UAB
Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

Vidal Guerra, Daniel Alejandro. *Predicció i control generalitzables de trajectòries de múltiples vehicles a interseccions arbitràries mitjançant informació d'estat i intenció*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Grau en Intel·ligència Artificial)

This version is available at https://ddd.uab.cat/record/336304

under the terms of the BY-SA license.

Generalizable Multi-Vehicle Trajectory Prediction and Control at Arbitrary Intersections Using State and Intention Information

Vidal Guerra, Daniel Alejandro

June 29, 2026

Abstract

This thesis follows and continues the work from [1] about predicting and controlling the trajectory of multiple vehicles at intersections using state and intention information. This thesis addresses the problem of predicting vehicle trajectories at diverse intersections. While their approach demonstrates promising results using message passing between agents to avoid collisions and previous states information, it fails to generalize to unseen intersection layouts since the model does not adapt to changes in road geometry, leading to unrealistic predictions, potential collisions with road boundaries, collisions between vehicles or vehicles leaving the roadway. To address this limitation, this work extends the original pipeline by generating a diverse dataset of intersections with varying geometries and topologies. A new representation and learning strategy is introduced to enable the model to infer and adapt to the underlying intersection structure, thereby improving generalization to arbitrary intersection types. The system's performance is evaluated through both quantitative metrics and qualitative visualization, exploring the model's adaptability and trajectory prediction performance across a variety of intersection configurations.

Keywords: Trajectory Prediction, Control, Intersection Generalization, GNN, Traffic Simulator

1 INTRODUCCTION

Intersections are the places where driving becomes more complex. Vehicles converge, priorities change, and small delays in making decisions can escalate into risky situations. A system that can anticipate how multiple vehicles will move a few seconds into the future helps planning, reduces collisions, and smooths traffic flow. Reliable trajectory prediction is therefore an important element of safe autonomous driving and cooperative assistance.

This thesis builds on [1], who predict multi-vehicle trajectories at intersections using each vehicle's current state and a high-level intention (go straight, turn left, turn right). Their model exchanges information between agents through message passing to capture interactions. Results are promising and reliable for standard intersections. But the approach struggles when the intersection layout at test time differs from the layouts seen during training. The model does not adapt to new lane geometries or altered topologies, which can lead to unrealistic paths, boundary contacts, or vehicles leaving the drivable area. In short, generalization to unseen intersections remains limited.

In this project, this limitation is addressed with two ideas that work together. First, a larger and more varied set of intersection scenarios is generated, covering a broad range of geometries and topologies. This increases the diversity of training data and exposes the model to diverse cases. Second, an intersection-aware representation and learning strategy were adopted to make the layout explicit to the predictor, so the network can infer structure and adapt its outputs to the local road shape. The goal is simple: keep the strengths of intent conditioning and message passing from [1], while improving robustness to new layouts. The method is evaluated with standard trajectory metrics in open-loop and with CARLA-SUMO simulations that reveal the performance of the model in real-time and closed-loop for qualitative results. The expectation is better lane adherence, safer motion, and stronger transfer to intersections not seen during training.

2 OBJECTIVES

The overall goal of this thesis is to design and evaluate a trajectory prediction and control model that can generalize to previously unseen intersection layouts by exploiting both

• Contact E-mail: DanielAlejandro.Vidal@autonoma.cat
• Supervised by: Jordi Casas Roma (Computer Science Department)
• Academic Year 2025/26

vehicle state and intersection layout information. The ideal result would be a multi-trajectory prediction model that has a flawless performance at any given intersection. The intersections are considered four roads that intersect at the center of the map (coordinates (0, 0) in the simulation), which can have any orientation and different path widths, carry traffic in both directions, and may have a priority lane or none.

2.1 Obtain more diverse data

This objective focuses on generating additional and more diverse data, where the intersections have different layouts, so that a model with better generalization to different intersections can be trained.

- Create new intersection layouts using the traffic simulator from SUMO with varied geometries and topologies (inclined approaches, different lane widths, rotated crosses, etc.).
- Generate additional traffic data on each intersection by simulating many episodes with different route files and traffic intensities. The data must consist of graphs of vehicles where the data represents the x and y position coordinates of the vehicles, along with their respective angle of orientation and intended action; the edges of the graphs capture the interactions between the vehicles, which provide information on whether they can collide with each other because their paths are crossed. Each graph must be associated in some way with information related to the shape of the corresponding intersection.
- Implement a method to capture the layout information from the intersections and link this information with the graph data to train an intersection-aware model.
- Build train, validation, and test splits that mix layouts and traffic conditions to evaluate generalization systematically.

2.2 Design a suitable architecture

Since the new data might have a different structure or input size, it might be necessary to redesign the model architecture or adapt it to the new data. This adaptation may consist of a simple input dimension increase if the intersection layout data is concatenated or combined with the vehicles data, or a new structure of the model if the layout data is in a different format and requires different processing, e.g., convolutions or other forms of embeddings.

2.3 Train the new model

This objective focuses on training the model using the extended dataset and evaluating its performance across multiple intersection layouts.

- Implement a new dataloader that builds batches of traffic graphs and links each to its corresponding intersection data.
- Train the model on multiple intersection layouts simultaneously, monitoring metrics such as ADE, FDE, miss rate, and collision rate.
- Evaluate the model on seen and unseen layouts in open-loop to obtain metrics and closed-loop for qualitative evaluation.

2.4 Test and analyze generalization

This objective focuses on testing the trained model in simulation and analyzing whether it generalizes to unseen intersection layouts.

- Integrate the trained model into the CARLA–SUMO co-simulation to observe if the performance of the model is able to 1) lead all vehicles to their destinations, 2) avoid collisions between vehicles or against the road boundaries, 3) simulate the give way according to the priority lanes correctly.
- Compare performance against the baseline model from [1] in terms of trajectory accuracy, collision rate, and qualitative behavior in CARLA.
- Analyze failure cases on challenging unseen layouts to identify limitations and directions for future work.

3 METHODOLOGY

This thesis follows a **Kanban-based project management approach** adapted for academic supervision, which emphasizes continuous progress evaluation and flexible prioritization. The main elements are:

3.1 Regular progress review

Meetings with my thesis tutor are organized at least every two weeks. At each meeting: - Progress on previous objectives is reviewed and validated. - New short-term objectives and next steps are jointly decided. - Any encountered issues, decisions, or workarounds are discussed.

3.2 Task tracking and documentation

- All intermediate steps, experiments, hyperparameters, and intermediate results are documented in a **central checklist** (maintained in Markdown or GitHub wiki). - This checklist serves as a visual Kanban board: tasks move from "To Do" $\rightarrow$ "In Progress" $\rightarrow$ "Done/Validated". - Code, data, and model checkpoints are version-controlled with Git to ensure reproducibility.

3.3 Additional good practices

- **Weekly self-checkpoints**: Short personal reviews (15–30 min) every Friday to update the checklist and flag blockers before tutor meetings. - **Risk mitigation**: Parallel experiments (e.g., multiple backbones) when time allows, to hedge against dead ends. - **Validation gates**: No advancement to the next general objective until key metrics (e.g., baseline reproduction) are met. - **Reproducibility focus**: isolated environments for SUMO/CARLA to avoid setup issues; seeded random states for experiments.

This structure ensures stable progress, early issue detection, and a step-by-step history for the final thesis defense.

4 Planning

The work plan for this thesis is organized into four main stages, aligned with the general objectives described above. Figure 10 in appendix C, shows the Gantt chart for the period from February to June 2026, including the intermediate deliveries and final submission.

4.1 Stage 1: Initial setup and first delivery

The first stage runs from 11/02/2026 to 16/03/2026 and ends with the first delivery. During this period the objectives, methodology, and planning of the project are defined in detail, and the state of the art is reviewed and written. This stage establishes the baseline: it clarifies the problem, identifies relevant related work, and produces the initial chapters of the report.

4.2 Stage 2: Data extension and model design

The second stage covers the period from 13/03/2026 to 15/05/2026 and culminates in the second delivery. The main goal is to obtain more diverse data and design a new architecture able to exploit it. Concretely, new intersection layouts are created in SUMO, additional traffic data are generated for these layouts, and representations will be made to represent the layout information. In parallel, research is carried out on possible architectural modifications, and the new map-conditioned model is trained on the extended dataset.

4.3 Stage 3: Evaluation and refinement

The third stage runs from 18/05/2026 to 12/06/2026 and corresponds to the third delivery. In this phase the trained model is tested both offline and in the CARLA–SUMO co-simulation, and its performance is compared with the baseline. If necessary, the architecture and training procedure are refined to improve the results, and the main chapters of the thesis (methods, experiments, results, discussion) are written and polished.

4.4 Stage 4: Finalization and submission

The final stage extends from 12/06/2026 to 30/06/2026 and is devoted to preparing the final submission. The written thesis is completed and revised, the best models and experimental artifacts are saved in a reproducible form, and the documentation and code are cleaned and organized. This stage ensures that the final report and accompanying material are ready for evaluation.

5 State of the Art

5.1 Interaction-aware trajectory prediction at intersections

Predicting vehicle motion at urban intersections requires modeling strong dependencies between agents, since each vehicle's future path depends not only on its own movement and state but also on the behavior and intention of surrounding vehicles. For this reason, many recent approaches use graph-based or interaction-aware models that represent vehicles as nodes and their relations as edges, enabling message passing across the scene.

Several works have successfully applied this idea to multi-agent trajectory prediction. Graph-based methods such as GRIP, multi-agent tensor fusion, and adversarial interaction-aware prediction explicitly model dependencies among surrounding actors and have shown strong performance in multi-vehicle settings [2–4]. Later works extended this line with more expressive spatio-temporal or heterogeneous graph formulations, including AST-GNN, EvoST-GAT, SRAI-LSTM, and heterogeneous edge-enhanced attention models [5–8]. These approaches are especially relevant at intersections, where priority, turning maneuvers, and conflict resolution depend on nearby vehicles.

The baseline model that motivates this thesis follows the same general principle. Zhu *et al.* propose a centralized GNN-based method for multi-vehicle trajectory prediction at intersections using vehicle state and high-level intention information, with message passing to aggregate interactions among agents [1]. This formulation is well-suited to cooperative intersection scenarios because it can reason jointly about multiple vehicles and their intended maneuvers. However, it does not explicitly encode the geometry of the intersection itself, which makes it vulnerable to overfitting to the layouts seen during training.

5.2 Map-aware representations for motion forecasting

A complementary line of research has shown that static scene structure is essential for motion forecasting. At intersections in particular, feasible trajectories are constrained by lane geometry, road connectivity, and turning radius, so accurate prediction requires some representation of the intersection layout.

One way to introduce this context is through rasterized images or occupancy maps, which can be processed by convolutional neural networks. This idea is attractive because it is simple and compatible with existing image-based encoders. In this thesis, it was the first map-aware strategy explored: each SUMO layout was converted into a binary raster image and then encoded into a compact embedding that conditioned the trajectory predictor; see some examples in figure 6. The use of an image encoder is also consistent with the broader vision-based modeling tradition in autonomous driving, where convolutional backbones remain a common way to extract spatial features [9, 10].

At the same time, multiple works suggest that structured map encodings are more effective when road topology matters. VectorNet represents HD maps and agent trajectories as vectorized polylines instead of images, arguing that rasterization is lossy and less efficient for motion prediction [11]. LaneGCN further develops this idea by constructing a lane graph from raw map data and modeling actor-to-lane and lane-to-lane relations explicitly [12]. These methods support the view that road geometry should not remain implicit when the objective is to generalize to new intersections.

5.3 Generalization across scenarios

Generalization beyond the training scenario remains a major challenge in trajectory prediction. A model can perform well on one fixed intersection, but it degrades dramatically when tested on layouts with different orientations, lane widths, or connectivity, even if the local traffic behavior is similar.

This issue has been studied from the perspective of multiple scenario transfer and domain knowledge. Ma *et al.* show that incorporating self-supervised domain knowledge can improve driving-behavior prediction across different scenarios and better support zero-shot or few-shot transfer [13]. Similarly, the inD dataset provides large-scale naturalistic trajectory recordings at German intersections and highlights the importance of evaluating motion prediction on realistic, diverse junction geometries [14]. These works reinforce the idea that layout diversity is not just a data issue but a crucial part of the prediction problem.

Intersection coordination and conflict management have also been studied from a control perspective. Work on non-signalized intersection navigation, decision-making based on conflict resolution, and real-time coordination of connected vehicles demonstrates that the geometry and priority structure of the intersection are essential for safe behavior [15–18]. In a related direction, cooperative automated driving and V2X communication studies emphasize that vehicles may need shared contextual information to resolve conflicts reliably [19–21]. Although these papers are not all framed as trajectory-prediction problems, they motivate the same requirement: intersection-specific structure matters.

5.4 Positioning of this thesis

This thesis builds directly on the interaction-aware formulation of Zhu *et al.* [1], but addresses its main limitation: the lack of explicit layout information. The first approach: **The Raster-Map Conditioning** (Section 8.1), explored in this work extends the baseline predictor with a learned embedding extracted from binary intersection images. This image-based approach is important because it provides a practical first step toward map conditioning while keeping the original graph-based predictor almost unchanged.

However, the image-based formulation also introduces an additional learning disadvantage: the network must infer geometric and topological properties from pixels, even though that information already exists explicitly in the SUMO network description. Motivated by the more structured map representations proposed in VectorNet and LaneGCN [11, 12], the second approach: **Explicit Layout-Feature Conditioning** (Section 8.2), explored in this thesis therefore moves toward a direct representation derived from the XML files of each intersection. Instead of asking the model to recover the layout from a binary image, the intersection can be described through explicit nodes, edges, lane widths, orientations, and connectivity relations.

For that reason, the two approaches should not be treated as unrelated alternatives but as two consecutive stages of the same research direction. The raster-image method constitutes the first attempt to make the predictor aware of the map, while the structured-layout method is a refinement inspired by prior map-aware forecasting literature and motivated by the need for stronger generalization. In this sense, the thesis studies both how to condition multi-agent prediction on intersection context and which form of layout representation is more suitable for that purpose.

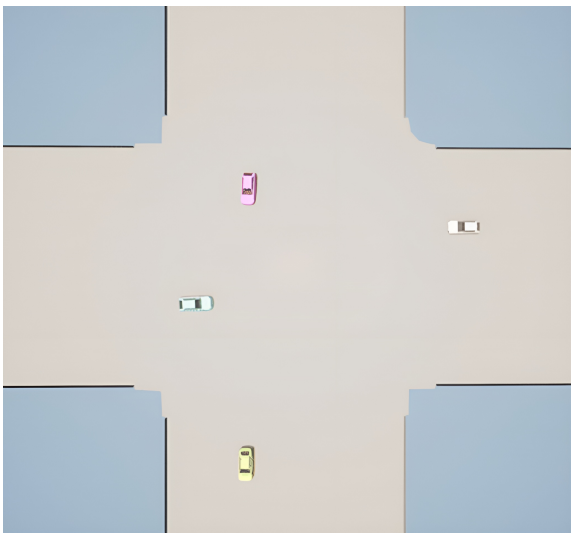

Fig. 1: CARLA simulator view for the standard four-way intersection. Several vehicles are spawned on the network and can be controlled with low-level actions such as steering and accelerating. Images from [1].

6 Experimental Environment

This project relies on a co-simulation between CARLA and SUMO. CARLA provides a high-fidelity 3D environment with realistic physics and rich urban maps, where vehicles can be controlled through low-level commands such as steering and acceleration. SUMO, in contrast, is a lightweight traffic simulator that defines road networks, priorities, and vehicle flows, and offers tools to design and modify intersection layouts. Figure 4. In the proposed setup, SUMO is used to specify and vary intersection geometries and traffic scenarios, while CARLA renders the same layouts and executes the learned controller in closed loop. An overview of the CARLA setup and the standard four-way intersection used in this work is shown in Figure 1.

The two simulators are connected through a synchronization bridge that keeps their clocks aligned and ensures that vehicles remain consistent in both worlds. At each simulation step, SUMO advances the traffic state on its road network, and the bridge propagates the corresponding positions and intentions to CARLA. The learned model then produces low-level control actions that are applied in CARLA and reflected back into SUMO. In this way, the same controller is evaluated in a physically consistent 3D environment while still benefiting from SUMO's flexible network and traffic configuration.

During early experiments, several typical failure modes appeared in this co-simulation. Some runs exhibited inter-vehicle collisions at the intersection center, while others showed unnecessary stops that created congestion or vehicles drifting outside the drivable area. These behaviors are illustrated in Figure 2 and motivated the need for improved data generation and layout-aware representations.

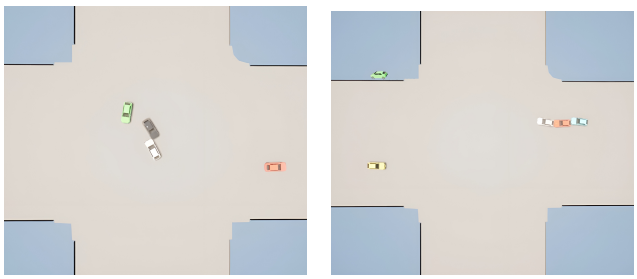

Fig. 2: Common failure modes in early co-simulation runs. (Left) collisions between vehicles at the intersection center. (Right) unnecessary stops that create congestion, along with a vehicle deviating off the road. Image from [1].

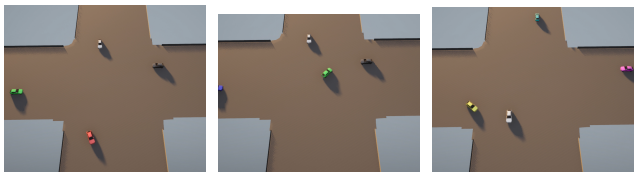

Fig. 3: CARLA online simulation with the baseline model on the standard intersection. Time-lapse from left to right shows: (left) four vehicles at the intersection with white and black waiting, (center) green turning left while white and black still yield, and (right) white proceeding straight as black has cleared the scene. The sequence demonstrates safe yielding, smooth turning, and no unnecessary stopping.

Training data for the predictor are generated in SUMO by simulating multiple intersection scenarios with different layouts and traffic patterns. The same layouts are then imported into CARLA so that the controller can be exercised online under matching geometry. Figure 3 shows an example in which the baseline model handles the standard intersection correctly, with safe yielding and smooth turning, while Figure 5 highlights cases where the model encounters off-distribution situations or fails to generalize to more complex layouts.

In summary, SUMO is used to define and vary intersection layouts and traffic flows, CARLA provides a realistic environment for closed-loop evaluation, and the bridge synchronizes both at a fixed time step so that the same learned model can be assessed consistently across a wide range of scenarios. Implementation details about software versions, scripts, and configuration files are provided in the appendix.

7 Data Generation

The dataset for this thesis is produced with SUMO as the source of small traffic and with CARLA as the target world for online evaluation. SUMO provides a road network, priority rules, and vehicle flows that is can controled precisely; CARLA then renders the same geometry and executes the learned model controller in closed loop. The goal of this chapter is to describe what the data looks like, how it is generated, how it is augmented, and how it is packaged for efficient training.

The data generation pipeline starts from a SUMO network file. This file is created using the SUMO Netedit tool, that allows the creation of intersections; and represents the intersection layout that contains lanes, connections, and priority roads. From this network, it is produced two kinds of inputs for a simulation: a route file and a configuration file. The **route file** defines which paths vehicles can take across the intersection, their type parameters, and when new vehicles spawn. In practice, it lists named routes such as *left_right*, *up_left*, and so on, and inserts vehicles with unique ids and departure times according to a creation probability and a straight/turn split. The **configuration file** references the network and the route file and records simulation options such as whether to ignore route errors, the rerouting device settings, logging verbosity, a GUI preset, and, crucially, which inputs to load. In short, the route file describes what traffic happens, and the configuration file tells SUMO how to run it on a specific network.



Fig. 4: SUMO simulation of the standard four-way intersection, used as the starting point for data generation and model evaluation.

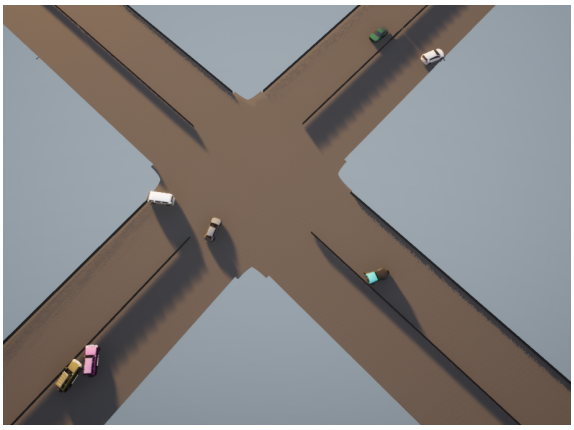

Fig. 5: Baseline performance on an unseen rotated cross (X-shaped) intersection. Multiple vehicles collide with the boundary walls, showing poor generalization to this layout.

With these files in place, we run SUMO to obtain *floating-car data* (FCD) for the whole simulation. The run is parameterized by a start time, an offset, a total duration, a simulation step length, and an overall traffic scale factor. The output is an time series where each time step contains all active vehicles with their ID, position, heading angle, and speed. Two extraction paths exist. In the older path, the FCD is first converted to OMNeT-style traces using SUMO's `traceExporter`, optionally restricted to a spatial boundary around the intersection center; the resulting files are then parsed into scene CSVs. In the newer path, we read the FCD directly and write CSV scenes without the intermediate OMNeT format. Both paths produce the same training CSV structure.

Each CSV encodes a short scene window with columns such as `TIMESTAMP`, `TRACK_ID`, `X`, `Y`, `yaw`, and `speed`. Scenes are built with a fixed window length, and a sliding window is used to create multiple, minimally overlapping samples. To ensure learnable supervision, only vehicles that are present for at least `OBS_LEN` + `PRED_LEN` steps are kept. A "current time" inside the window is chosen close to the end of the sequence, and only agents near the intersection center at that instant are marked as targets; others are ignored. This selection focuses the dataset on the interactions that matter for crossing and turning. Timestamps in the file name indicate the inclusive start and end of the scene, which simplifies traceability.

After CSV generation, simple data augmentation was applied to improve robustness. The intent is to teach the model to recover from small disturbances that often occur during closed-loop control. Augmentations include small spatial perturbations of positions, limited perturbations of heading angle and speed consistent with vehicle movement, and optional short model-predictive-control rollouts that create near-boundary recovery trajectories. The augmentation budget is controlled by an integer parameter that sets how many variants to generate per base scene; setting it to zero disables augmentation. All augmented samples inherit the same scene windowing and target-agent selection rules.

For training efficiency, the CSV scenes are then converted into serialized `pickle` files. Each pickle groups the tensors needed by the dataloader, for example agent states over the observation horizon, the prediction targets, and any metadata per scene, such as the sampling rate and scene length. Using `pickle` avoids repeated CSV parsing and reduces startup overhead, which is important when training with small batch sizes and frequent disk access. The final dataset is organized into `train` and `val` splits, each containing the pickled scenes and a parallel CSV copy for inspection.

To summarize, the data flow is: network file $\rightarrow$ routes file and configuration file $\rightarrow$ SUMO simulation with FCD output $\rightarrow$ scene extraction to CSV with target-agent filtering and sliding windows $\rightarrow$ augmentation to create near-miss and perturbed samples $\rightarrow$ conversion to `pickle` for fast loading. The resulting dataset captures the agent states and interactions most relevant to intersection negotiation and is ready to train intent-aware, interaction-aware predictors. In the next subsection, it is described how to generate additional intersection layouts to broaden the geometry and topology coverage.

7.1 Generating New Intersections and Layout-Aware Augmentation

To create data for new intersections, we start from the standard SUMO network file. This file is duplicated and the copy is open in `netedit`. Then the node positions, edge shapes, and lane widths are modified so that the layout looks meaningfully different. Typical edits include changing the inclination of approaches, adjusting curvature at corners, and widening or narrowing selected lanes to add variety. After each edit, let `netedit` rebuild connections and priorities, then save the network and pass it through the same pipeline as the original map to generate traffic data and augment it.

Automatic intersection generation. In the original pipeline, each new intersection layout was created manually in netedit, which limited how many distinct geometries could be explored. To increase diversity and reduce manual effort, a new script was developed that generates families of intersections automatically starting from the default layout. The script parses the base .net.xml file to extract the node and edge lists, applies random but constrained perturbations to their positions and shapes (for example, shifting nodes or rotating approaches), and then writes out a modified network description. The resulting node and edge sets are passed to netconvert, which regenerates a valid SUMO network with consistent lanes, priorities, and connections. In practice, this procedure can create dozens of geometrically distinct but structurally valid intersections in a few seconds, whereas manual design of a single layout previously required significant time.

Once a new network has been generated, the downstream pipeline remains unchanged. A route file and a configuration file are created for each layout, SUMO simulations are run to obtain floating-car data, scenes are extracted into CSV with the same sliding-window and target-agent criteria, and the dataset is converted to pickle for training. The only difference is that the space of intersection geometries is now populated automatically rather than by hand.

Once a new network is available, the downstream pipeline remains unchanged. A route file and a configuration file are generated for each layout, SUMO is run to obtain floating-car data, scenes are extracted into CSV files using the same sliding-window and target-agent criteria, and the dataset is converted to `pickle` for training. This makes it possible to populate the space of intersection geometries automatically rather than by hand.

In the first approach mentioned before, each intersection map is converted into a binary image that captures the drivable area. The conversion reads edges, connections, and corner geometry from the network file and rasterizes them at a fixed resolution in meters per pixel. The image occupies a fixed spatial extent around the intersection center so that the scale is consistent across maps. Examples of three binary maps are shown in Figure 6. A single-channel occupancy map is sufficient for training the representation, although multi-channel variants are possible, for example, centerlines in one channel and boundaries in another. The important part is that all maps share the same coordinate frame and resolution.

These raster maps are augmented to improve coverage before training an autoencoder. Horizontal and vertical flips are applied, rotations by 90° or 180°, and small rotations by a few degrees to simulate approach angle variation. Light Gaussian blur and low-amplitude noise are added to make the encoder robust to embedding differences. We avoid modifications that change topology since those would create roads that do not exist. Augmentations preserve metric scale and keep the intersection center aligned so the embedding remains comparable across maps.

The autoencoder learns a compact embedding of the intersection layout from these binary images. The encoder compresses the image into a vector $\mathbf{z}$, and the decoder reconstructs the original map. A reconstruction loss is used, such as binary cross-entropy or Dice loss. Once trained, one embedding per map is computed and supplied $\mathbf{z}$ to the

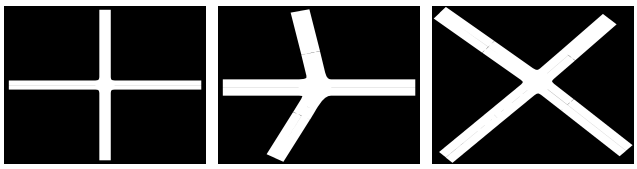

Fig. 6: Binary raster maps used to learn a layout embedding. The three panels show different intersection layouts: (left) standard intersection with finer streets, (center) intersection with two inclined roads, and (right) X-shaped intersection.

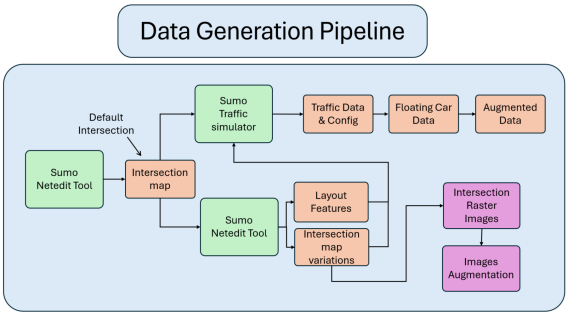

Fig. 7: Overview of the data generation pipeline. Starting from a default SUMO intersection created with `netedit`, new intersection layouts are generated by applying constrained geometric variations to the original map. Each layout is then used to produce route and configuration files for SUMO, which generate traffic data and floating-car data (FCD). In parallel, the same layouts are converted into explicit layout features and rasterized into binary images, which can later be augmented and used as input representations for the model.

trajectory predictor as side information. The embedding can enter as a global context vector that is broadcast to all agents, or it can modulate intermediate layers through simple conditioning, for example, feature concatenation. This gives the predictor explicit knowledge of the intersection structure and helps it adapt when the geometry at test time differs from the layouts used during training.

In summary, new intersections are created by editing the base network file in `netedit`, simulated with their own route and configuration files to produce CSV scenes, and then paired with a layout embedding learned from binary image maps. The layout augmentations enlarge coverage without breaking topology, and the learned embedding supplies compact structural information to the model during training and evaluation.

The second approach leverages the full structural information already provided by SUMO, requiring minimal feature engineering while making the intersection geometry explicit. In this approach, the nodes, edges, lane widths, and geometric dimensions of a given intersection are extracted by parsing the corresponding network file. Since SUMO can reconstruct an intersection map from nodes, edges, connections, orientations, and distances, new layouts can be generated by applying random but constrained perturbations to these elements. This produces a diverse set of valid intersection maps, which can then follow the same data-generation pipeline to obtain traffic data and floating-car data (FCD). The complete data generation pipeline, including map rasterization and traffic data augmentation is illustrated in Figure 7.

8 Map-Conditioned Multitrajectory Prediction Model

This chapter describes the two layout-aware extensions explored in this thesis. Both approaches start from the same baseline multi-agent predictor of Zhu *et al.* [1], which models vehicle interactions through message passing over a graph of agents. The difference lies in how the intersection layout is represented and injected into the model.

The **first approach** conditions the predictor on a binary raster image of the intersection. The image is processed by a convolutional encoder to obtain a compact layout embedding, which is then concatenated to the input features of every agent. This design makes the model layout-aware while preserving the original interaction module.

The **second approach** uses the structural information already available in the SUMO network files. Instead of learning a layout representation from images, the model receives an explicit vector built from nodes, edges, widths, orientations, and other geometric attributes extracted from the network file. In this case, the predictor architecture remains unchanged; only the input dimensionality increases because the layout vector is concatenated directly to the same per-agent features used in the baseline.

Inputs and outputs. For each agent i at time t, we define the state vector

$$\mathbf{x}_t^i = \left[x_t^i,\ y_t^i,\ \psi_t^i,\ v_t^i,\ \mathbf{g}^i\right],$$

where (x_t^i, y_t^i) are the coordinates in the intersection reference frame, ψ_t^i is the heading angle, v_t^i is the speed, and $\mathbf{g}^i$ is a one-hot encoding of the intended maneuver (go straight, turn left, or turn right). Given the observed scene, the model predicts a future horizon of $H = 30$ positions for each target agent:

$$\hat{\tau}_{t+1:t+H}^i = \left[\hat{x}_{t+1}^i, \hat{y}_{t+1}^i, \ldots, \hat{x}_{t+H}^i, \hat{y}_{t+H}^i\right].$$

As in the baseline, interactions between vehicles are handled by a message-passing module operating on the agent graph. The proposed extensions do not modify the interaction mechanism itself; they only augment the input with information about the layout of the current intersection.

8.1 First approach: Raster-Map Conditioning

The first approach provides the model with an explicit summary of the intersection through a binary image of the map. Each intersection is converted into a single-channel raster that marks the drivable area within a fixed metric window and resolution. A convolutional encoder $E(\cdot)$ maps this image to a layout embedding

$$\mathbf{z} = E(\mathbf{M}), \qquad \mathbf{z} \in \mathbb{R}^d,$$

where $\mathbf{M}$ denotes the binary map image and d is the embedding dimension.

In our implementation, $E(\cdot)$ is based on a ResNet-18 backbone [9] pretrained on ImageNet [10]. Since the number of available intersection images is limited, training the

encoder from scratch is not practical. For that reason, the encoder is either frozen or only partially fine-tuned, for example, by updating the last residual block while keeping the remaining layers fixed. When the encoder is frozen, the embedding $\mathbf{z}$ is computed once per map and cached.

The resulting layout vector is broadcast to all agents in the scene and concatenated to the original state input:

$$\tilde{\mathbf{x}}_t^i = \left[\mathbf{x}_t^i,\ \mathbf{z}\right].$$

Therefore, every agent is processed together with a shared description of the intersection in which it is currently driving. This allows the predictor to adapt its outputs to different geometries while keeping the same graph-based interaction structure.

The downstream predictor itself remains almost identical to the baseline. It contains an encoder for the per-agent state, a message-passing layer to exchange information across vehicles, and a decoder that outputs the future trajectory. The only architectural change is that the input dimension increases from $\dim(\mathbf{x}_t^i)$ to $\dim(\mathbf{x}_t^i) + d$. In other words, the first approach introduces a new encoder branch for the raster map, but leaves the trajectory predictor unchanged apart from the concatenated conditioning vector.

8.1.1 Coordinate and scale alignment

For the raster-based approach to work consistently across layouts, the agent coordinates and the map image must share the same origin, orientation, and scale. We place the center of the intersection at the origin of the reference frame, align the vertical axis of the raster with the map orientation used in SUMO, and use a fixed metric resolution in meters per pixel. All scenes, both during training and evaluation, are represented in this common frame.

This alignment is important because the learned embedding $\mathbf{z}$ should encode structural differences between layouts, not arbitrary differences caused by inconsistent map orientation or scale. Without this normalization, the same intersection geometry could appear as different layouts to the encoder.

8.1.2 Training raster-map conditioning

When using raster-map conditioning, training is performed with scenes coming from multiple intersections, and several layouts are held out for validation. Each mini-batch intentionally mixes samples from different maps so that the predictor learns to interpret different layout embeddings in parallel. This forces the model to associate the same maneuver intention with different local geometries.

If the map encoder is frozen, the cached embeddings are loaded during training and the additional computational cost is small. If the encoder is fine-tuned, a smaller learning rate is used for the convolutional backbone and a larger one for the trajectory predictor. Early stopping is based on validation ADE.

Figure 8 summarizes this first approach. Each training sample contains a vehicle interaction graph together with the binary map image of the corresponding intersection. During the forward pass, the image is encoded into a layout vector, which is then concatenated to the node features before message passing and trajectory decoding.

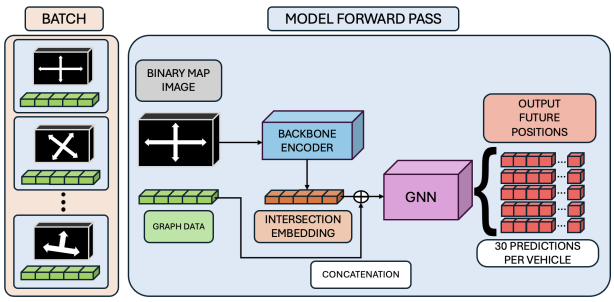

Fig. 8: Pipeline of the first approach model forward pass. Each scene provides a vehicle interaction graph and a binary map image identifying the intersection layout. The image is processed by a pretrained convolutional encoder to obtain a layout embedding, which is concatenated to the graph node features before the GNN predicts the next 30 positions for each vehicle.

8.2 Second approach: Explicit Layout-Feature Conditioning

The second approach simplifies the previous design by removing the image representation and the map encoder altogether. Instead of asking the network to infer geometry from a rasterized map, the layout is described directly through structured features extracted from the SUMO `.net.xml` file.

Let

$$\mathbf{m} \in \mathbb{R}^p$$

denote the vector of explicit layout features for a given intersection, where $\mathbf{m}$ may include quantities derived from nodes, edges, lane widths, approach orientations, distances, connectivity information, or aggregated geometric descriptors. This vector is shared by all agents belonging to the same scene and concatenated directly to the original state vector:

$$\bar{\mathbf{x}}_t^i = \left[\mathbf{x}_t^i,\ \mathbf{m}\right].$$

In this second approach, the predictor receives exactly the same agent information as in the baseline and in the first approach: position, heading, speed, and intention. The only difference is that the layout description is no longer obtained from a binary image through an encoder. Instead, it is already available as a structured feature vector and is appended directly to the input. As a result, the model architecture remains unchanged: there is no convolutional backbone, no raster map branch, and no learned visual embedding stage. See figure 9 for a better understanding.

From an architectural point of view, this means that the second approach is even simpler than the first one. If the baseline predictor operates on inputs of dimension $\dim(\mathbf{x}_t^i)$, then the new input dimension becomes

$$\dim(\bar{\mathbf{x}}_t^i) = \dim(\mathbf{x}_t^i) + p.$$

All subsequent components of the network remain exactly the same: the per-agent encoder, the message-passing layers, and the trajectory decoder are preserved without modification.

This design has two potential advantages. First, it reduces complexity by eliminating the need to train or fine-tune a separate encoder for map images. Second, it gives the

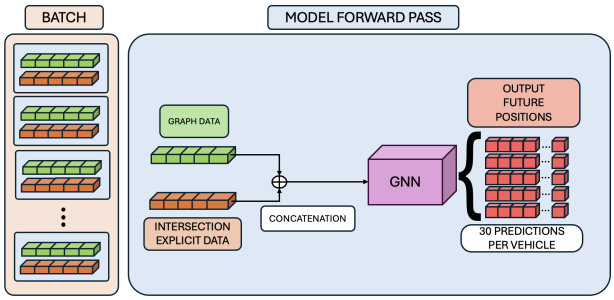

Fig. 9: Pipeline of the second approach model forward pass. Each scene provides a vehicle interaction graph and the corresponding explicit intersection geometry data, which is concatenated to the graph node features before GNN predicts the next 30 positions for each vehicle, like the first approach.

model direct access to geometric information that SUMO already provides explicitly, rather than forcing the network to recover it indirectly from pixels. In this sense, the second approach can be understood as a more structured and lightweight alternative to the first one.

8.3 Comparison of both approaches

The two approaches share the same prediction target, the same graph-based interaction mechanism, and the same training scenes. Their main difference is the representation of the intersection layout:

$$\text{First approach: } \tilde{\mathbf{x}}_t^i = [\mathbf{x}_t^i, \mathbf{z}],$$

$$\text{Second approach: } \bar{\mathbf{x}}_t^i = [\mathbf{x}_t^i, \mathbf{m}].$$

In the first case, $\mathbf{z}$ is a learned embedding obtained from a binary raster image through a convolutional encoder. In the second case, $\mathbf{m}$ is an explicit vector of geometric features extracted directly from the SUMO network description. The first approach is more flexible and can, in principle, learn rich visual cues from the layout, whereas the second approach is simpler and more direct, since it uses structured information that is already available.

For this reason, the thesis studies both variants as two consecutive steps in the same research direction. The first approach explores whether a learned image-based embedding can help the predictor generalize to unseen intersections. The second approach investigates whether the same goal can be achieved more efficiently by preserving the original predictor and only extending its input with explicit layout features.

8.4 Loss Function

The loss function used to train the model is composed of an imitation loss, a collision loss, and a weighted position loss over the prediction horizon.

The imitation loss measures the average L_2 distance between the predicted future trajectory and the ground truth:

$$\mathcal{L}_{\text{imitation}} = \frac{1}{N} \sum_{k=1}^{N} \sum_{t=1}^{T} \left\| \mathbf{S}_k^t - \hat{\mathbf{S}}_k^t \right\|_2,$$

where $\mathbf{S}_k^t$ and $\hat{\mathbf{S}}_k^t$ denote the ground-truth and predicted state of vehicle k at time step t, respectively.

The collision loss penalizes pairs of vehicles whose predicted trajectories come closer than a safety threshold λ. For each pair (i, j), the collision term is defined as

$$\mathcal{L}_{\text{collision},i,j} = \begin{cases} 0 & \text{if } \min_t \left\| \mathbf{S}_i^t - \mathbf{S}_j^t \right\|_2 > \lambda, \\ \lambda - \min_t \left\| \mathbf{S}_i^t - \mathbf{S}_j^t \right\|_2 & \text{otherwise.} \end{cases}$$

The total collision loss is then obtained by summing over all interacting pairs:

$$\mathcal{L}_{\text{collision}} = \sum_{i,j} \mathcal{L}_{\text{collision},i,j}.$$

This term discourages vehicle-to-vehicle conflicts at intersections and helps the model learn safer predictions.

To further improve short-term accuracy and closed-loop stability, we also use a weighted mean-squared error over the predicted positions:

$$\mathcal{L}_{\text{pos}} = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{H} w_t \left\| \hat{\mathbf{p}}_t^i - \mathbf{p}_t^i \right\|_2^2,$$

where $\mathbf{p}_t^i = (x_t^i, y_t^i)$ is the position of vehicle i at time step t. Early time steps are weighted more strongly, with $w_1 = 25$, $w_{2..5} = 5$, and $w_{6..H} = 1$, to emphasize short-horizon precision.

The final training objective combines these terms:

$$\mathcal{L} = \mathcal{L}_{\text{imitation}} + \alpha \, \mathcal{L}_{\text{collision}} + \beta \, \mathcal{L}_{\text{pos}},$$

where α and β are weighting coefficients chosen empirically.

8.5 Evaluation Metrics

We report four metrics during validation and in `evaluate()`:

- *ADE* (Average Displacement Error). Mean Euclidean distance over all prediction steps:

$$\text{ADE} = \frac{1}{NH} \sum_{i=1}^{N} \sum_{t=1}^{H} \left\| \hat{\mathbf{p}}_t^i - \mathbf{p}_t^i \right\|_2.$$

  Lower is better. It measures overall trajectory quality.

- *FDE* (Final Displacement Error). Distance at the last prediction step:

$$\text{FDE} = \frac{1}{N} \sum_{i=1}^{N} \left\| \hat{\mathbf{p}}_H^i - \mathbf{p}_H^i \right\|_2.$$

  Lower is better. It measures long-term accuracy.

- *MR* (Miss Rate). Fraction of trajectories with FDE $> 4\,\text{m}$:

$$\text{MR} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1} \left(\left\| \hat{\mathbf{p}}_H^i - \mathbf{p}_H^i \right\|_2 > 4\,\text{m} \right).$$

  Lower is better. It captures outright failures.

- *CR* (Collision Rate). For each pair of interacting agents, we check whether their predicted trajectories ever come closer than $2\,\mathrm{m}$:

$$\mathrm{CR} = \frac{1}{|\mathcal{E}|} \sum_{(u,v)\in\mathcal{E}} \mathbf{1}\left(\min_{t\in\{1,\dots,H\}} \left\| \hat{\mathbf{p}}_t^u - \hat{\mathbf{p}}_t^v \right\|_2 < 2\,\mathrm{m} \right),$$

  where $\mathcal{E}$ is the set of evaluated interacting pairs. Lower is better. It measures whether predicted paths would lead to risky overlaps.

8.6 Evaluation Protocol

After training, both approaches were evaluated both quantitatively and qualitatively. Quantitative evaluation is performed by running the predictor on the validation and test splits and computing the metrics above. Qualitative evaluation is performed in CARLA, where the learned controller is executed in closed loop and the resulting vehicle behavior can be inspected directly.

For the first approach, the corresponding map embedding $\mathbf{z}$ is computed from the binary raster image of the current intersection and concatenated to the agent inputs. For the second approach, the explicit layout vector is concatenated directly to the same per-agent inputs, without using an image encoder. In both cases, the model is tested on the standard intersection and on unseen layouts in order to assess whether the layout conditioning improves generalization.

Summary: Both approaches share the same training objective and evaluation metrics. The only difference lies in how the intersection layout is provided to the model: through a learned embedding in the first approach, and through explicit structured features in the second. ADE, FDE, MR, and CR are used to compare the models, while CARLA simulations provide a qualitative check on closed-loop behavior.

9 Results

In this section, we present the results of the experiments for the two layout-aware approaches proposed in this thesis. The first approach conditions the predictor on binary raster maps through an image encoder, whereas the second approach uses explicit layout features extracted directly from the SUMO network files. In both cases, the models are compared against the baseline under two scenarios: (1) seen map with unseen traffic data, and (2) unseen map with unseen traffic data.

In this context, *seen* refers to whether the model has been trained on a particular map or traffic distribution. A map is considered seen if the model has seen traffic data generated from that layout during training. However, the same map may still be evaluated under a different traffic distribution that was not observed during training.

9.1 Qualitative Results

9.1.1 First approach: raster-map conditioning

Seen map, unseen traffic data

When the map is seen but the traffic data changes, the baseline model still performs well and maintains good prediction accuracy. The raster-map conditioned model also performs well, although it shows a slight degradation with respect to the baseline in some turning maneuvers. Overall, the differences remain small and the trajectories are still coherent.

Unseen Map

When the map and, therefore, the traffic data are unseen, both models perform the worst. The baseline model fails significantly. Unfortunately, the raster-map model struggles a lot with unseen maps as well, but it can be noticed that some vehicles can cross the intersection, avoiding crashing with other vehicles, and some other vehicles simply take more time before they collide with other vehicles or walls. This behavior is consistent with the quantitative results reported below.

9.1.2 Second approach: explicit layout-feature conditioning

Seen map, unseen traffic data

For seen maps with unseen traffic data, the explicit layout-feature model performs slightly worse than the baseline and the raster-map conditioned model. The predictions remain usable, but the turning behavior is less precise and the trajectories are not as smooth as in the first approach.

Unseen map

For unseen maps, the explicit layout-feature approach behaves similarly to the raster-map conditioned model, although it is slightly weaker in some cases. Even so, it still helps reduce the severity of some failure modes, especially by avoiding more collisions than the baseline and raster-map approach. In other words, while the second approach does not clearly outperform the first one, it shows that explicit layout information can still support more stable behavior on unseen intersections.

9.2 Quantitative Results

The quantitative evaluation uses the Average Displacement Error (ADE), Final Displacement Error (FDE), Miss Rate (MR), and Collision Rate (CR). Lower values are better for all metrics.

Table 1 summarizes the quantitative results for the baseline model, the raster-map conditioned model, and the explicit layout-feature model.

Scenario	Model	ADE	FDE	MR	CR
seen map, unseen traffic	Baseline	1.10	2.13	0.16	0.08
	Raster-Map	1.19	2.22	0.19	0.10
	Layout Features	3.89	6.43	0.40	0.20
unseen map	Baseline	3.21	4.21	0.85	0.65
	Raster-Map	3.17	4.16	0.82	0.63
	Layout Features	2.70	5.93	0.90	0.19

TABLE 1: Quantitative evaluation results of the baseline model, the raster-map conditioned model, and the explicit layout-feature model under different scenarios. ADE and FDE are reported in meters, MR is the miss rate, and CR is the collision rate. Lower metric values are better.

The results confirm the qualitative observations. On seen maps, the raster-map conditioned model stays close to the baseline but does not clearly improve it, while the explicit layout-feature model performs worse in all metrics. On unseen maps, the raster-map conditioned model performs slightly better than the baseline, and the explicit layout-feature model shows similar overall behavior, with a lower collision rate but weaker displacement accuracy. This suggests that explicit layout features may help reduce collisions in difficult unseen scenarios, even if they do not consistently improve trajectory accuracy.

10 Discussion

The results suggest that both layout-aware approaches capture useful information about the intersection, but their performance is limited by the amount of training data available. The raster-map conditioned model is the most stable overall, especially on unknown maps, where it slightly improves over the baseline. The explicit layout-feature model is less accurate on known maps, but it can still reduce collisions in some unseen layouts.

A likely explanation is that both approaches try to learn a rich representation of intersection geometry from a relatively small dataset. The raster-based model must interpret layouts through a convolutional encoder, while the explicit-feature model relies on structured geometric descriptors that may not fully capture the complexity of the map. In both cases, the model would likely benefit from a larger and more diverse training corpus.

This limitation is especially relevant here because data generation was slow and expensive. Producing layouts, generating traffic, and preparing the training data took many days, even with only one augmentation setting, and the full dataset covered only around 50 intersections. A larger and longer collection process with stronger computational resources would probably improve generalization and make the learned representations more reliable.

Another practical limitation was the instability of the virtual machine environment. During long data-generation runs, the process sometimes failed for reasons that were not always identifiable, which caused several days of data collection to be lost. A more robust infrastructure would reduce interruptions and allow a broader range of layouts and traffic patterns.

Overall, the experiments show that adding layout information is beneficial, but not sufficient to solve generalization completely. The raster-based approach remains the stronger of the two in terms of consistency, while the explicit layout-feature approach is promising because it can reduce collisions in some unseen scenarios without requiring an image encoder.

11 Future Work

Although the proposed approaches already show that layout information can be incorporated into multi-vehicle trajectory prediction, several directions remain open for improving performance, robustness, and scalability.

- **Increase training data diversity.** The most direct way to improve generalization is to expand the dataset with a larger and more varied collection of intersection layouts and traffic scenarios. In particular, the layout-generation process could be further automated so that many valid `.xml` network files are produced without manual intervention. This would make it possible to generate traffic data at a much larger scale and expose the model to a broader range of geometries, lane widths, approach angles, and priority configurations.

- **Improve augmentation strategies.** The current augmentation scheme could be extended with more realistic perturbations that preserve topology while increasing variability. This may include richer traffic-flow variations, small geometric distortions, controlled changes in vehicle density, or simulation-based perturbations that emulate noisy closed-loop behavior. A more diverse augmentation pipeline could help the model become more robust, especially under unseen traffic conditions.

- **Use more expressive layout encoders.** The raster-based approach relies on a pretrained image backbone, while the explicit-feature approach compresses the layout into a structured vector. Future work could explore encoders that are more specialized for road geometry, such as graph neural networks over lane connectivity, hybrid raster-graph encoders, or encoder-decoder architectures designed for topological scene understanding. These alternatives may preserve more useful spatial information than a generic visual backbone.

- **Test alternative training objectives.** The current loss already balances imitation and collision avoidance, but future work could investigate additional terms that explicitly encourage lane adherence, comfort, temporal consistency, or better interaction reasoning. A more refined objective may improve closed-loop behavior and reduce the cases where the model predicts trajectories that are accurate in open loop but unstable in simulation.

- **Scale up the evaluation protocol.** Future experiments should validate the methods on a larger set of intersection types, traffic densities, and unseen map configurations. It would also be useful to report ablations that isolate the effect of each representation choice, backbone selection, and augmentation setting. This would make it easier to identify which component contributes most to generalization.

12 Conclusions

In this thesis, we proposed and evaluated two layout-aware extensions to a multi-vehicle trajectory prediction model for intersections. The first approach uses a pretrained image encoder to transform binary raster maps into layout embeddings, while the second approach replaces the image branch with explicit layout features extracted directly from the SUMO network files. Both approaches were tested against the baseline under seen and unseen map conditions.

The raster-map conditioned model provides the most consistent behavior overall, especially on unseen maps,

where it slightly improves over the baseline. The explicit layout-feature model is less accurate on seen maps, but on unseen maps it performs comparably to the raster-based approach and can reduce collisions in some cases. This suggests that explicit geometric information is useful, even if the current representation is still too simple to outperform the raster-based version consistently.

A key takeaway is that generalization to unseen intersections is strongly limited by data availability. The dataset was expensive to generate, relatively small, and occasionally affected by failures in the virtual machine environment, which reduced the usable data and slowed experimentation. With a larger and more reliable dataset, both approaches would likely improve substantially.

Despite these limitations, the thesis contributes a complete pipeline for generating intersection scenarios, training layout-aware predictors, and evaluating them quantitatively and qualitatively in CARLA. It also shows that intersection structure can be incorporated either through raster embeddings or through direct geometric features, providing a solid basis for future work on larger datasets and more expressive layout encodings.

REFERENCES

[1] D. Zhu, Q. Khan, and D. Cremers, "Multi-vehicle trajectory prediction at intersections using state and intention information," in *arXiv preprint arXiv:2301.02561*, 2023.

[2] X. Li *et al.*, "Grip: Graph-based interaction-aware trajectory prediction," in *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019.

[3] T. Zhao *et al.*, "Multi-agent tensor fusion for contextual trajectory prediction," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[4] J. Li *et al.*, "Interaction-aware multi-agent tracking and probabilistic behavior prediction via adversarial learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[5] H. Zhou *et al.*, "Ast-gnn: An attention-based spatio-temporal graph neural network for trajectory prediction," *Neurocomputing*, vol. 445, pp. 298–308, 2021.

[6] H. Tang, P. Wei, J. Li, and N. Zheng, "Evost-gat: Evolutionary spatio-temporal graph attention network for trajectory prediction," *Neurocomputing*, vol. 491, pp. 333–342, 2022.

[7] Y. Peng *et al.*, "Srai-lstm: Social relationship attention and intention lstm for trajectory prediction," *Neurocomputing*, vol. 490, pp. 258–268, 2022.

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

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[11] J. Gao, C. Sun, H. Zhao, and Y. Shen, "Vectornet: Encoding hd maps and agent dynamics from vectorized representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[12] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, J. Feng, and R. Urtasun, "Learning lane graph representations for motion forecasting," in *Proceedings of the European Conference on Computer Vision*, 2020.

[13] H. Ma, Y. Sun, J. Li, and M. Tomizuka, "Driving behavior prediction across scenarios with self-supervised domain knowledge," in *IEEE Intelligent Transportation Systems Conference (ITSC)*, 2021.

[14] J. Bock *et al.*, "The ind dataset: A drone dataset of naturalistic road user trajectories at german intersections," *IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[15] V. Sezer, T. Bandyopadhyay, D. Rus, E. Frazzoli, and D. Hsu, "Autonomous navigation of unsignalized intersections under uncertainty of human driver intent," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[16] Z. Wang *et al.*, "Decision-making at urban intersections based on conflict resolution," *Journal of Advanced Transportation*, 2021.

[17] Q. Ge *et al.*, "Real-time coordination of connected vehicles at intersections using graphical mixed-integer optimization," *IET Intelligent Transport Systems*, vol. 15, no. 6, pp. 795–807, 2021.

[18] H. M. Wang, S. S. Avedisov, O. Altintas, and G. Orosz, "Multi-vehicle conflict management with status and intent sharing under time delays," *IEEE Transactions on Intelligent Vehicles*, 2022.

[19] I. Llatser *et al.*, "Cooperative automated driving use cases for 5g v2x communication," in *IEEE 5G World Forum*, 2019.

[20] R. Deng, B. Di, and L. Song, "Cooperative collision avoidance for overtaking maneuvers in cellular v2x-based autonomous driving," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 4434–4446, 2019.

[21] M. Houmer, M. Ouaissa, and M. Ouaissa, "Secure authentication scheme for 5g-based v2x communications," *Procedia Computer Science*, vol. 198, pp. 276–281, 2022.

A Project Setup - Implementation Details and Reproduction

This project uses a CARLA–SUMO co-simulation. CARLA is a high-fidelity 3D simulator built on Unreal Engine that provides realistic physics, rich city maps, and a Python API to spawn, sense, and control vehicles. It accepts low-level commands such as steering and acceleration, which lets us test closed-loop behavior in a physically consistent world. SUMO is a small traffic simulator that defines road networks, priorities, and flows. It also offers tools to build networks and to export them to OpenDRIVE. In our pipeline, SUMO supplies controllable intersection scenarios while CARLA renders and executes them for online evaluation. An overview of the CARLA setup and the standard intersection used in this project is shown in Figure 1.

A.1 Software and packages

The project expects CARLA and SUMO to be installed locally. The README pins specific versions of both simulators and recommends checking that CARLA launches to a default city map and that the SUMO GUI starts correctly. For Python, the environment targets Python 3.7 and uses PyTorch, TorchVision, Torchaudio, PyTorch Geometric (PyG), and common scientific packages (matplotlib, `cvxpy`, `lxml`, `pandas`). Version pinning is used to avoid breaking changes when training and running the bridge.

A.2 Repository layout inside CARLA

The co-simulation builds on the official CARLA–SUMO bridge and replaces some of CARLA's scripts. The README shows where to copy the repository's folders into the CARLA tree and which items are new, which are replacements, and which originals remain. This ensures the bridge, custom map assets, and control scripts are found at the expected paths when you run experiments.

A.3 Map activation in CARLA

The project uses an OpenDRIVE intersection map. You activate it with CARLA's *config* utility by pointing to the provided `.xodr` file. After activation, open CARLA and confirm that the intersection is loaded. This sets the 3D world that mirrors the SUMO network you will simulate.

A.4 SUMO tools and environment variable

The bridge relies on SUMO's tools (TraCI, `netconvert`, `netedit`). The environment variable `SUMO_HOME` must point to the SUMO installation directory so that Python can import `traci`. If you see an *ImportError* complaining that `traci` is missing, the path is incorrect and needs to be fixed. Do not try to import traci using `pip install traci` because you will be trying to use a different library also called traci from Python that will raise errors.

A.5 Running the CARLA–SUMO co-simulation

The entry point to online evaluation is a synchronization script that connects CARLA and SUMO. It requires (i) a SUMO configuration file (`.sumocfg`) for the scenario, (ii) a traffic-light manager setting that delegates signals to CARLA, (iii) a time step length that sets the co-simulation tick, (iv) an optional SUMO GUI flag if you want to visualize the simulation in the SUMO UI, and (v) a path to a pretrained model if you want to evaluate learned controllers. When launched with these parameters, vehicles appear in both simulators and move in lockstep. Typical failure cases observed in early tests are collisions, unnecessary full stops that cause congestion, and vehicles drifting outside the drivable area. These behaviors are illustrated in Figure 2.

A.6 Generating training data in SUMO

The repository includes a script to export trajectories from SUMO. You specify the sequence duration in seconds and the split to generate, training or validation. The output is a set of CSV files with states over time per agent. If you want to design your own networks, SUMO's `netedit` lets you create or edit `.xml` maps. You can then convert them to OpenDRIVE `.xodr` with `netconvert` so that CARLA can load the same geometry. A well-configured model in CARLA can perform strongly on the standard intersection, as shown in Figure 3. By contrast, on an unseen intersection map, for example, a rotated cross layout, the baseline fails more often, with boundary impacts and congested traffic (Figure 5).

A.7 Preprocessing and augmentation

A preprocessing script converts CSV sequences into pickled datasets and can add MPC-based augmentation. You pass the input CSV folder, the output PKL folder, and the number of augmented samples per sequence. Setting the augmentation count to zero disables this step. These augmentations are alternative paths that are created by adding some noise to the positions of the vehicles to generate more paths from them using SUMO so that the model is robust to perturbations in the trajectories. The goal is to expose the learner to unexpected states that occur during closed-loop control.

A.8 Training

The training script expects paths to the training and validation folders, along with the number of epochs, an experiment identifier, and a batch size. Trained weights are written to a folder named after the experiment ID. These weights can be supplied to the co-simulation for online evaluation on CARLA using the same intersection geometry.

Summary: In short, SUMO defines the network, traffic, and exported trajectories for learning. CARLA renders the same geometry, accepts low-level control, and hosts the online tests. The bridge synchronizes both at a fixed time step and passes control decisions to vehicles, which lets us evaluate prediction and control in realistic conditions using the same learned model.

B Reproducibility and transparency

The implementation and supplementary resources for this work, including the code and data, are available on the project's public GitHub repository: https://github.com/Dani13vg/TFG. This repository provides all necessary materials for reproducing the experiments and extending the research.

C Planning - Gantt Chart visualization

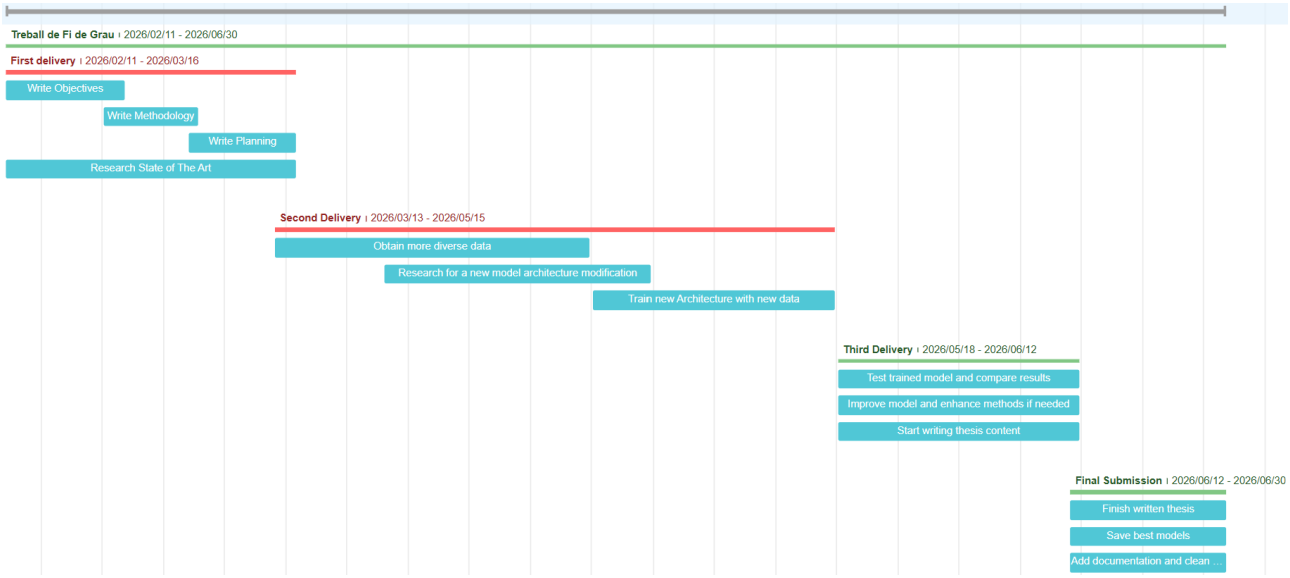

Fig. 10: Gantt chart of the thesis planning, showing the four main stages, intermediate deliveries, and final submission.