.

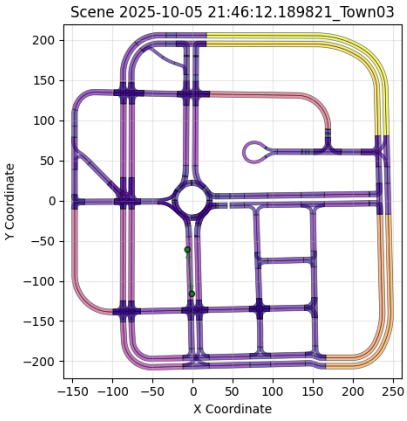


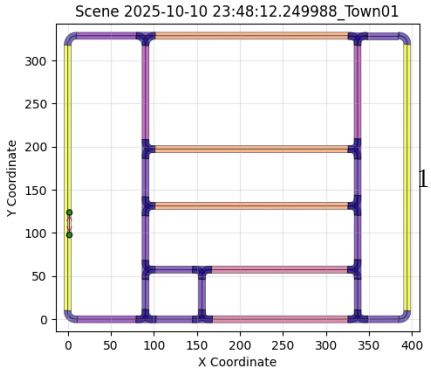


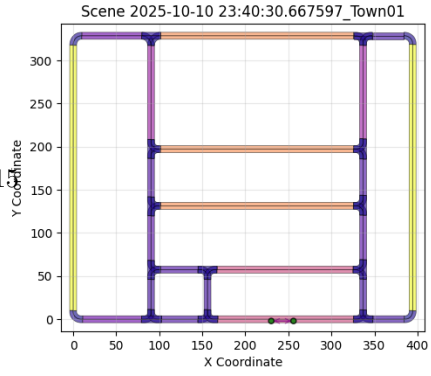












Scene 2025-10-10 23:35:58.762011_Town03