POLITECNICO DI TORINO

POLITECNICO DI TORINO

Master Degree course in Data Science and Engineering

Master Degree Thesis

Generative Models for Vehicle Trajectory Synthesis

Supervisors
Prof. Claudio Ettore CASETTI
Giuseppe PERRONE

Candidate
Giovanni GROSSI

ACADEMIC YEAR 2025-2026

Acknowledgements

Grazie a tutti!

Abstract

The data used in validating autonomous vehicles is generally limited to real world datasets which cover just few traffic patterns. For example, only about 30 percent of the trajectories found in the Waymo Motion Dataset have durations exceeding 10 seconds. Thus, this thesis aims to use deep generative models to synthesize vehicle trajectories as they pass through a roundabout, given their entry and exit points. In this thesis two generative models were implemented and tested on several SUMO simulated vehicle trajectories in a Roundabout in Milan. A Generative Adversarial Network (GAN), and an autoregressive Transformer based model that is able to make multimodal predictions and generate motion. The results obtained in the case of a single vehicle demonstrated that the GAN suffered from training instability and did not easily adapt to the trajectory generation problem; the Transformer exhibited more stable behavior during training and produced trajectories which satisfied the physical constraints to a higher degree than those generated by the GAN. Based on these findings, the Transformer architecture was extended to multi-vehicle scenarios. Preliminary experiments suggest that deep generative models can be effectively employed to synthesize vehicular trajectories and traffic scenarios suitable for testing and validating autonomous driving systems. The results highlight clear advantages of transformer-based approaches for roundabout trajectory generation, while also emphasizing the intrinsic challenges of modeling stable and coordinated multi-agent interactions. In particular, generating diverse, physically consistent, and socially coherent behaviors among interacting vehicles remains an open and demanding research problem.

Contents

Chapter 1

Introduction

Motivation

The rapid development of Autonomous Driving (AD) systems is expected to dramatically increase the safety, efficiency and accessibility of transportation systems. In order to deploy AD systems safely in real-world environments, extensive testing is required to validate their performance across a wide range of traffic scenarios. Simulation environments are used to evaluate the safety and reliability of AD systems in controlled, yet representative traffic environments. One major challenge with simulation-based testing of AD systems is the creation of realistic and diverse traffic scenarios that reflect the complexity of real world driving situations. Both traditional and current approaches to traffic simulation use pre-recorded data from real-world traffic scenarios or hand-crafted rules and heuristics to generate traffic scenarios. Real-world traffic data has many limitations when used to generate traffic scenarios; one limitation is that most real-world traffic datasets, e.g., Waymo Open Motion Dataset [6], Argoverse [3] and nuScenes [2], contain a large number of incomplete or fragmented vehicle trajectories because of sensor occlusion and limited observational periods.Therefore, scenarios generated from existing real-world data do not provide enough coverage for comprehensive evaluations of AD systems. Alternative approaches for expanding scenario coverage only rely on heuristic methods such as procedural generation and manual design of test cases. Tools like CARLA [5] and SUMO [15] allow users to create controlled testing environments for simulations, but their default traffic generation uses hand-crafted behavioral models that cannot capture the subtle complexities inherent in real world traffic behavior. The sim-to-real gap remains a major challenge in developing effective testing and training environments for autonomous vehicles. Recent upgrades in deep generative modeling may be able to solve the above challenges. Deep learning based methods can learn complex distributions of trajectories from data, enabling the generation of realistic, diverse, and complete vehicle trajectories without relying on explicit programming of rules. Some examples of successful applications of deep learning to model sequential data include Generative Adversarial Networks (GANs) [8] and Transformer-based architectures [19], which have shown great success in modeling sequential data in multiple areas such as natural language processing and time-series forecasting [20].

Problem and Scope

This dissertation is a major step in generating traffic scenes (in terms of vehicle trajectories) that are realistic at aroundabouts, based on provided entry/exit points. Roundabouts create challenging traffic scenes because they have:

- **Geometric Complexity**: Multiple lanes entering/exiting a roundabout with a circular roadway shape; and the need for vehicles to travel along curved roads while maintaining lane discipline.
- **Physical Constraints**: The created trajectories must be consistent with all road boundaries and also adhere to the physical constraints of both the entry/exit boundary conditions.
- **Temporal Coherence**: The trajectory must be consistent temporally for extended periods of time.

The primary research question investigated in this thesis is: can deep generative models create realistic, diverse, and physically plausible single vehicle trajectories over roundabouts with specified entry and exit constraints?

In order to answer this research question, we conducted a comparative evaluation of two fundamentally different generative architectures:

1. **Generative Adversarial Networks (GANs)**: a paradigm that utilizes adversarial learning to train a generator network to produce trajectories that cannot be distinguished from real trajectories [8, 9].
2. **Autoregressive Transformers**: a likelihood-based approach that generates trajectories. The model generated the full trajectory in a single forward pass. Enforcing temporal consistency through casual masking [7, 19]

Both paradigms were designed to generate complete, fixed length trajectory sequences $\mathbf{X} \in \mathbb{R}^{T\times 4}$ where $T = 120$, where each position encoded spatial coordinates (x, y), speed v, and heading angle θ. Both models are tasked with generating trajectories that have given start and ending points $\mathbf{S} = [x_{\text{start}}, y_{\text{start}}, x_{\text{end}}, y_{\text{end}}]$.

We chose to limit the scope of our research to single-vehicle trajectory creation without specifically modeling the impacts of multi-agent behavior. Limiting the research to single-vehicle trajectory creation allowed us to concentrate and thoroughly evaluate the fundamental challenges of trajectory synthesis, i.e. physical feasibility, geometric limitations, temporal coherence, and distributional realism before moving into more complex multi-vehicle scenarios.

Contributions

This thesis presents a comparative analysis among two different generative models, GAN and Transformer. The following are the main contributions of this thesis:

1. **Design of a GAN-Based Trajectory generator**: We introduce a GAN-based architecture to generate trajectories that incorporates three key concepts: Transformer-based generator with causal attention. StyleGAN-inspired style injection and per

timestep noise. [13] Relativistic Pairing GAN Loss with Gradient Regularization [9]. We also introduced physical consistency and road boundary constraints as well as hard endpoint enforcement for entry-exit conditions.

2. **The Transformer-Based Trajectory Generator**: Inspired by TrafficGen [7], this model was modified to produce a full trajectory of a vehicle that travels through a roundabout scenario. The proposed method generates a full trajectory in one forward pass. Thus, produces absolute position information at every time step over a fixed number of time steps ($T = 120$). The proposed method is conditioned on the boundary constraints of entry and exit points, as well as a random, stochastic latent vector, so it can produce many different trajectories for the same set of conditions. The architecture of the proposed method is based on three components: a condition encoder, a memory module, and a Transformer decoder that operates on learnable temporal query tokens with both causal self-attention and cross-attention over conditioning inputs. For modeling multimodality, the network produces $K = 3$ trajectory candidates, and it is trained using a Winner-Takes-All (WTA) strategy.

3. **Experimental Evaluation**: We trained and evaluated both models on several SUMO-simulated trajectories, to assess:

   - **Quality** of generated trajectories and endpoint accuracy
   - **Motion smoothness** and physical plausibility
   - **Efficiency** in terms of computational time
   - **Trade-offs** between the two approaches.

Thesis Organization

The thesis is divided into the following chapters:

- **Chapter 2**: *Related Work 2* provides an overview of prior research concerning predictive and generative modeling in relation to trajectories and its application to autonomous vehicle systems (Classical Methods, Recurrent Networks, Generative Models).

- **Chapter 3**: *Methodology GAN and Transformer-based Approaches* proposed method based on GANs and transformers, and shows a detailed description of the architectures of the generator and discriminator. The adversarial and physics-based training objective. The implementation-related issues and the transformer-based methodology for generation, multimodal trajectory prediction and for the total training process.

- **Chapter 4**: *Experimental Evaluation* compares the two methodologies experimentally in terms of Trajectory Quality, Physical Plausibility, Computational Efficiency. The best resulting model will be tested including Multi vehicle Scenarios and Simulator Integration

- **Conclusion** provides a summary of findings; Discusses Limitations; Outlines Future Research Directions.

Chapter 2

Related Work

The following chapter introduces the theoretical context on which the thesis is based and locates it in the research scenario. We start with a brief review of Artificial Intelligence and Machine Learning, which represent the main research fields. Then we shift our focus to neural network and generative models. We conclude this section with an introduction to architectures built for the main purpose of this thesis, generating reliable traffic scenarios.

2.1 Artificial Intelligence and Machine Learning

In recent years, these topics have become among the most discussed worldwide.
Artificial Intelligence(AI) is a branch of computer science that aims to create systems capable of performing activities that normally require human intelligence. Computer vision, decision making and language understanding are the most common tasks for this kinds of systems. The development of hardware capable of delivering significant computing power has made it possible to turn theoretical insights dating back to the 1980s into reality.
Machine Learning(ML) is a subset of a AI. The main objective of this field is to develop algorithms capable of learning patterns from data. These systems do not follow predefined rules. Relying exclusively on these patterns learned from the data, they are capable of making predictions or decisions. ML algorithms improve their outputs through experience, adjusting their internal weights exclusively on the basis of data received as input.

2.1.1 Neural Networks

Deep neural networks comprise layers of interconnected nodes that can learn how data relates in complex non-linear ways. They are the most popular type of model for predicting future trajectories and motion forecasts because they can capture both temporal and spatial dependencies. To train a deep neural network, the network is run iteratively with adjustments to the network's weight based on minimizing a loss function. When training, each iteration involves running the input data through the network to create a

forecast; the difference between the forecasted value and the actual value is calculated by comparing the forecast to the actual using a loss function (such as Mean Squared Error MSE). The difference is then calculated through backpropagation to determine how to adjust the network's weights based on an optimization algorithm (such as SGD or one of its variants - e.g. Adam or RMSprop) and the formula $w \leftarrow w - \eta \frac{\partial L}{\partial w}$.

The ability to use deep neural networks to learn hierarchical feature representations has enabled deep neural networks to learn how to represent simple features in earlier layers and higher level features in later layers. A number of other architectural advancements have also greatly contributed to the success of deep neural networks, including: Convolutional Neural Networks (CNNs) that are well-suited to processing data that comes in the form of grids (e.g. images); Recurrent Neural Networks (RNNs) that are optimized for processing sequential data; and Residual Networks (ResNet) that include skip connections that enable the training of deep models.

2.2 Deterministic and Generative Models

Deterministic and generative neural networks have been categorized according to their objective function, or learning goal.

A Deterministic Model learns a one-to-one mapping from inputs to outputs by estimating the conditional probability $P(Y|X)$. Therefore, these models always predict exactly one value when an input is provided. They are typically applied to tasks including classification, regression and sequence prediction.

Generative Models attempt to estimate the distribution of the training data so that they may generate new data samples that closely match the original data distribution. Unlike deterministic models, generative models do not make predictions of individual outputs; rather they seek to represent the original data distribution $p_{data}(\mathbf{x})$ using their own model representation $p_{model}(\mathbf{x})$. Generative models are very popular for tasks like data augmentation, simulation and missing data replacement [1, 4, 10]. The methods employed for generating data samples can be further divided into explicit density models which define a probability distribution and learn the parameters of that distribution (e.g. VAEs and Autoregressive Models), and implicit density models which generate data samples directly without defining the underlying data distribution. GANs fall under the implicit density models category and use an adversarial training methodology.

Variational Autoencoders(VAEs)

Although VAEs are not the focus of this dissertation, they do provide relevant background information regarding generative modeling. As opposed to a fixed latent representation, a VAE learns a probabilistic latent space. By utilizing the learned distribution, new samples can be produced by sampling from this distribution. Unfortunately, VAEs tend to generate overly-smooth results in complex domains, leading to the use of other alternatives that better represent reality such as GANs.

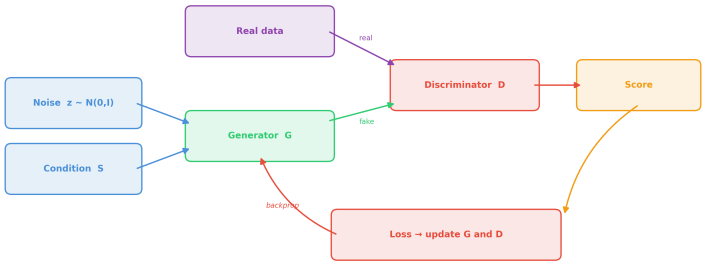

Figure 2.1. Schematic representation of a GAN

2.3 Generative Adversarial Networks (GANs)

Generative Adversarial Networks represent a relevant shift in the paradigm of generative modeling. Inspired by game theory, GANs learn to generate synthetic data through an adversarial training process. Let's delve in to the description of this type of models.

Architecture and Basic Principles

A GAN consists of two neural networks that are trained simultaneously acting as the players of the game theory.

- **Generator(G)**: Starting from a random noise vector the generator takes this vector Z and transforms it into a synthetic sample. Generating samples identical to real data is the target of this network.
- **Discriminator(D)**: The discriminator takes both real samples and data generated by G. It is structured as a binary classifier trained to output the probability that the given output is real or generated.

During the training phase, these two neural network is in a continuous challenge. This process should improve the two players constantly; the generator improves at creating fake data, and the discriminator finds new ways to distinguish real data from fake data. A basic structure of is represented in Figure 2.1.

Training Objective

Originally GANs models defined a minmax game with the following objective :

$$\min_G \max_D V(D,G) = \mathbf{E}_{\mathbf{x}\sim p_{data}}[\log D(\mathbf{x})] + \mathbf{E}_{\mathbf{z}\sim p_z}[\log(1 - D(G(\mathbf{z})))]$$

The discriminator maximizes its side of the equation by correctly classifying real (high $D(\mathbf{x})$) and fake samples (low $D(G(\mathbf{z}))$). The generator minimizes its objective by generating samples that lead the discriminator to error (high $D(G(\mathbf{z}))$).

When the discriminator is working correctly, the generator aims to reduce the Jensen-Shannon divergence between the data distribution and the model distribution, making the generated samples increasingly similar to real data.

$$D_{JS}(p_{data}||p_{model}) = \frac{1}{2}D_{KL}(p_{data}||p_{avg}) + \frac{1}{2}D_{KL}(p_{model}||p_{avg})$$

where $p_{avg} = \frac{1}{2}(p_{data} + p_{model})$.
In practice, the generator is often limited in training by using a non-saturating loss function to avoid gradient vanishing:

$$\mathcal{L}_G = -\mathbf{E}_{\mathbf{z}\sim p_z}[\log D(G(\mathbf{z}))]$$

Training Dynamics and Challenges

GAN training presents various challenges that have been addressed in several studies:

- **Mode Collapse**: The generator is not capable of generating various samples in a scenario. This can happen when G finds a certain subset of samples that consistently trick the Discriminator to label them as real data. In this context, G will never be able to synthesize realistic data.
- **Training Instability**: Since the objective is to train two different models, this can lead to oscillations between the two where neither network converges to a stable solution.
- **Vanishing Gradients**: When the discriminator becomes too good in distinguish real from fake data. This led to a zero gradient for the generator and stopped its learning.
- **Evaluation Difficulty**: GANs lack a standard quantitative measure of training progress, unlike models trained with likelihood objectives.

Advances in GAN Training Stability

Standard GAN discriminators are independent when evaluating a single example, and does not use the comparison of the generated example to the real data as a form of guidance. Relativistic GAN (RpGAN) [11] changed the way the GAN critic is trained to evaluate whether a sample represents an example of a true image or if it is *less realistic* than a sample that represents a true image. Because RpGAN uses paired examples, the gradients of all of the training steps are much better informed and less likely to collapse into fewer modes during training and less likely to have vanishing gradients.

An additional form of regularization used to stabilize GAN training is gradient regularization (R1/R2) [16], where you add a penalty for the norm of the gradient of the discriminator on both the real and generated samples. When the norm of the gradient of the discriminator gets too large, it becomes difficult for the discriminator to learn because it can block the learning signal of the generator. Training with RpGAN and R1/R2 regularization is a common form of GAN training today [9]

GANs for sequential data

Originally, GANs were designed to handle static data such as images. Extending them to temporally dependent data, like time series or vehicle trajectories, requires some adjustments to manage this dependency.

- **Recurrent GAN Architectures**: Early approaches like C-RNN-GAN and RT-GAN directly applied recurrent neural networks as the generator and discriminator. The generator produces sequences recurrently, taking noise and previous outputs as input at each time step [17].

- **TimeGAN**: Yoon et al. proposed Time-series Generative Adversarial Networks, a model specifically designed for time-series generation. TimeGAN introduces several key innovations [20]

  1. Embedding Network: Maps the high-dimensional feature space to a lower-dimensional latent space, reducing the complexity of the adversarial learning problem.
  2. Recovery Network: Reconstructs the original features from the latent representations.
  3. Supervised Loss: In addition to the unsupervised adversarial loss, TimeGAN includes a different loss that explicitly pushes the generator to acquire stepwise conditional distributions:

     $$\mathcal{L}_S = \mathbf{E}_{s,x_{1:T}\sim p}\left[\sum_t ||h_t - g_X(h_S, h_{t-1}, z_t)||^2\right]$$

     This loss makes sure that the generator learns to produce realistic connections between consecutive time steps, not just realistic marginal distributions.
  4. Joint Training: Generator and discriminator networks are trained jointly, with the embedding space conditioned to facilitate temporal patterns learning.

  TimeGAN objective brings about reconstruction, supervised, and unsupervised losses:

  $$\min_{\theta_e,\theta_r}(\lambda\mathcal{L}_S + \mathcal{L}_R)$$

  $$\min_{\theta_g}(\eta\mathcal{L}_S + \max_{\theta_d}\mathcal{L}_U)$$

  where λ and η are hyperparameters that balance the different objectives.

- **WTGAN (Wasserstein TimeGAN)**: Combines the TimeGAN structure with the Wasserstein loss function to improve training stability and model collapse, particularly useful for complex time-series with irregular dynamics

2.3.1 GANs for Traffic Trajectory Generation

The adaptation of GANs for urban traffic generation presents unique challenges:

- Multi-agent interactions: Vehicles in urban traffic scenarios interact with each other, and the generated trajectories must respect rules and other vehicles.
- Physical constraints: Synthetized trajectories must be physically realistic (accelerations, road constraints, collision avoidance).
- Map conditioning: Routes must be consistent with the road network.

These difficulties have motivated the development of peculiar architectures that combine GAN principles with scenario-specific constraints, as will be discussed in the following sections.

2.4 Transformer Models

Transformers are one of the most relevant model structures for tasks such as language processing, computer vision, and time series. Transformers overturned Recurrent Models by replacing recurrence with a self-attention mechanism [19]. This model computes representations for all positions using self-attention to directly map dependencies.

Self-Attention Mechanism

The self-attention mechanism, as said in the last paragraph, is the key innovation introduced by Transformers; this mechanism allows each position of a sequence to consider all the other positions. Self-attention, given an input sequence, calculates a sum of weighted values; these weights are assigned by the compatibility between the single position and the sequence.
Scaled Dot-Product Attention: The attention function maps Query Q, keys K, and values V to an output:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where d_k is the dimension of the keys.
The scaling factor $\frac{1}{\sqrt{d_k}}$ prevents the dot products from growing too large; in this way, the softmax can compute useful gradients. For self-attention, Q, K, and V are all derived from the same input sequence through learned linear projections.
Multi-Head Attention: The transformer, instead of computing a single function, relies on multiple attention mechanisms in combination,

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O$$

where each head is computed as:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

This multi-head configuration allows the model to retrieve information from different representation subspaces in different positions and understand how they impact the previous or subsequent ones. In the original Transformer, $h = 8$ heads are used, each with size $d_k = d_v = d_{model}/h = 64$.

Transformer Architecture

The transformer is composed of an encoder-decoder structure:
Encoder:The encoder comprises of $N = 6$ identical layers, each containing two sub-layers:

1. Multi-head self-attention mechanism
2. Position-wise feed-forward network

Each sub-layer is wrapped with a residual connection and layer normalization:

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

The feed-forward network applies two linear transformations with a ReLU activation:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

Decoder:The decoder also includes $N = 6$ identical layers. Each layer has three sub-layers:

1. Masked multi-head self-attention which prevents attention to future positions.
2. Multi-head attention over encoder outputs (cross-attention)
3. Position-wise feed-forward network

The masking in the decoder makes sure that predictions for position i rely only on known outputs at positions less than i. This preserves the autoregressive property. Transformer architecture is represented in Figure 2.2

Positional Encoding

Since the Transformer has no recurrence or convolution, it does not have a built-in understanding of sequence order. To add positional information, we include positional encodings in the input embeddings. The original Transformer uses sinusoidal positional encodings:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

In this formula, *pos* refers to the position and i indicates the dimension. This approach allows the model to learn to focus on relative positions, as PE_{pos+k} can be expressed as a linear function of PE_{pos} for any fixed offset k. Alternatively, we can use learned positional embeddings that perform similarly.

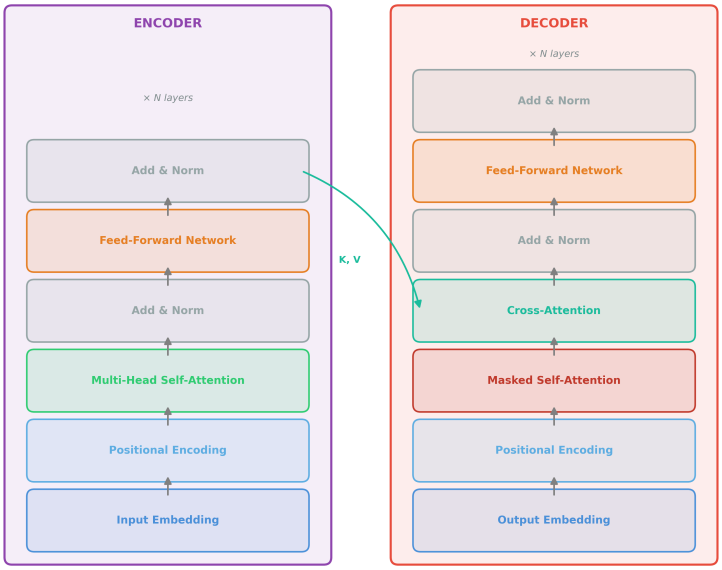

Figure 2.2. Schematic representation of a Transformer

Computational Complexity Analysis

The self-attention mechanism has different computational characteristics than recurrent and convolutional layers.

Layer Type	Complexity per Layer	Sequential Operations	Max Path Lengths
Self Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$

Observations:

- Parallelization: Self-attention needs only $O(1)$ sequential operations, allowing for full parallelization
- Path Length: The longest path between any two positions is $O(1)$, which helps learn long-range dependencies.
- Memory: The $O(n^2)$ complexity can be a bottleneck for very long sequences. This drives the need for efficient variants.

Transformer Variants and Extensions

The original Transformer was really successful. That led to a lot of different versions of the Transformer.

- **Encoder- Models**: like BERT are really useful. They only use the encoder part for tasks like classification and understanding features from the data. The good thing about these models is that they can look at everything at the same time. This is thanks to attention, which lets each part of the data be contextually compared to every other part.
- **Decoder- Models**(for example GPT): For generating tasks requiring autoregressive generation, utilize the decoder stack and apply causal (one-directional) attention mechanics from the transformer architecture.
- **Efficient Transformers**: Address the quadratic complexity of self-attention through various approximations:
- **Linear Attention** (Performer, Linear Transformer): Approximate soft-max attention with kernel methods
- **Sparse Attention**: Restrict attention to local windows and selected global positions

Decoder-Only Transformers for Conditional Generation

The original Transformer has a combined encoder/decoder architecture. A very relevant family of models utilizes the decoder stack solely. In this format of models, which were originally developed for autoregressive language models [19]the model generates all positions in a target output sequence one at a time with each position attending to all previous positions due to use of causal masks. These architectures are particularly well-suited to generation tasks where the goal is to generate an entire sequence from a smaller input.

In addition, when there exists some form of external conditioning data for example, boundaries for the output sequence or context for the scene then this can be included through use of cross-attention over a set of memory tokens which have been encoded from the conditioning inputs. Thus, we can utilize the architecture of an encoder-conditioned decoder and therefore clearly separate the task of representing context from the task of generating sequences. Each layer in the decoder performs two types of attention causal self-attention to its own previously-generated tokens; cross-attention to the fixed conditioning memory. This provides for the ability to combine various forms of conditioning information into a single generative architecture without having to modify the core generative structure of the model.

2.4.1 Transformers for Sequential and Temporal Data

Transformers have done well on lots of tasks that involve sequences and that is not just limited to understanding natural language:

- **Time Series Forecasting**: Transformers are really good at understanding patterns that happen over time. This is useful for Time Series Forecasting because it needs to understand both short-term and long-term patterns, in Time Series Forecasting.

- **Motion and Trajectory Prediction**: When we want to figure out where things like cars or people are going to move in the future the Transformers can help us do that. Transformers can:
  Capture interactions between multiple agents through cross-attention.
  Model the influence of the environment (maps, obstacles) through conditioning.
  Generate diverse, multi-modal predictions through appropriate output distributions.

- **Multimodal Prediction and Winner-Takes-All Training** Predicting the future of an object's trajectory in the environment is fundamentally a multi-modal problem. For the same set of initial conditions there can be many possible futures; and models such as Multipath++ [18] generate K candidate trajectories along with probabilities associated with each one. In order to train this type of model it has become common practice to use Winner-Takes-All (WTA) which trains the model by only allowing gradients to flow through the path that is most similar to the actual ground truth, thus training the model to discover K different and diverse paths rather than one mean trajectory.

Advantages for Traffic Modeling:

1. Multi-agent Modeling: Attention mechanisms naturally model interactions between vehicles.

2. Long-range Dependencies: Important for understanding how past behavior influences future trajectories

3. Parallel Processing: Enables efficient training on large datasets

The combination of Transformers with generative modeling approaches offers a powerful framework for traffic scenario generation, enabling both the capture of complex temporal dynamics and the generation of diverse, realistic scenarios.

2.5 Diffusion Models

Diffusion models are a class of generative models that have shown excellent performance on many data generation tasks over recent years. The operation of these models is based on a two-stage approach: a forward process in which an increasing amount of noise is added to the input data, and a reverse process in which a neural network attempts to remove this noise and generate new realistic data samples.

Because diffusion models can model multiple, multimodal data distributions (which makes them particularly suitable for trajectory prediction and generating traffic scenarios), but because their iterative denoising process can be very computationally expensive, diffusion models were not the main subject of this thesis.

2.6 Traffic Generation Model

2.6.1 TrafficGen

The TrafficGen model developed by Feng and others uses real-world data to generate realistic driving environments that contain many different kinds of driving scenarios, making it easier to evaluate how an autonomous driving system may perform under different conditions [7].

Motivation and Problem Setting

There are many limitations to existing methods of traffic scenario generation:

1. **Replay-based**: Take recorded trajectories from real-world datasets and simply replay them. These approaches require a large amount of data to recreate the environment accurately while still maintaining fidelity; additionally, the recorded trajectories are often incomplete.

2. **Rule-based**: Create vehicle placement and behavior using hand-developed rules. These methods do not allow for a diverse enough set of scenarios and do not accurately model the complexity of real-world traffic.

3. **Previous generative methods**: Generative models, such as SceneGen, can provide traffic snapshots; however, post-processing must take place before use in a simulation environment and such systems do not produce dynamic vehicle trajectories.

The method proposed in TrafficGen aims to address the above limitations by learning to generate both the initial vehicle state and the long-term future trajectory of the vehicles from the observed trajectories present in the real-world.

Problem Formulation

A traffic scenario is defined as $\tau = (\mathbf{m}, \mathbf{s}_{1:T})$, where: $\mathbf{m}$ is the High-Definition (HD) road map $\mathbf{s}_{1:T} = [s_1, ..., s_T]$ is the state series of traffic vehicles Each $s_t = \{s_t^1, ..., s_t^N\}$ contains states of N vehicles at time t.

Given an existing scenario, TrafficGen learns to generate new scenarios $\tau' = (\mathbf{m}, \mathbf{s}'_{1:T'})$ with similar distribution but different states and potentially longer time horizons.

Architecture

The estimate of TrafficGen is a Vector Based Traffic Representation. The HD map is split into smaller areas all the lanes have been vectorized; The vector v_i representing the segment of a lane gives you the coordinates for these segments as:

$$v_i = (p_i^s, p_i^e, t_i, u_i) \oplus (m_i, q_i^v, h_i, vel_i, bbox_i)$$

where: - p_i^s, p_i^e: start and end points of the lane segment - t_i, u_i: lane type and traffic light state - m_i: boolean indicating vehicle presence - $q_i^v, h_i, vel_i, bbox_i$: vehicle position, heading, velocity, and bounding box

TrafficGen follows an encoder-decoder structure:

- **Encoder**: Rely on Multi-Context Gating (MCG) blocks to combine information from various areas to create a single global context level c' and regional features $\mathbf{v}' = \{v_i'\}$.
- **Decoder**: Creates traffic scenarios in two segments:
  1. **Vehicle Placement**: Place vehicles based on how you would define a categorical probability distribution over sites and then use the probability distribution to sample attributes or vehicles (the cars) in the normal car distribution using the Gaussian Mixer Model(GMM):

$$w_j = \text{MLP}_{place}(v_j'), \quad i \sim \text{Categorical}([w_1, ..., w_I])$$

  2. **Trajectory Generation**: Multimodal Trajectory predictions using a motion forecasting model to create K trajectories and their associated probabilities at each timestep:

$$\text{Decoder}(c') = \{pos_{t:t+L}^k, prob_k\}_{k=1}^K$$

Training Strategy

TrafficGen uses BERT-like training by masking random vehicle areas so the model will learn how to reconstruct the masked regions. Loss function is defined by combining Binary Cross Entropy for vehicle presence with Negative Loglikelihood for the distribution of attributes:

$$\mathcal{L} = \mathcal{L}_{presence} + \mathcal{L}_{pos} + \mathcal{L}_{heading} + \mathcal{L}_{speed} + \mathcal{L}_{size}$$

where $\mathcal{L}_{presence} = \text{BCE}(w_i, m_i)$ and each attribute loss is the negative log-likelihood under its respective GMM distribution.

Application

There are several applications for TrafficGen:

- **Scenario generation**:Develops entirely new traffic scenarios based on HD mapping information.
- **Scenario augmentation**: Generates additional vehicles to add to existing traffic scenarios.
- **Trajectory inpainting**: Fills in missing pieces in incomplete or broken-down trajectories.

Testing has demonstrated that utilizing TrafficGen-generated traffic scenarios for autonomous driving agent training results in improved performance and safety over the utilization of real-world data only.

2.6.2 Multipath++

One of the state-of-the-art trajectory prediction models currently used to generate multilayer trajectories within multiple traffic simulation systems is Multipath++ [18]. It is incorporated into TrafficGen. Understanding how different trajectories are produced provides useful insight into this thesis.

Motion Prediction Problem

Predictions on future motion for all agents (vehicles, pedestrians, and cyclists) within a given location (roadway, pedestrian path, etc) depend on three factors:

- Historical trajectories of all agents
- Scene context (road geometry, traffic signals, lane markings)
- Interactions with other agents

How agents interact with each other affects the prediction of motion. If an agent is approaching another agent at an intersection, the agent's actions may affect the prediction of motion.

Architecture and Output

Multipath++ relies on polyline representations and attention-based processing for processing historical data from all agents. The objective is to encode the context of the scene and historical events for each actor. The main feature of this design is the generation of K different mode outputs based on the agent's conditions at any time. Each agent will have k:

$$\{(\pi_k, \{\mu_t^k, \Sigma_t^k\}_{t=1}^T)\}_{k=1}^K$$

where π_k is the probability of mode k, and μ_t^k, Σ_t^k are the mean and covariance of the predicted position at time t.The probabilistic nature of this representation allows quantification of uncertainty and the collection of multiple samples from the distribution of trajectories. A second module will aggregate the predictions of all agents based on each other's interactions to produce a consistent view of the scene.
In terms of generating traffic, the predictions of motion forecasting models such as Multipath++ can produce realistic trajectories based on actual traffic data with multiple sample cases through the different modes of trajectories produced and long-term generation using autoregressive rollouts, resulting in many different possible trajectories from an agent over an extended period of time. When combined with generative modeling techniques such as TrafficGen, these models can produce realistic and complete traffic scenarios that follow the road structure and normal driving behaviour.

Chapter 3

Methodology

The objective of this thesis is to build a model able to generate realistic vehicle trajectories. The following chapter describes the procedures developed to create synthetic traffic trajectories within a roundabout environment. These simulations are conditioned on starting and ending points, suitable for populating road scenarios and training self-driving systems.

To reach the specified goal, two different deep learning models are compared:

1. **Trajectory GAN** A GAN with Transformer backbone training relying on Relativist pairing(RpGAN) objectives and scenario specific physical constraints

2. **Trajectory Transformer** Transformer-based model trained with supervised reconstruction losses

These two models are compared and evaluated on their performances executed on the same dataset and task: to generate a realistic trajectory, i.e., one that respects conditions such as the roadway and speed, for a vehicle in a roundabout. Conditioned by an entry and exit point.

3.1 Problem Formulation

We have a specified Entry and Exit point on a roundabout to generate a realistic trajectory of a single vehicle from a given Starting Point (x_{start}, y_{start}) to a target location (x_{end}, y_{end}), while providing constraints of remaining on the road surface and displaying visually realistic motion, accelerate and decelerate without sudden changes and producing speeds which are appropriate to traffic regulations.

We can define the problem of generating trajectories as learning a conditional distribution:

$$p(\mathbf{X} \mid \mathbf{S})$$

where $\mathbf{X} \in \mathbb{R}^{T \times F}$ is the generated trajectory with T time steps and F features per step, and $\mathbf{S} = [x_{start}, y_{start}, x_{end}, y_{end}]$ represents the conditions of the endpoint. Each

point in the trajectory contains $F = 4$ features: position (x, y), speed v, and angle of view α. The generated trajectories are created using a latent noise vector $\mathbf{z} \sim \mathcal{N}(0, I)$ along with the endpoint conditions $\mathbf{S}$ as inputs to the model. Because of this stochasticity of the latent input, we are able to generate many unique trajectories between the same start and end points, thus accounting for the variability in human driving.

3.2 Trajectory GAN

To that end, we will be employing a conditioned GAN to produce driving trajectories called Trajectory GAN. This will utilize adversarial training techniques combined with state-of-the-art Transformer architecture and domain-specific road compliance constraints in order to create the generator and discriminator of our GAN. We will also be utilizing a Relativistic pairing GAN (RpGAN) objective augmented by physical and road adherence loss to train the generator and discriminator in a Relativistic manner.

3.2.1 Architecture

Generator

The generator uses a style-based conditioning method, as inspired by StyleGAN's treatment of images, but this will be applied to sequential data in a decoder-only Transformer architecture.

Input Construction The generator has three different types of input, which we will combine into one sequence before sending it to the generator. The first input is composed of the global style vector, which is formed by combining the latent noise with the endpoint conditions:

$$\mathbf{s} = \mathrm{FC}_{style}([\mathbf{z}; \mathbf{S}]) \in \mathbb{R}^{d_{model}}$$

This global style of trajectory will be shared across all T timesteps. Thus making sure that the trajectory-level information is constant, no matter the timestep. Secondly, having independent noise at each timestep enables local variation:

$$\boldsymbol{\epsilon}_t \sim \mathcal{N}(0, I), \quad \mathbf{n}_t = \mathrm{FC}_{noise}(\boldsymbol{\epsilon}_t)$$

Third, the endpoint conditions are embedded per-timestep to reinforce boundary constraints: $\mathbf{c}_t = \mathrm{FC}_{cond}(\mathbf{S})$.

The combined input is:

$$\mathbf{x}_t = \mathbf{s} + \mathbf{n}_t + \mathbf{c}_t + \mathrm{PE}(t)$$

The sinusoidal positional encoding is represented as $\mathrm{PE}(t)$

Causal-transformation. The generating network is a stack of causal mask transformer encoder layers $\mathbf{M}_{causal}$:

$$\mathbf{H} = \text{Transformer Encoder}(\mathbf{X},\ \text{mask} = \mathbf{M}_{causal})$$

The causal mask, since it is upper-triangular, will prevent a position from being influenced by future positions. This will provide autoregressive temporal consistency

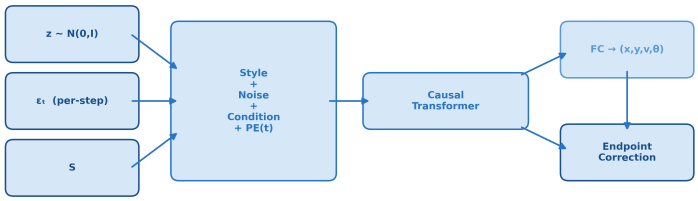

Figure 3.1. Schematic representation of the Generator

between time-steps. The output projection will provide an output size of $F = 4$ features per timestep:

$\hat{\mathbf{y}}_t = \mathrm{FC}_{out}(\mathbf{h}_t)$.

An architectural design choice was applying deterministic correction to the predicted coordinates (x, y); this will ensure that the trajectory will start and end in the same place. To compute this, we do a baseline linear interpolation of the first and last valid timesteps of the trajectory:

$$\mathbf{b}_t = (1 - \tau_t)\,\mathbf{p}_{start} + \tau_t\,\mathbf{p}_{end}$$

where $\tau_t = t/(L_i - 1)$ is the normalized time for a trajectory of length L_i. A parabolic window $w_t = 4\,\tau_t(1 - \tau_t)$ is then applied, which equals zero at endpoints and one in the middle:

$$\hat{\mathbf{p}}_t = \mathbf{b}_t + w_t \cdot (\mathbf{y}_t^{xy} - \mathbf{b}_t)$$

The first and last valid timesteps are hard-set to $\mathbf{p}_{start}$ and $\mathbf{p}_{end}$ respectively. This approach is fully differentiable while guaranteeing exact endpoint satisfaction, regardless of trajectory length. The structure of the generator is stylized in Figure 3.1.

Discriminator Architecture

The discriminator $D(\mathbf{X}, \mathbf{S})$, also called a critic, is structured as a bidirectional Transformer encoder. The critic elaborates the complete trajectory simulation simultaneously; its task is to output a single scalar score that indicates if it considers the simulation realistic or generated. Unlike the generator, the discriminator looks at the sequence in its completeness. For this reason, this model is able to identify inconsistencies and violations of physical constraints across the complete generated path.

Input Embedding: First, the features of the trajectory are projected to the model dimension:

$$\mathbf{H}_0' = \mathrm{FC}_{\mathrm{input}}(\mathbf{X}) \in \mathbb{R}^{T \times 256}$$

Condition Integration: The critic incorporates the conditioning information through additive transmission, in a similar way to the generator:

$$\mathbf{H}_0 = \mathbf{H}'_0 + \text{repeat}(\text{FC}_{\text{cond}}(\mathbf{S}), T)$$

Positional Encoding: The same sinusoidal positional encodings are considered:

$$\mathbf{H}_0 \leftarrow \mathbf{H}_0 + \text{PE}$$

Bi-directional Transformer Encoder:$N_{\text{layers}} = 3$ Transformer encoder layers without causal masking process the whole sequence:

$$\mathbf{H}_\ell = \text{TransformerLayer}_\ell(\mathbf{H}_{\ell-1}, \text{padding\_mask} = \mathbf{m})$$

The same architectural components of the generator are used by each layer of the critic (4 attention heads, 1024-dimensional feed-forward layers, GELU activation, dropout 0.1), but in this case, the sequence is processed bidirectionally.

Masked Mean Pooling: To combine the representation at the sequence level while respecting the padding mask, we apply masked mean pooling:

$$\mathbf{h}_{\text{global}} = \frac{\sum_{t=0}^{T-1} m_t \cdot \mathbf{H}_{N_{\text{layers}},t}}{\sum_{t=0}^{T-1} m_t}$$

where m_t is the validity mask ($m_t = 1$ for valid timesteps, $m_t = 0$ for padding).

Output Score: The global representation is finally projected to a single scalar score:

$$D(\mathbf{X}, \mathbf{S}) = \text{FC}_{\text{out}}(\mathbf{h}_{\text{global}}) \in \mathbb{R}$$

The goal of the critic is to output a high value score for real trajectories and low scores for synthetic ones. The two models work against each other. The generator is trained to produce sequences that can fool the critic into assigning high scores to its outputs.

Training Objective

Several loss functions are combined in the training objective. The reason for using different functions is to ensure that the generator is capable of producing trajectories that respect not only feature distributions but also trajectories that are physically plausible and respect road constraints.

Relativistic Pairing Gan (RpGAN)

I relied on the Relativistic Pairing GAN framework introduced by Jolicoeur-Martineau [11]. This approach has been shown to ensure a more stable training process than classical GAN formulations. The main insight of RpGAN, as proven in recent work [9], the critic is capable of providing stronger gradients when it tries to predict the probability that a real sample is more realistic than a fake one. In this way, the model collapse is reduced compared to the original GAN loss.

Discriminator Objective: The loss of the discriminator is formulated as:

$$\mathcal{L}_D = \mathbb{E}_{\mathbf{X}_{\text{real}} \sim p_{\text{data}}, \mathbf{z} \sim p_z} \left[\text{softplus}\left(-(D(\mathbf{X}_{\text{real}}, \mathbf{S}) - D(G(\mathbf{z}, \mathbf{S}), \mathbf{S})) \right) \right]$$

where $\text{softplus}(x) = \log(1 + e^x)$ is the smooth approximation of ReLU. In this structure the critic tries to maximize the margin $D(\mathbf{X}_{\text{real}}, \mathbf{S}) - D(\mathbf{X}_{\text{fake}}, \mathbf{S})$.

Generator Objective The generator objective is to reverse this ordering:

$$\mathcal{L}_G^{\text{adv}} = \mathbb{E}_{\mathbf{X}_{\text{real}} \sim p_{\text{data}}, \mathbf{z} \sim p_z} \left[\text{softplus} \left(D(\mathbf{X}_{\text{real}}, \mathbf{S}) - D(G(\mathbf{z}, \mathbf{S}), \mathbf{S}) \right) \right]$$

The described methodology provides more significant gradients to the generator throughout training and has been proven to reduce mode collapse risk in conditional generation tasks [9].

R1/R2 Gradient Regularization

The critic can rapidly became too good in distinguish the real trajectories and the fake ones. For this reason i had to stabilize its training. To so I applied zero-centered gradient penalty on both real and generated samples. This regularization, known as R1/R2 regularization [16] , has been proven to be a key procedure to achive stable GAN training. This regularization has been used extensively in modern GAN architectures [9].

R1 Penalty (Real Data) The R1 penalty is computed on real sequences:

$$\mathcal{R}_1 = \frac{1}{2} \mathbf{E}_{\mathbf{X}_{\text{real}} \sim p_{\text{data}}} \left[\| \nabla_{\mathbf{X}_{\text{real}}} D(\mathbf{X}_{\text{real}}, \mathbf{S}) \|^2 \right]$$

R2 Penalty (Generated Data) The R2 penalty is computed on synthetic trajectories:

$$\mathcal{R}_2 = \frac{1}{2} \mathbf{E}_{\mathbf{z} \sim p_z} \left[\| \nabla_{\mathbf{X}_{\text{fake}}} D(G(\mathbf{z}, \mathbf{S}), \mathbf{S}) \|^2 \right]$$

For padded trajectories, we consider only valid timesteps by masking the gradient norms:

$$\| \nabla_{\mathbf{X}} D(\mathbf{X}, \mathbf{S}) \|^2_{\text{masked}} = \frac{\sum_{t=0}^{T-1} m_t \| \nabla_{\mathbf{x}_t} D(\mathbf{X}, \mathbf{S}) \|^2}{\sum_{t=0}^{T-1} m_t \cdot 4}$$

The total critic loss becomes:

$$\mathcal{L}_D^{\text{total}} = \mathcal{L}_D + \gamma(\mathcal{R}_1 + \mathcal{R}_2)$$

where $\gamma = 15.0$ is the weight assigned to the gradient penalty, chosen based on the decision made in recent work on stable GAN training [9]. The R1/R2 penalties are computed every $k = 2$ discriminator steps to reduce computational weight while maintaining training stability.

Physical Consistency Loss

To help the generator to create sequences that respect basic kinematic constrains. We created a physical consistency loss the penalizes the generated sequences that do not respect the expected patterns. This loss is composed by different components:

- **Spatial Smoothness Loss**: Penalty for large position changes between one time stamp and the following one. The reason is to encourage smooth movements:

$$\mathcal{L}_{\text{smooth}}^{\text{pos}} = \frac{1}{\sum_{t=0}^{L-2} m_t m_{t+1}} \sum_{t=0}^{L-2} m_t m_{t+1} \|\mathbf{p}_{t+1} - \mathbf{p}_t\|$$

- **Speed Range Penalty**: We penalize large speed changes:

$$\mathcal{L}_{\text{smooth}}^{\text{speed}} = \frac{1}{\sum_{t=0}^{L-2} m_t m_{t+1}} \sum_{t=0}^{L-2} m_t m_{t+1} |v_{t+1} - v_t|$$

- **Speed Range Penalty**: We penalize unrealistic speed values. To do so we applied for speeds outside a predefined range:

$$\mathcal{L}_{\text{range}}^{\text{speed}} = \frac{1}{\sum_{t=0}^{L-1} m_t} \sum_{t=0}^{L-1} m_t \left[\text{ReLU}(0.02 - v_t) + \text{ReLU}(v_t - 0.98)\right]$$

  where the bounds [0.02, 0.98] create a margin within the normalized [0, 1]range to accommodate the denormalization mapping.

The total physical consistency loss is:

$$\mathcal{L}_{\text{phys}} = 0.1\mathcal{L}_{\text{smooth}}^{\text{pos}} + 0.1\mathcal{L}_{\text{smooth}}^{\text{speed}} + 0.2\mathcal{L}_{\text{range}}^{\text{speed}}$$

These physical constraints are similar in substance to the supervised loss in TimeGAN [20], which explicitly pushes the model to capture conditional distributions in temporal data for each timestamp.

Road Constraint Loss

A key condition that makes a generated trajectory realistic is the condition of remaining within the physical boundaries of the context. To ensure that, I relied on a differentiable *road constraint loss*.

1. **Signed Distance Field**: Starting form a 2D distance map $\mathcal{D} : \mathbb{R}^2 \to \mathbb{R}$ of the roundabout geometry, where $\mathcal{D}(x, y) = 0$ for that are on the road surface and $\mathcal{D}(x, y) > 0$ for coordinates outside the road boundaries. A regular grid of size $H \times W$ is the discretization of the distance map.

2. **Distance Sampling**: For each point (x_t, y_t) of a synthesized trajectory, we first denormalize to metric coordinates:

$$\begin{bmatrix} \tilde{x}_t \\ \tilde{y}_t \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \end{bmatrix} \odot (\mathbf{X}_{\max} - \mathbf{X}_{\min}) + \mathbf{X}_{\min}$$

where $\mathbf{X}_{\min}, \mathbf{X}_{\max}$ are the limits of the roundabout collected during preprocessing as normalization parameters. The next step consists of sampling the distance field using bilinear interpolation through differentiable grid sampling:

$$d_t = \text{GridSample}(\mathcal{D}, \text{normalize}(\tilde{x}_t, \tilde{y}_t))$$

3. **Road Loss Formulation**: As previously said, the target is to penalize trajectories that venture outside the road constraints:

$$\mathcal{L}_{\text{road}} = \frac{1}{\sum_{t=0}^{L-1} m_t} \sum_{t=0}^{L-1} m_t \cdot d_t^2$$

To strongly penalize off-road points, I used a quadratic penalty (power = 2). The complete loss is normalized by the total number of timesteps. This prevents long trajectories from dominating the gradient. The quadratic penalty is used to increase the magnitude of large violations so that trajectories that deviate from the road by a significant amount will have a strong penalty associated with them.

4. **Warmup Schedule**: To prevent the generator from learning shortcuts that can easily fool the critic, the road constraint is introduced gradually. In this way, the generator learns basic motion patterns before enforcing strict road compliance. This warmup strategy is inspired by approaches in sequential generation [20]. We use a linear warmup schedule:

$$\lambda_{\text{road}}(n) = \begin{cases} 0 & n < n_{\text{start}} \\ \frac{n - n_{\text{start}}}{n_{\text{ramp}}} \lambda_{\text{road}}^{\max} & n_{\text{start}} \leq n < n_{\text{start}} + n_{\text{ramp}} \\ \lambda_{\text{road}}^{\max} & n \geq n_{\text{start}} + n_{\text{ramp}} \end{cases}$$

where n is the training step, $n_{\text{start}} = 10{,}000$ is the warmup start step, $n_{\text{ramp}} = 30{,}000$ is the ramp duration, and $\lambda_{\text{road}}^{\max} = 2.0$ is the target weight

Loss Function

The total generator loss combines all previous components:

$$\mathcal{L}_G^{\text{total}} = \mathcal{L}_G^{\text{adv}} + \lambda_{\text{phys}} \mathcal{L}_{\text{phys}} + \lambda_{\text{road}}(n) \mathcal{L}_{\text{road}}$$

where $\lambda_{\text{phys}} = 0.5$ and $\lambda_{\text{road}}(n)$ follows the warm up schedule described before.

3.2.2 Training Procedure

The training schedule utilizes an alternating optimization scheme typical of GAN training. Several stabilization procedures inspired from recent GAN architectures [9].

Table 3.1. Trajectory GAN training hyperparameters.

Parameter	**Value**
Model dimension d	256
Attention heads	4
Generator layers	5
Discriminator layers	3
Feed-forward dimension	1024
Latent dimension	256
Batch size	128
Optimizer	Adam
Generator learning rate	2×10^{-4}
Discriminator learning rate	1×10^{-4}
Critic updates (n_{critic})	2
Gradient penalty (λ_{GP})	20

Hyperparameters

The key hyperparameters are summarized in Table 3.1.

Noise Injection

Another technique utilized to improve training stability is injecting Gaussian noise to the inputs of the discriminator [12]. The purpose of this methodology is to prevent discriminators from overfitting, thus becoming too good at distinguishing real and generated data.

$$\tilde{\mathbf{X}} = \mathbf{X} + \sigma_n \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$$

.

The noise is injected while respecting the padding mask. i.e., applied only to valid timesteps. The noise standard deviation is scheduled on exponential decay schedule:

$$\sigma_n(e) = \max(\sigma_{\min}, \sigma_0 \cdot \alpha^e)$$

where $\sigma_0 = 0.02$ is the initial noise level, $\alpha = 0.998$ is the decay factor, $\sigma_{\min} = 0.001$ is the minimum noise level, and e is the epoch number. In this way, the critic is allowed to focus on coarse distribution in early stages and fine details later on.

Mixed Precision Training

Automatic mixed precision (AMP) is utilized for training with the goal of both speeding up training and reducing memory usage by using PyTorch's built-in AMP. For the forward passes,, we use FP16 (half precision) but we maintain the loss scaling and gradient calculations in FP32 (single precision) accuracy. The gradient penalties (R1/R2) are intentionally calculated in FP32 to ensure numerical stability, in accordance with best practices in GAN literature [14].

Inference

A single forward pass is sufficient to create a new trajectory given its starting location and ending location.

1. Sample a latent code from the prior: $\mathbf{z} \sim \mathcal{N}(0, I_{256})$
2. build the conditioning vector: $\mathbf{S} = [x_{\text{start}}, y_{\text{start}}, x_{\text{end}}, y_{\text{end}}]$
3. Generate the trajectory: $\mathbf{X} = G(\mathbf{z}, \mathbf{S})$
4. Detect the end of the trajectory based on vehicle exit criteria and truncate it to the desired length

A latent variable $\mathbf{z}$ is defined to account for all behavioral variability. For a given pair of boundary conditions, different samples of z will allow the model to produce multiple possible trajectories, which reflect the multimodal nature of real driving behavior.

Because latent sampling is stochastic, a diverse set of trajectories can be created for each entry-exit pair. Therefore, the variability found in actual driver behaviours can be captured through the multimodal nature of each set of trajectories. Similar sampling strategies have been utilized in other generative models for sequential data [20]. To produce multiple trajectory samples based on one condition, steps 1-3 can be repeated using different random seeds.

3.3 Conditioned Trajectory Transformer

The Conditional Generative Adversarial Networks (GAN) architecture allows for flexibility in generating trajectories; however, the training instability and variability in results are major concerns in practice. The adversarial training process has high sensitivity to hyperparameters, and this may cause both unstable convergence and large variance in the generated trajectories. Therefore, these limitations led us to develop a framework that uses a Transformer-based approach to model the sequential dependency of the trajectory using an attention mechanism and to provide a more stable and structured framework for trajectory generation. The second methodology is described in the next session. The second method is based on the use of a transformer to produce artificial traffic trajectories. This is an adaptation of a trajectory generation strategy presented in TrafficGen [7] and applied to the SUMO dataset and to the roundabouts scenario studied in this case. As opposed to other methods that predict the displacement at each time step, the proposed model predicts the position of the vehicle at each time step directly, in one single forward pass, using all $T = 120$ time steps.

Architecture

The proposed model consists of three components: a condition encoder that encodes the conditions of entry and exit, a memory component that combines the condition encoding with a stochastic latent vector, and a transformer decoder that produces the entire trajectory.

Ego Condition Encoder The condition encoder is a 2-layer MLP that takes as input the boundary conditions $s = (x_{\text{start}}, y_{\text{start}}, x_{\text{end}}, y_{\text{end}}) \in \mathbb{R}^4$ and produces a latent representation $h_{\text{ego}} \in \mathbb{R}^d$ of these conditions. It uses the layer normalization and the GELU activation function. The dimension of the model is $d = 256$.

Memory Assembly The decoder performs cross-attention to a set of conditioning tokens. In the case of a single vehicle, the memory contains 2 tokens:

$$M = [h_{\text{ego}};\, h_z] \in \mathbb{R}^{2\times d} \tag{3.1}$$

where $h_z = \text{GELU}(\text{LayerNorm}(\text{Linear}(z)))$ is a projection of the latent vector $z \sim \mathcal{N}(0, I_{256})$. This transformation allows the model to produce different trajectories for the same entry-exit pair by randomly selecting different values of z.

Transformer Decoder This decoder creates a complete vehicle trajectory based on the $T = 120$ learnable query tokens $Q \in \mathbb{R}^{T\times d}$, for each timestep of the trajectory. Temporal encoding is achieved through the addition of sinusoidal positional encodings [19].

The 120 query tokens are processed through $N = 6$ Transformer decoder layers [19]. Each of these layers includes:

1. A masked multi-head self-attention layer, with a causal mask applied to ensure that timestep t only attends to previous positions $1, \ldots, t-1$
2. Multi-head cross-attention over memory M
3. A fully connected feedforward network with GELU activation.

Each of the above layers utilizes residual connections and use pre-norm layer normalization. The model has 8 attention heads, 1024-dimensions for feed-forward, and dropout of 0.1. The causal mask forces the model to predict the position of the vehicle at time t as a function of the trajectory up to time t, and not as a function of any future positions of the vehicle, which represents the sequential nature of vehicle motion. The hidden state of the final layer is then transformed into $F = 4$ feature values per timestep via independent output heads with sigmoid activation:

$$\hat{y}_t^{(k)} = \sigma(\mathrm{MLP}(H_N[t])) \in [0,1]^4, \quad k = 1, \ldots, K \tag{3.2}$$

The sigmoid forces the output to be within the normalized range of the input data. The model outputs absolute positions directly.

Multimodal Output The model produces $K = 3$ trajectory candidates, each from an independent output head, yielding Modes $\in \mathbb{R}^{K\times T\times 4}$. Mode probabilities are computed from the global average of the decoder output:

$$p = \mathrm{softmax}\left(\mathrm{MLP}\left(\frac{1}{T}\sum_{t=1}^{T} H_N[t]\right)\right) \in \mathbb{R}^K \tag{3.3}$$

At inference, the mode with highest probability can be selected, or modes can be sampled from the distribution, or a weighted average can be computed. The summary of the architecture shown in Figure 3.2

Training Procedure

Training Objective The loss function combines several components to produce trajectories that are accurate, physically plausible, and diverse.

Winner-Takes-All (WTA) Loss. Only the mode closest to the ground truth receives gradient updates, following the approach used in [7]:

$$L_{\mathrm{WTA}} = \min_{k\in\{1,\ldots,K\}} \frac{\sum_t m_t \|M_k[t] - X_{\mathrm{target}}[t]\|^2}{\sum_t m_t} \tag{3.4}$$

where m_t is the validity mask.

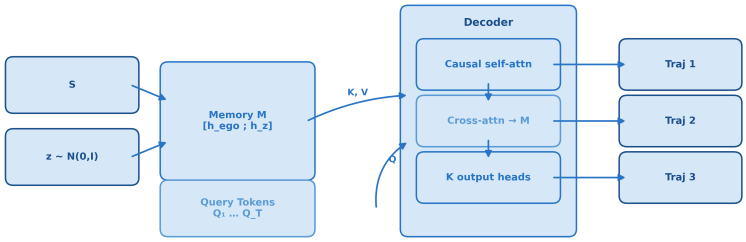

Figure 3.2. Schematic representation of the Transformer

Endpoint Loss. Enforces the boundary conditions on the start and end points:

$$L_{\text{endpoint}} = \|\hat{x}_0 - x_{\text{start}}\|^2 + \|\hat{x}_{L-1} - x_{\text{end}}\|^2 \tag{3.5}$$

Smoothness Loss. Penalizes abrupt accelerations through second-order differences:

$$L_{\text{smooth}} = \frac{\sum_t m_t m_{t+1} m_{t+2} \|\Delta^2 x_t\|^2}{\sum_t m_t m_{t+1} m_{t+2}} \tag{3.6}$$

where $\Delta^2 x_t = (x_{t+2} - x_{t+1}) - (x_{t+1} - x_t)$.

Diversity Loss. Penalizes modes that are too similar, with minimum distance threshold $d_{\min} = 0.05$:

$$L_{\text{diversity}} = \sum_{i=1}^{K-1} \sum_{j=i+1}^{K} \max(0,\, d_{\min} - d_{ij}) \tag{3.7}$$

where d_{ij} is the average pairwise distance between modes i and j.

Soft Corridor Loss. The lateral deviation from the ground truth is computed as $d_{\perp,t} = (p_t - p_t^{\text{GT}}) \cdot n_t$, where n_t is the unit normal to the ground truth trajectory at time t, computed from the tangent direction as described in the lane keeping loss of Section 3.2. A graduated penalty is then applied in three zones:

- Safe zone ($|d_{\perp,t}| < 1.5\,\text{m}$): no penalty
- Warning zone ($1.5\,\text{m} \leq |d_{\perp,t}| < 2.5\,\text{m}$): quadratic ramp
- Danger zone ($|d_{\perp,t}| \geq 2.5\,\text{m}$): strong quadratic penalty

The thresholds are converted from meters to normalized coordinates. This formulation provides smooth gradients at the zone boundaries and reflects the physical structure of the road: small deviations within the lane are tolerated, while leaving the road is strongly penalized.

Total Loss. The complete training loss is:

$$L_{\text{total}} = L_{\text{WTA}} + L_{\text{endpoint}} + L_{\text{smooth}} + L_{\text{diversity}} + L_{\text{corridor}} \tag{3.8}$$

A portion of the weight on the corridor loss is based on a warm-up schedule that starts at 30% for the first ten epochs. Then, it increases linearly until it reaches 100% after forty epochs. That allows the model to learn the base structure of the trajectory first, then enforce road compliance.

Hyperparameters The key hyperparameters are summarized in Table 3.2.

Table 3.2. Trajectory Transformer training hyperparameters.

Parameter	Value
Model dimension d	256
Attention heads	8
Decoder layers	6
Feed-forward dimension	1024
Dropout	0.1
Modes K	3
Latent dimension	256
Batch size (effective)	512 (128×4)
Optimizer	AdamW
Peak learning rate	1.5×10^{-4}
Weight decay	0.05
Warmup / Total epochs	8 / 120
Gradient clipping	1.0

Learning Rate Schedule A cosine annealing schedule with linear warmup is employed, with peak learning rate $\eta_{\max} = 1.5 \times 10^{-4}$, minimum learning rate $\eta_{\min} = 0.01 \cdot \eta_{\max}$, and warmup over the first 8 epochs.

Inference Generating a trajectory requires a single forward pass:

1. Sample $z \sim \mathcal{N}(0, I_{256})$
2. Encode boundary conditions S and assemble memory
3. Decode to obtain K modes and probabilities
4. Select the final trajectory

There is no need for an iterative process to generate several trajectories for the same condition. The process described above can be repeated to generate trajectories with new samples of z. Each inference operation takes approximately seven milliseconds to generate one trajectory.

3.4 Dataset: SUMO Simulation of Piazza Adigrat

Vehicle trajectories were simulated using SUMO (Simulation of Urban MObility) traffic simulation software as the source of the training datasets. The road scenario selected for the simulation was the Piazza Adigrat, a common roundabout located in Milan, Italy. The geometry of the roundabout, including lane markings, entry and exit roads, and

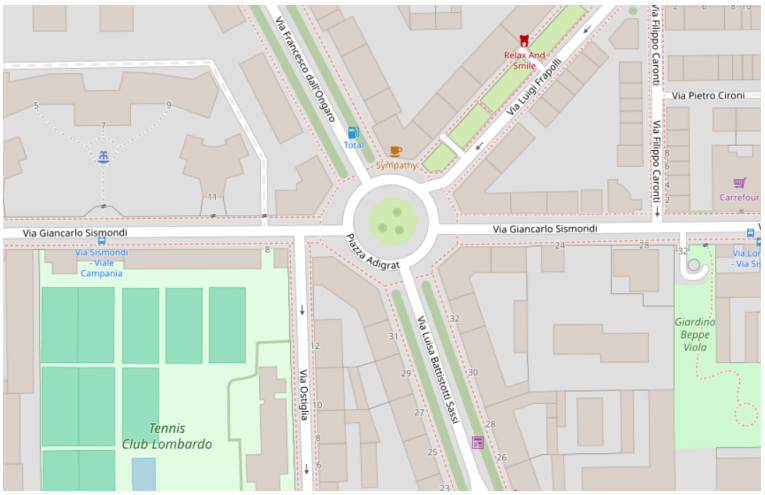

Figure 3.3. Piazza Adigrat from OpenStreetMap

traffic rules, was extracted from OpenStreetMap and converted into SUMO's network format.

There are five streets entering or exiting the roundabout, each entry or exit have different lanes associated with it. In generating vehicle trajectories, the default SUMO simulations and car-following models and lane-changing models were used, thus providing a realistic range of behavioral characteristics for acceleration, deceleration and lateral movement of vehicles within .

In this phase the simulations consist of a single vehicle free to move within a roundabout for a maximum of 60 and a minimum of 10 seconds. For the moment no other vehicles are considered. Vehicle position, speed, and heading were recorded in 0.5 second interval. The final dataset is composed of nearly 170000 complete trajectories, representing several possible entry-exit route combinations through the entire roundabout.This was then split, with 0.15% set aside as the validation set.

Coordinate System: In SUMO trajectories can be extracted in a global coordinate system (meters). The coordinates of every trajectory are used to define the spatial extent of the context. We can define a normalization standard selecting the maximum and minimum coordinates$[x_{\min}, y_{\min}]$ and $[x_{\max}, y_{\max}]$, for all simulations.

Feature Construction: For each vehicle simulation i, the time series of the states is processed into a standard format.*Position* (x_t, y_t) are the actor features indicating its global coordinates in meters.*Speed*$v_t = \sqrt{(\dot{x}_t)^2 + (\dot{y}_t)^2}$ is computed between each step. *Heading*: $\theta_t = \text{atan2}(\dot{y}_t, \dot{x}_t)$ that represents the orientation of the vehicle.

3.4.1 Normalization

Min-Max technique is utilized to normalize all the features to facilitate model training:

$$z_t^{\text{norm}} = \frac{z_t - z_{\min}}{z_{\max} - z_{\min}},$$

Since our evaluation is based on denormalized features all the normalization parameters $\{\cdot_{\min}, \cdot_{\max}\}$ for each feature are calculated from the entire training set and stored. The context vector $\mathbf{S}$ is normalized using the same spatial values as positions.

3.4.2 Padding and Sequence Length

The lengths of trajectories can vary based on how the vehicle behaved. For batch processing of the trajectories using fixed length inputs, all trajectories are padded to a maximum length of $T = 120$, representing a length of 60 seconds:

- The trajectories that have lengths shorter than T will be filled by repeating the last position, speed of zero, and the same heading as the last time step of the trajectory.
- The true length $L \leq T$ of each trajectory is recorded
- - A binary mask $\mathbf{m} \in \{0, 1\}^T$ indicates valid timesteps ($m_t = 1$ for $t < L$, $m_t = 0$ for $t \geq L$)

The use of this padding scheme will enable the networks to handle variable length sequences in an efficient manner, while still using the mask mechanism to be aware of the actual boundaries of the trajectory.

Chapter 4

Numerical Evaluation

4.1 Methodology

This chapter compares empirically the two generative approaches that have been developed for generating trajectories for a single vehicle the Trajectory GAN and the Trajectory Transformer using a set of $N = 5000$ random samples from the validation subset of the SUMO dataset. Two deterministic baseline models without learned parameters were used for reference: linear interpolation; cubic spline interpolation; both of which are conditioned on the boundary constraints and therefore generate one, fixed trajectory given each input. By comparing the generative models to these baselines we can determine if learning has generated some useful structure beyond plain geometric interpolation. All models had an identical evaluation pipeline as follows: spatial metrics were calculated in de-normalized coordinates (in metres); while kinematics metrics were generated by processing the speed and direction signals.

We focus on three dimensions:

- **Trajectory accuracy** Displacement-based error.
- **Lane adherence** Geometric consistency w.r.t. a lane corridor around the ground-truth centerline.
- **Physical realism** distribution-level similarity of motion variables and path efficiency.

The Transformer is a probabilistic generative model given the same boundary conditions it randomly selects a sample from a multivariate normal distribution with zero mean and identity covariance matrix ($z \sim N(0, I)$) and thus produces a different trajectory for each of its calls to evaluate; the evaluation of the Transformer therefore follows the best K method which is the most common method used for evaluating multimodal trajectory prediction models [6] [3]; with K = 10 independent samples for each input, the smallest ADE among all the 10 candidate trajectories is reported as the result for each case. The advantage is based on selecting the sample with the smallest amount of displacement error when compared to the ground truth. Therefore, an additional evaluation was done using a Transformer where $K = 1$, so that there would be one sample and thus no need for

selection. All metrics show little to no difference ($< 0.5\%$) from the previous evaluations, and therefore it can be concluded that the model will perform well regardless of whether the "best of K" samples are used. Additionally, the differences between the two models cannot be attributed to the number of samples selected for use, but rather as an artifact of how each were evaluated. The values in Table 4.1 were taken at $K = 1$ to maintain consistency with both the GAN and the deterministic baselines.

4.2 Evaluation Metrics

4.2.1 Trajectory Accuracy

The Average Displacement Error (ADE), which represents the average Euclidean distance in meters between predicted and true positions for each valid time step, is calculated with respect to the ground truth. The summary statistics provided are the mean, standard deviation, median, and the 95th percentile. The mean and the 95th percentile represent the most typical behavior and failures, respectively; whereas the standard deviation measures the distributional spread. A relative ADE is also provided, where it is obtained through normalization of the absolute mean ADE using the mean length of the trajectories in the dataset (280 m).

Lane Adherence Metrics

To assess how well each trajectory meets road or lane requirements, we calculate lateral deviation from the true trajectory projected along the local normal vector of the true trajectory. We define a corridor threshold τ and then use it to calculate:

- **Corridor violation rate (point-wise)**: Percentage of time steps when the lateral deviation is greater than τ.
- **Lane Keeping Rate (LKR)**: Percentage of time steps where the trajectory lies within the corridor, i.e., $100 -$ violation rate.
- **Fully in-lane (sequence-wise)**: Percentage of sequences with zero violations (i.e., there was never a time step when the lateral deviation exceeded τ).

4.2.2 Physical Realism Metrics

Two distinct approaches have been taken to evaluate the realism of the generated motion:

- **Path efficiency** Path efficiency has been assessed via the *path ratio* which is defined as the ratio of the predicted path length versus the true path length. In this way, ratios close to 1 indicate that the generated trajectory was able to cover a similar distance than the true trajectory. Only the median is reported to reduce the impact of extreme outliers.
- **Motion distribution similarity** via **Wasserstein-1 (W1)** distance computed over *i* speed and *ii* turning rate (derived from heading differences over time). Low

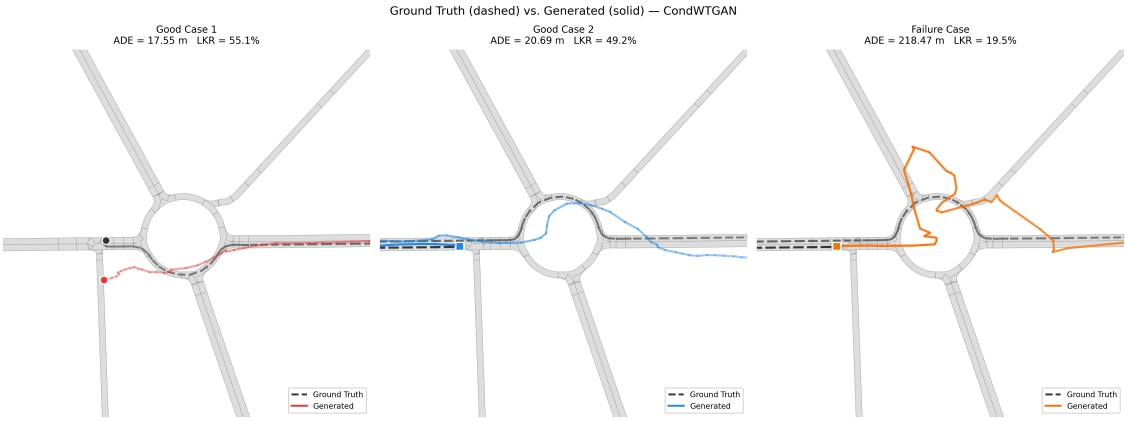

Figure 4.1. Qualitative trajectory comparison for the TGAN on Piazza Adigrat. Ground-truth trajectories are shown in dashed black; generated trajectories in solid colour. Even in the three good cases, the generated paths deviate substantially from the ground truth (ADE 17–29 m, LKR 24–55%), and the failure case (ADE = 218.47 m, LKR = 19.5%) exhibits a closed loop outside the road boundaries, a typical symptom of GAN training instability.

W1 distances indicate that the statistical characteristics of the motion are consistent with the statistical characteristics of the real data.

4.3 Quantitative Results

In Figure 4.1, we show four example trajectories created by the Transformer with a range of ADE percentiles as compared to the true trajectories. Regardless of the entry/exit configuration, the model produced geometrically consistent trajectories. However, the amount of deviation increased with increasing error percentiles.

Table 4.1 summarises the main quantitative metrics for all models. Transformer values refer to $K = 1$.

4.3.1 Trajectory Accuracy

As a result of using the Transformer, we were able to decrease the average absolute difference in Euclidean distance (ADE) from 26.52 meters (GAN) to 15.55 meters. Thus, we decreased the average ADE by about 40%. Additionally, the tail behavior improved significantly; the 95th percentile ADE decreased from 123.97 meters to 38.85 meters.

Although the TGAN achieves a slightly lower median ADE (9.67 m vs. 11.29m), its heavy-tailed error distribution (high standard deviation and very large 95th percentile) suggests that the model is not consistently reliable across the dataset. The genetic models clearly outperformed all all deterministic baselines with a large margin of performance (ADE $>$ 34 m for each interpolation method), indicating that the learned models were able to learn structure beyond just simple geometric interpolation.

Table 4.1. Comparative evaluation of generative models and deterministic baselines (N=5,000 samples, single-vehicle setting). ↑ higher is better, ↓ lower is better. Bold is best per row. Mean trajectory length $280m$. W1 = Wasserstein-1 distance.

Metric	**Transformer**	**GAN**	**Linear**	**Spline**
Trajectory Accuracy				
ADE mean (m) ↓	**15.55**	26.52	34.85	36.22
ADE median (m) ↓	11.29	**9.67**	26.38	27.88
Rel. ADE (%) ↓	**5.7**	9.5	12.4	12.9
ADE p_{95} (m) ↓	**38.85**	123.97	—	—
Lane Adherence ($\pm 2\,m$ corridor)				
LKR (%) ↑	**91.98**	44.83	51.84	23.16
Lateral dev. mean (m) ↓	**1.16**	8.55	12.76	15.89
Corridor violations (%) ↓	**8.01**	55.17	48.16	76.84
Fully in-lane (%) ↑	**41.18**	0.54	22.74	4.22
Physical Realism				
Path ratio (med.) ≈ 1	**1.021**	1.66	0.92	0.93
W1 turning rate ↓	**0.025**	0.617	9.355	9.035
W1 speed ↓	**0.077**	0.391	2.617	2.558

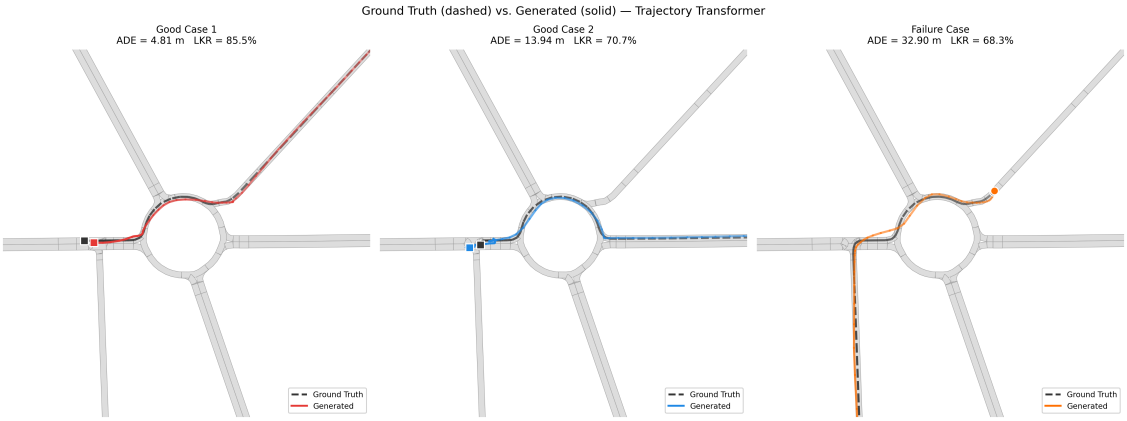

Figure 4.2. Qualitative trajectory comparison for the Trajectory Transformer on Piazza Adigrat. In all three good cases the generated trajectory closely follows the ground truth both geometrically and in terms of lane compliance. The failure case shows a partial deviation near the roundabout exit, yet remains within a plausible corridor.

4.3.2 Lane Adherence

Both Lane Adherence Metrics were also the largest source of separation between the two methods. The Transformer had a lane keeping ratio of 91.98% 2 m corridor while the GAN was at 44.83%. This results in a reduction in corridor violations for the Transformer (from 55.17% to 8.12%) and a significant decrease in the average lateral distance off the center line of the road from 8.55 m to 1.16 m). The Transformer was consistently on the road center line while the GAN drifted significantly in most cases.

Fully in-Lane Metric shows the quality differences in robustness. Only 0.54% of the trajectories generated by the TGAN were fully within the corridor while 41.18% of the trajectories generated by the Transformer were fully within the corridor. However, it should be noted that the fact that approximately 59% of the transformer's trajectories violated the corridor at least one time indicates that the goal of full geometric compliance has not yet been achieved, thus there is still opportunity for improvement.

4.3.3 Physical Realism

The median path ratio for the Transformer is approximately 1 (1.021) which indicates that the lengths of the paths produced by the model are nearly equal to the lengths of the true paths. In contrast, the median path ratio for the TGAN is 1.66, indicating that the paths produced by the model were on average 66% longer than their ground truth counterparts. It is possible that this is due to instability in the training process as the generator may produce paths with unnecessary redundancy or oscillation and therefore increase the overall path length without accurately representing real world vehicle dynamics.

In addition, the distributional similarity metrics support the superiority of the Transformer over the GAN. Specifically, the W1 distance was decreased significantly for both

turning rate (from 0.617 to 0.025) and speed (from 0.391 to 0.077). Therefore, it can be inferred that the Transformer model does not only generate paths that are geometrically accurate, but also captures realistic kinematics.

4.4 Discussion

To summarize, we found significant numerical evidence supporting the Transformer as the preferred generative model for producing individual vehicle trajectories on a roadway where constraints exist. Although the GAN may occasionally outperform the Transformer in terms of its median displacement error, the GAN's performance will likely be poor (due to instability regarding error distributions, low LKR, and high path ratios), therefore, we can conclude that the model failed to develop a general representation of the domain.

On the contrary, the Transformer provided significantly improved reliability of all percentiles of error, better geometric consistency, physics-based simulation, and more optimized trajectories. Furthermore, the results indicated that there existed less than a 0.5% difference when comparing K = 1 to K = 10 for all metrics; indicating that the variability of samples generated by the model is relatively low, and the model produces good quality trajectories regardless if the best sample is selected using a reference or not.

Although some limitations remain, they include approximately 59% of the generated trajectories produced by the Transformer violated the 2 m corridor at least once, suggesting the model has not yet fully complied with geometry requirements. Second, this study does not examine how the performance of the model varies by scenario (i.e., road curvature, initial speed, length of the trajectory). A subgroup analysis may help identify what types of scenarios pose the greatest challenges for the model and enable targeted fine tuning efforts. Third, the physical realism metrics presented were calculated using only one sample for each input and should be viewed as averages; for safety-critical applications, it would be beneficial to evaluate the inter-sample variation of kinematic parameters.

Future research will assess how suitable the output of the Transformer can be as an initial starting point for developing solutions that need coherent paths in space for other applications including, but not limited to multi-agent scenarios and a methodical process for assessing the safety of autonomous vehicles.

The training dynamics of the two models provide a direct visual explanation for the observed performance gap. As shown in Figure4.3, the Trajectory Transformer converges smoothly over 250 epochs, with a stable and contained gap between training and validation loss throughout. The CondWTGAN, by contrast, exhibits a markedly different behaviour: Figure4.4 shows recurring instability spikes in the generator loss at multiple points during training, whilethe discriminator loss collapses near zero on the training set as the critic overfits to the training distribution. This pattern is consistent with the well-documented failure modes of adversarial training, and directly explains the heavy-tailed ADE distribution and high corridor violation rate reported in Table 4.1.

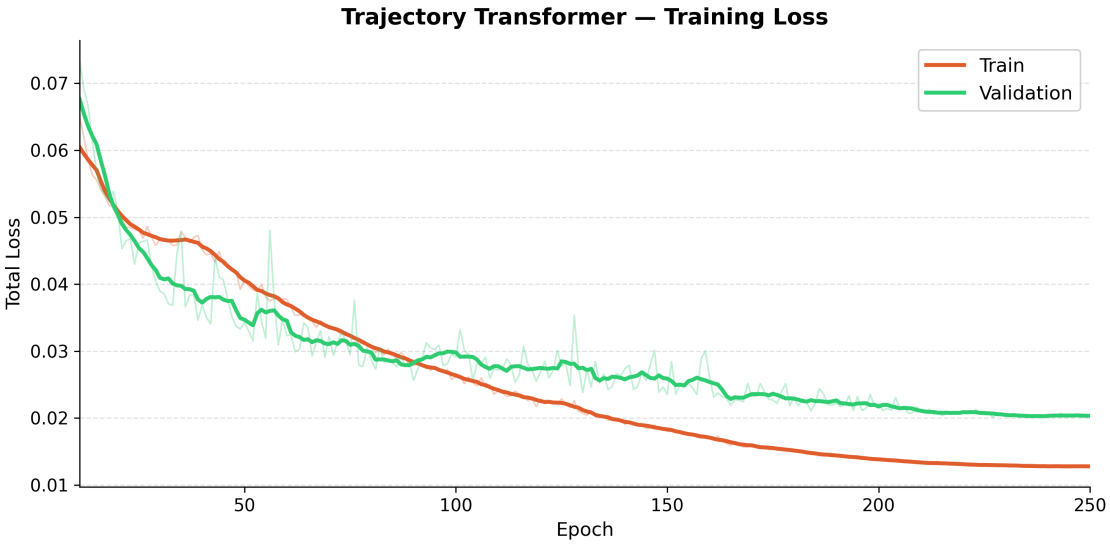

Figure 4.3. Training and validation loss of the Trajectory Transformer over 250 epochs. The model converges stably, with a contained gap between training and validation loss and mild overfitting emerging only in the final stages.

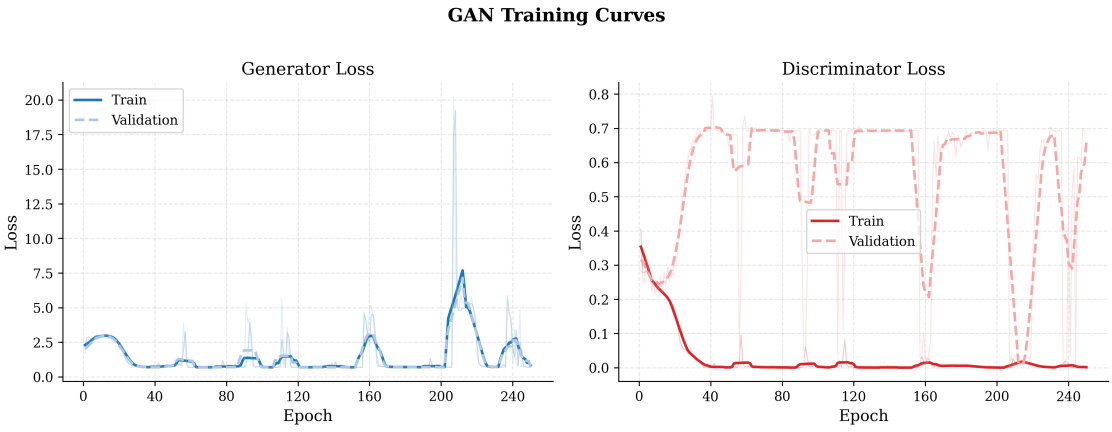

Figure 4.4. Training and validation loss of the TGAN over 250 epochs. The generator loss (left) exhibits recurring instability spikes throughout training, while the discriminator loss (right) collapses near zero on the training set. This pattern directly explains the heavy-tailed error distribution and low lane adherence reported in Table 4.1.

4.5 Extension to Multi-Vehicle Context-Conditioned Generation

4.5.1 From Ego-Only to Multi-Agent Conditioning

The Transformer-based trajectory generator was formulated in an ego-only setting, where the model predicts a single vehicle trajectory conditioned on boundary constraints and latent variables. While this formulation allows accurate modeling of geometric and physical constraints, it does not explicitly account for interactions with other agents.

In this section, the model is extended to a multi-vehicle setting by incorporating the state and behavior of surrounding vehicles as additional conditioning inputs. This extension preserves the original architecture and autoregressive decoding mechanism, and introduces interaction modeling exclusively through an enriched conditioning representation.

4.5.2 Multi-Vehicle Dataset Construction

The multi-vehicle dataset was generated using the same SUMO simulation environment described in Section 3.3, applied to the Piazza Adigrat roundabout in Milan. Unlike the single-vehicle setting — where each simulation contained a single ego vehicle traversing the roundabout in isolation — the multi-vehicle simulations activate the full SUMO traffic demand model, producing concurrent vehicle streams across all five entry and exit points of the roundabout. Traffic demand was configured using SUMO's `randomTrips` generator, ensuring a range of traffic densities across simulations. The default car-following model (Krauss) and lane-change model (LC2013) were retained, consistent with the single-vehicle setup, so that the behavioral dynamics of context vehicles are governed by the same simulator used to generate the ego trajectories.

Ego and context vehicle definition. Ego and Context Vehicles. For each simulation run, one vehicle is designated as the ego vehicle: the agent whose path the model is trained to create. All vehicles in the scene that are active during the ego's path creation are defined as context vehicles. A context vehicle is added to the conditioning set if its path overlaps temporally with the ego's path, i.e., if it exists in the simulation for at least one step within the ego's $T = 120$ step window. Each context vehicle's path is referenced to the ego's path reference system: each context vehicle's path is expressed as a sequence of states $(x_t, y_t, v_t, \theta_t)$ over the same T steps. If a context vehicle enters or leaves the scene during the ego's path, the context vehicle's path is extended with zeros for the missing steps and the corresponding flag in a binary per-vehicle validity mask is set.

Dataset format. Each training sample is composed of the following tensors:

- $\mathbf{X} \in \mathbb{R}^{T\times 4}$: ego trajectory (x, y, v, θ)
- $\mathbf{S} \in \mathbb{R}^4$: boundary conditions $(x_{\text{start}}, y_{\text{start}}, x_{\text{end}}, y_{\text{end}})$
- $\mathbf{C} \in \mathbb{R}^{K\times T\times 4}$: context vehicle trajectories, with $K = 6$ slots

- $\mathbf{M} \in \{0,1\}^K$: binary padding mask ($M_k = 1$ if slot k is empty)
- $L \in \mathbb{N}$: valid length of the ego trajectory

The maximum number of context vehicles per sample is fixed at $K = 6$, chosen to cover the large majority of observed scenarios. Samples with fewer than six active context vehicles have their unused slots zeroed and flagged in $\mathbf{M}$, following the same padding and masking convention described in Section 3.3.2 for ego trajectories. All features are normalized using the same min-max parameters computed on the single-vehicle training set, ensuring consistency across the two settings.

4.5.3 Architectural Extension

The key modification consists in augmenting the conditioning space with a dynamic multi-agent context. Let $\mathbf{X}_{\text{ego}}$ denote the ego trajectory and $\mathbf{X}_{\text{ctx}} = \{\mathbf{X}^{(k)}\}_{k=1}^{K}$ the set of context vehicle trajectories. The model now conditions on both:

$$p(\mathbf{X}_{\text{ego}} \mid \mathbf{S},\, \mathbf{z},\, \mathbf{X}_{\text{ctx}}) \tag{4.1}$$

Temporal Context Encoding. The trajectories of surrounding vehicles are processed through a dedicated `TemporalContextEncoder (TCE)`. This module encodes the temporal evolution of each context vehicle while preserving sequential dependencies. Given a set of context trajectories, each represented as a sequence of states over time, the encoder applies a temporal Transformer to extract a sequence of context embeddings:

$$\mathbf{H}_{\text{ctx}} = \texttt{TCE}(\mathbf{X}_{\text{ctx}},\, \mathbf{M}) \in \mathbb{R}^{(K \cdot T_{ds}) \times d} \tag{4.2}$$

Temporal information is preserved through sinusoidal positional encodings, ensuring that the ordering of observations is maintained. Padded slots (where $M_k = 1$) are zeroed prior to encoding and excluded from the attention computation via the padding mask, ensuring that empty vehicle slots do not contribute to the conditioning representation. This design allows the model to capture both short-term interactions (e.g., proximity) and longer-term behavioral patterns (e.g., yielding or merging).

Tokenization and Memory Construction. The encoded context is transformed into a set of tokens that can be consumed by the decoder. These tokens are concatenated with the existing conditioning tokens used in the ego-only formulation.Every context vehicle provides a temporal tokenization of $Tds = 24$ temporal tokens that are derived from its 120-step time series down sampled by a stride of 5 via strided convolution, which allows the decoder to attend to the temporal behavior of each nearby vehicle individually as opposed to a single representation of the pool of vehicles The final memory structure used by the decoder is therefore:

$$\mathbf{M} = [\mathbf{h}_{\text{ego}};\ \mathbf{h}_z;\ \mathbf{H}_{\text{ctx}}] \in \mathbb{R}^{(2+K \cdot T_{ds}) \times d} \tag{4.3}$$

where $\mathbf{h}_{\text{ego}}$ encodes boundary and ego-related conditioning information, $\mathbf{h}_z$ represents the latent multimodal embedding, and $\mathbf{H}_{\text{ctx}}$ contains the encoded representations

of surrounding vehicles. Importantly, the decoder architecture remains unchanged. The extension is entirely realized by enriching the memory through additional context tokens.

Interaction Modeling. This design enables interaction modeling through attention mechanisms. During decoding, the cross-attention layers can dynamically attend to context tokens, allowing the model to adapt the generated trajectory based on the behavior of surrounding agents. This approach avoids explicitly modeling pairwise interactions or introducing specialized interaction modules. Instead, interactions emerge implicitly through attention, enabling flexible and scalable modeling of multi-agent scenarios.

4.5.4 Training Setup

Although the multi-vehicle model used the same training parameters for hyperparameters as the single-vehicle Transformer shown in Table 3.2, there were two adjustments made to these training parameters. The effective batch size for the multi-vehicle model was decreased to 256 (as opposed to the 128 vehicle contexts per batch used for the single-vehicle model. This reduction in effective batch size was required due to the greater memory requirements for the context tensor$\mathbf{C} \in \mathbb{R}^{B\times K\times T\times 4}$ for the multi-vehicle model. Additionally, the corridor loss warm-up period for the multi-vehicle model was increased from 40 epochs to 60 epochs. This increase allowed the model to develop an awareness of how to behave when interacting with other vehicles over a longer period of time prior to the imposition of the strict road compliance constraints.

The total training loss retains the same structure described in Section 3.2.3:

$$L_{\text{total}} = L_{\text{WTA}} + L_{\text{endpoint}} + L_{\text{smooth}} + L_{\text{diversity}} + L_{\text{corridor}} \tag{4.4}$$

No interaction-specific loss term was added: the model is not explicitly penalized for proximity to context vehicles during training. Collision avoidance, if it emerges, does so as an implicit consequence of learning to imitate ground-truth ego trajectories that were themselves generated in the presence of surrounding traffic.

4.5.5 Collision Modeling

In the multi-agent setting, evaluating safety requires explicit consideration of collisions and proximity violations. Collisions are computed based on the minimum distance between the ego vehicle and any context vehicle.

Let $d_t^{(k)}$ denote the distance at time t between the ego vehicle and context vehicle k. A collision condition is defined as:

$$d_t^{(k)} < \tau \tag{4.5}$$

where τ is a distance threshold. Three complementary metrics are used:

Sequence-wise Collision Rate. A trajectory is classified as colliding if:

$$\min_{t,k} \; d_t^{(k)} < \tau \tag{4.6}$$

This metric captures whether a collision occurs at any point in the trajectory.

Timestep-wise Collision Occupancy. This metric measures the percentage of timesteps where the distance constraint is violated:

$$\frac{\#\{t, k : d_t^{(k)} < \tau\}}{\text{total valid timesteps}} \tag{4.7}$$

It reflects how persistent unsafe interactions are over time.

Global Collision Counts. The number of timestep-level collision counts under the specified threshold represents the total number of time steps that were collision-free (no global collisions were reported), and the total number of timesteps that contained valid data (those that had a complete record of the data) for all ego-context pair combinations in the evaluation set (1399917 total). These are global metrics that will allow us to evaluate model performance relative to safety.

Multiple thresholds are considered to capture different levels of interaction:

- 0.5 m: physical collision
- 1.0 m: near-collision
- 3.0 m: interaction proximity
- 5.0 m, 7.0 m: broader interaction zones

This multi-scale evaluation is critical, as ego-only models do not enforce interaction constraints, whereas multi-agent models must balance trajectory realism with safety.

4.5.6 Quantitative Results

The evaluation results are organized into three groups: trajectory accuracy, lane adherence across multiple corridor thresholds, and collision safety. The evaluation set consists of $N = 5\,000$ random samples from the validation subset of the multi-vehicle SUMO dataset.

Figure 4.5 shows three representative multi-vehicle scenarios generated by the model. In each case, the ego vehicle (coloured trajectory) successfully traverses the roundabout while the context vehicles (black) follow their ground-truth paths. The generated trajectories remain geometrically plausible and lane-consistent across diverse entry–exit configurations and traffic densities, confirming qualitatively the quantitative results reported below.

Trajectory Accuracy and Lane Adherence

Table 4.2 shows the principal metrics evaluating the accuracy and lane adherence of the model. In addition to the two metrics used for the single-vehicle model (the Mean Absolute Error or MAE and the final displacement error), two new metrics have been

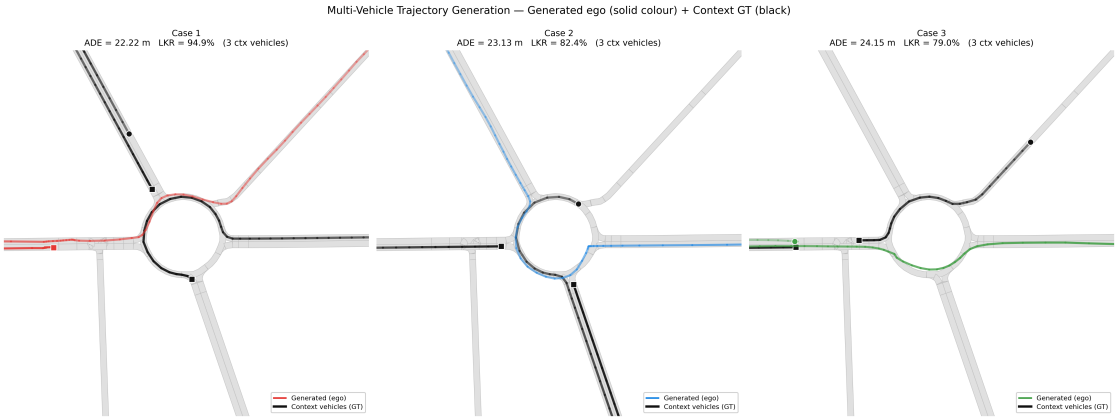

Figure 4.5. Qualitative trajectory comparison for the multi-vehicle context-conditioned Transformer on Piazza Adigrat. The generated ego trajectory (solid colour) is shown alongside the ground-truth trajectories of context vehicles (black). In all three cases the ego vehicle navigates the roundabout while remaining geometrically consistent with the surrounding traffic, demonstrating that the model implicitly adapts its path to the presence of context agents without any explicit collision avoidance loss.

added for the multi-vehicle model. The first metric is the final displacement error (FDE), which calculates the Euclidean distance between the position at the last valid timestep of the generated trajectory and the corresponding position in the ground truth trajectory. The second metric is the mean jerk, which evaluates the smoothness of the generated trajectory by calculating the average rate of change of acceleration.

Table 4.2. Trajectory and lane adherence metrics for the multi-vehicle context-conditioned Transformer ($N = 5\,000$ samples). ↑ higher is better, ↓ lower is better.

Metric	**Mean**	**Median**
ADE (m) ↓	15.91	11.66
Mean lateral deviation (m) ↓	1.19	0.82
Max lateral deviation (m) ↓	5.92	2.40
$\text{LKR}_{2\text{m}}$ (%) ↑	91.82	96.77
$\text{Corridor violation}_{2\text{m}}$ (%) ↓	8.18	3.23
Mean jerk (m/s^3) ↓	1.09	1.04

Both the lane adherence metrics of the multi-vehicle model are similar to the corresponding metrics of the single-vehicle model. Specifically, both models achieved an $LKR2m$ value of 91.82%, and both models exhibited a mean lateral deviation of approximately 1.19 meters. This suggests that the richer contextual conditioning provided by the multi-vehicle model did not negatively impact the performance of the corridor loss mechanism used to evaluate the adherence of the generated trajectories to the lanes.

The mean jerk of 1.09 m/s^3 of the multi-vehicle model is also consistent with typical values measured during normal urban driving. Furthermore, the mean jerk value indicates

that the generated trajectories do not exhibit abrupt changes in velocity.

The path ratio distribution requires careful interpretation. While the median is 1.022, indicating that generated paths are nearly equal in length to the ground truth the mean is inflated to 9.7 by a small number of extreme outliers (max = 15 531). These outliers correspond to failure cases in which the autoregressive decoder enters a degenerate loop, generating a trajectory that circles repeatedly within the roundabout before exiting. Although rare (the 99th percentile is 1.43), these cases highlight a failure mode of autoregressive generation in constrained environments, and are consistent with the loop cases reported for the single-vehicle GAN in Section 4.3.3. For this reason the median, rather than the mean, is reported for the path ratio.

Lane Adherence Across Multiple Corridor Thresholds

Table 4.3 reports the corridor violation rate and sequence-level violation statistics across five thresholds, providing a more granular picture of lateral compliance than the single ±2 m threshold reported in Section 4.3.

Table 4.3. Lane adherence metrics across corridor thresholds. Point-wise violation rate: percentage of timesteps that exceed threshold. Sequence violation: percentage of trajectories with at least one violation. Sequence severe: percentage of trajectories where more than 50 percent of timesteps violate.

Threshold	**Point-wise viol. (%)**	**Seq. viol. (%)**	**Seq. severe (%)**
±1.0 m	30.74	94.93	92.56
±1.5 m	12.84	76.17	61.34
±2.0 m	8.18	59.36	42.00
±2.5 m	6.12	47.86	32.51
±3.0 m	4.92	39.94	26.45

The point-wise violation rate 8.18% and LKR 91.82% appear acceptable at the ±2 m corridor used in the single-vehicle evaluation. However, 59.36% of trajectories violate the ±2 m corridor at least once, suggesting that long-term compliance over an entire 60 second trajectory remains difficult. Tightening the threshold to ±1.5 m increases the sequence violation rate to 76.17%, while widening the threshold to ±3.0 m decreases it to 39.94%. This gradient indicates most violations are marginal in magnitude: they are captured by the ±2 m threshold but disappear at ±3.0 m, suggesting lateral deviations are concentrated between 2 – 3 m rather than large off-road excursions.

Collision Safety

Table 4.4 reports collision metrics at five distance thresholds. The global occupancy column is computed over a total of 1 399 917 valid ego–context timestep pairs across the evaluation set.

A major difference in how sequence-wise vs. time-step-wise metrics are evaluated is found. When using a "strict" threshold (i.e., 0.5 m and 1.0 m) for collisions or occupancy, both the collision rate and occupancy of the model are almost non-existent. Only 36 and

Table 4.4. Collision safety metrics at increasing distance thresholds. Collision rate: percentage of trajectories with at least one event. Occupancy: percentage of valid timestep pairs under threshold. Global events: total count over 1 399 917 valid pairs.

Threshold	Collision rate (%)	Occupancy (%)	Global occ. (%)	Global events
0.5 m	0.14	0.0021	0.0026	36
1.0 m	0.48	0.010	0.011	158
3.0 m	2.58	0.083	0.090	1 258
5.0 m	15.46	0.436	0.426	5 969
7.0 m	31.70	1.138	1.043	14 606

158 timestep-level events were reported as collisions out of nearly 1.4 million timestep levels and only corresponded to an occupancy level of 0.003% and 0.011% respectively. This near-zero value clearly indicates that the model seldomly produces ego trajectory's that physically overlap with context vehicles.

As the threshold increases, the collision rate grows significantly, reaching over 30% at 7 m. However, the corresponding occupancy remains low 1.14%, indicating proximity events are typically brief. A threshold of 7 m is equivalent to approximately 3.5 vehicle lengths — a gap frequently observed in normal roundabout traffic — so most flagged events at this threshold reflect realistic following and merging behavior rather than model failures. It is important to note that no explicit collision avoidance loss was used during training; instead, the model learns to avoid context vehicles implicitly by imitating ground truth ego trajectories that were themselves generated in the presence of surrounding traffic. The low collision rate at strict thresholds suggests that this implicit signal is sufficient for basic safety, although it does not provide formal guarantees.

4.5.7 Discussion

The extension to multi-vehicle conditioning enables the model to incorporate interaction-aware behavior without modifying the core architecture. The results demonstrate that the model is capable of maintaining low collision rates at strict thresholds, while allowing realistic proximity interactions.

The multi-vehicle model achieves trajectory accuracy nearly identical to the single-vehicle Transformer, confirming that attention-based conditioning over up to six context vehicles is a lightweight and non-disruptive mechanism. The path ratio outliers and the sequence-level corridor violations highlight the same limitations observed in the single-vehicle evaluation: while the model performs well in the majority of cases, a small fraction of trajectories enter degenerate patterns that are not corrected by the current loss formulation.

While there are challenges in capturing the multi-agent dynamics of an environment due to potential outlier data points or a large number of violations of the sequencing rules as defined by the simulation parameters, the use of the Transformer's ability to condition on past interactions using an attention-based method provides the capability to capture those short term interactions. The question remains whether it will be possible

to maintain a level of global consistency among all agents when interacting over long periods of time. These results confirm that there is a high degree of additional complexity associated with expanding trajectory generation models that include multiple agents, specifically how one balances realism, diversity and safety.

Chapter 5

Conclusion

This thesis investigated the use of deep generative models for the synthesis of realistic vehicle trajectories in a roundabout environment. The core research question was whether generative models could produce trajectories that are physically plausible, geometrically consistent, and diverse enough to serve as synthetic data for testing and validating autonomous driving systems. Two fundamentally different generative paradigms were implemented, trained, and evaluated on the same dataset of SUMO-simulated trajectories at Piazza Adigrat, Milan.

5.1 Summary of Contributions

The first contribution of this thesis is a comparative implementation of two generative architectures for conditional trajectory synthesis. The **Trajectory GAN** (TGAN) combines a Transformer-based generator with a Relativistic Pairing GAN objective, StyleGAN-inspired style injection, and domain-specific physical and road compliance losses. The **Trajectory Transformer** utilizes a direct prediction method similar to [7]. In this model, the full trajectory is generated within one forward pass through a Transformer based decoder that is conditioned on a stochastic latent vector, as well as the boundary constraints.

The second contribution is the quantitative evaluation framework itself, which combines trajectory accuracy metrics, lane adherence metrics, and distribution-level realism metrics into a unified evaluation pipeline. This framework provides a more complete picture of generative model quality than displacement error alone, and can be reused in future work on trajectory synthesis.

A third aspect of this research has been the expansion of the Transformer-based model into multi-vehicle scenarios. The authors have demonstrated that the addition of context trajectories that are encoded from the conditioning memory allows the model to capture interaction-aware behavior without having to modify the core decoder architecture. Therefore, the authors have shown that attention-based conditioning is a flexible and scalable mechanism for generating multi-agent trajectories.

5.2 Main Findings

The quantitative results reported in Chapter 4 support several clear conclusions.

First, the Transformer-based approach substantially outperforms the GAN across almost all evaluation dimensions. It reduces mean Average Displacement Error by approximately 40% (from 26.52 m to 15.55 m) and reduces corridor violations from 52.90% to 8.01% within a ± 2 m lane corridor. Most importantly, the 95th percentile ADE drops from 123.97 m to 39.44 m, indicating that the Transformer produces far fewer catastrophic failures. Distribution-level metrics confirm this advantage: the Wasserstein-1 distance on turning rate decreases from 0.617 to 0.025, and on speed from 0.391 to 0.077, suggesting that the Transformer learns motion dynamics that are statistically close to those of real simulated trajectories.

Second, the GAN exhibits the training instability and heavy-tailed error distributions that are well-documented in the GAN literature [8]. Despite achieving a slightly better median ADE ($9.67m$ vs. 11.67 m), the GAN's high variance and poor lane adherence make it an unreliable generator for safety-critical applications. The adversarial training objective, while powerful in principle, proved difficult to balance against the physical and road compliance constraints required by the trajectory generation task.

Additionally, the expansion to multi-vehicle conditioning demonstrates that the Transformer architecture is capable of incorporating the state of nearby vehicles while still maintaining accurate single-vehicle trajectory generation. In fact, collision rates at tight distance thresholds ($0.5m$ and $1.0m$) are very low (0.135% and 0.48%, respectively). This indicates that the model does not generate trajectories which frequently physically overlap with context vehicles.

5.3 Limitations

Several limitations exist regarding this study. The trained and tested models were only trained and tested using synthetic data created by SUMO from a single roundabout. Although SUMO does create realistic car-following and lane-change behaviors in the simulation, the actual simulated trajectory created will not have all of the complexity and variability found in real-world sensor data. It is therefore unclear whether the models can be generalized to real-world datasets such as Waymo Open Motion [6] or Argoverse [3].

Since Deterministic Baseline models such as Linear Interpolation, Spline Interpolation were presented in the Single-Vehicle Evaluation, no Equivalent Baselines were used in Multi-Vehicle Setting, since it is much harder to define a Geometric Interpolation. Therefore, without a reference model such as simply interpolating between entry and exit points of the intersection; it is not possible to compare the quality of the synthesized trajectories relative to each other but rather to evaluate the relative quality of the two models used in comparison to one another.

Finally, the multi-vehicle extension has been developed preliminarily. The number of vehicles to include in the contextual input, the treatment of the size of the set of agents, and the joint training dynamics of the TemporalContextEncoder have not been completely ablated and therefore, remain as open questions related to the stability and

social coherence of the interaction modeling component.

5.4 Future Work

This research has produced many promising avenues for further study.

The first and most obvious area of investigation will be to test the Transformer model with actual (real world) trajectory data. It will be important to adapt the normalization pipeline and the road constraint loss function to deal with the noisy, missing sensor information that will likely occur in the real world. Testing whether the Transformer model trained with SUMO data can generalize to a "roundabout" environment in the real world would also be an important area of investigation.

The second area of investigation will be to use the synthesized scenarios generated by the Transformer model as a simulation backend in SUMO. Exporting the synthetic trajectories from the Transformer model into a format compatible with SUMO would enable the evaluation of whether the use of synthetic scenarios as training data improves the performance of autonomous vehicles. This would enable a "loop" between generating synthetic scenarios and validating their usefulness as training data.

The third area of investigation is related to the multi-vehicle architecture used in this work. Future work should investigate mechanisms for modeling interactions between multiple vehicles, such as using graph based representations or using factorized attention over pairs of agents. These architectures could be able to provide stronger guarantees of avoiding collisions while maintaining social consistency than the implicit attention-based conditioning mechanism employed in this work. Additionally, training the model on more dense traffic scenarios and measuring the quality of the generated scenarios using metrics that are designed to measure the realism of multi-agent interactions, such as scene level consistency and interaction plausibility, would provide a more complete picture of its capabilities.

Finally, investigating the use of diffusion-based generative models for trajectory synthesis represents a natural extension of the work described here. Recently, diffusion models have demonstrated high-quality results for motion generation tasks, and they may be capable of producing greater diversity and physical plausibility than either the GAN or Transformer approaches employed in this thesis, although the cost will be significantly increased inference time.

5.5 Closing Remarks

Overall, this thesis has demonstrated that deep generative models, including the use of architectures based on Transformers are a viable and effective method for generating vehicle trajectories within constrained road environments. Furthermore, the results of this thesis have clearly shown the benefits of using likelihood-based sequential generation as opposed to adversarial training within this specific domain, and identified the remaining open challenges of modeling stable and socially coherent interactions among multiple agents. Additionally, this dissertation provides a basis for continued research into the development of data-driven methods for generating traffic scenarios to validate autonomous

driving systems.

Bibliography

[1] Shekoofeh Azizi, Basil Mustafa, Fiona Ryan, Zachary Beaver, Jan Freyberg, et al. Synthetic data from diffusion models improves imagenet classification. *arXiv preprint arXiv:2304.08466*, 2023.

[2] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11621–11631, 2020.

[3] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Seong Joon Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays. Argoverse: 3d tracking and forecasting with rich maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8748–8757, 2019.

[4] Rui Chen et al. A unified framework for generative data augmentation. *arXiv preprint arXiv:2310.00277*, 2023.

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

[6] Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chao Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Benjamin Sapp, Charles R. Qi, Yin Zhou, Zhenyang Yang, Alex Chouard, Pei Sun, Jiquan Ngiam, Vijay Vasudevan, Alex McCauley, Jonathan Shlens, and Dragomir Anguelov. Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9710–9719, 2021.

[7] L. Feng, Q. Li, Z. Peng, S. Tan, and B. Zhou. Trafficgen: Learning to generate diverse and realistic traffic scenarios. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2023.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, volume 27, 2014.

[9] Y. Huang, A. Gokaslan, V. Kuleshov, and J. Tompkin. The gan is dead; long live the gan! a modern baseline gan. In *Advances in Neural Information Processing Systems*, 2024.

[10] Mohamed Ibrahim et al. Generative ai for synthetic data generation: A systematic review. *arXiv preprint arXiv:2407.00116*, 2024.

[11] Alexia Jolicoeur-Martineau. The relativistic discriminator: A key element missing from standard gan. *arXiv preprint arXiv:1807.00734*, 2018.

[12] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. In *Advances in Neural Information Processing Systems*, 2020.

[13] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[14] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[15] Pablo A. Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *Proceedings of the IEEE Intelligent Transportation Systems Conference*, 2018.

[16] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *Proceedings of the International Conference on Machine Learning*, pages 3481–3490, 2018.

[17] Olof Mogren. C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*, 2016.

[18] Balakrishnan Varadarajan et al. Multipath++: Efficient information fusion and trajectory aggregation for behavior prediction. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2022.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[20] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.