Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Vision Lab

# DiffuseDrive
## An Exploration of Diffusion Models for Autonomous Driving

Robot Learning Course Project - Spring 2023

Dennis Vilgertshofer
Department of Information Technology and Electrical Engineering

Advisors: Yan Wu, Dr. Yifan Liu
Supervisor: Prof. Dr. Fisher Yu

04.08.2023

# Abstract

Autonomous navigation has gained increasing attention in recent years. However, much of the recent work published on the topic focuses on increases in driving performance, using increasingly complex systems. As such it often becomes hard to pinpoint the root cause of arising issues. While some of the recent work has also considered the interpretability concerns of autonomous driving, they aren't end-to-end and also rely on many manually designed heuristics. With advances in generative modeling, namely Diffusion models, in the area of generating high-fidelity images as well as their promise in decision-making processes, they may potentially be included in autonomous driving pipelines.
With this project, an end-to-end autonomous driving architecture is designed that bases its decision making process on future trajectories that are sampled using a Diffusion model. The Diffusion process is conditioned using information about the past trajectory of the vehicle, as well as additional information. While the model struggles with highly complex traffic situations, it is capable of turning as well as following and changing lanes in simpler scenarios. Despite its issues, it shows promise regarding the inclusion of diffusion models in autonomous navigation systems and may well lead to better interpretability in the future.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In recent years, Artificial Intelligence (AI) has taken the world by storm. Their ability to make sense out of seemingly unrelated data allows them to effectively work on all kinds of problems. Especially in the Computer Vision (CV) community, there has been explosive progress in recent years. The introduction of diffusion models has lead to new heights in the generation of high-resolution, high-fidelity images. A prominent example of a model for text-to-image generation is OpenAI's DALL-E 2[1].
Despite their far-reaching impact, they are not a panacea. They often work in a highly-specialized setting with a clear problem structure, having trouble generalizing to unseen data.
One setting where AI has still struggled to find its footing is autonomous navigation. Obeying traffic rules as well as keeping an overview of surrounding vehicles and obstacles leads to an incredibly complex problem set, with safety concerns requiring autonomous driving systems (ADS) to have a near-zero fault tolerance. Due to the aforementioned safety concerns, it is crucial to know where the root cause of arising problems lie. Yet it is often hard to pinpoint what exactly went wrong.

## 1.1 Motivation and Goal

Much of the current work covering autonomous navigation rests their focus on improving the efficacy of the system. This leads to models that perform well, yet their lack of interpretability leads to difficulties pinpointing the root cause in case of failure. The recently released InterFuser[18] ADS, which currently takes the number one spot on the leaderboard of the popular CARLA autonomous driving benchmark[2][7], alleviates this issue by providing intermediate interpretable features during operation. Since these interpretable features are generated in a highly abstract level, deployment in a real-world scenario is next to impossible due to the need to provide annotated data in this highly abstract level for model training.
The goal of this project is to leverage the generative abilities of diffusion models to create an end-to-end ADS that provides intermediate interpretable results without the need for costly annotations. Rather than planning in such an abstract space, the goal is to use diffusion models to generate interpretable features that lie in the same feature space as the input data.

## 1.2 Thesis Organization

First, an overview of the relevant theory for the practical part of the project is given in the Background chapter 2. This includes an overview of generative modeling using Diffusion models, as well as a brief

[1]https://openai.com/dall-e-2
[2]https://carla.org

description of the CARLA benchmark. Following this, in the Related Work chapter 3, the model that is currently ranked first on the CARLA leaderboard is introduced. Some of the relevant papers on Diffusion models in the context of image generation as well as sequential decision-making are also introduced. Next, in the Materials and Methods chapter 4, the data collection process using CARLA, as well as the proposed architecture is discussed. In the Results chapter 5, the improvements to the dataloading process as well as some visualizations produced by the proposed architecture are shown. The project report is concluded in the Conclusion chapter 6, where the achievements are summarized as well as some future improvements are discussed.

## 1.3 Division of Work

Despite only my name being listed amongst the authors, this project was originally intended to be a group project. Due to various circumstances, it was decided that I write a separate report. Much of the work that is talked about in this report has not been implemented by me personally, but rather the rest of the group. For the version that was adjusted by me, please refer to the GitHub repository[3] Their work and report as well as further information on the project is available on their GitHub repository[4].

[3]https://github.com/AyexGG/DiffuseDrive_dennisv
[4]https://github.com/graldij/DiffuseDrive

# Chapter 2

# Background

In this chapter, the building blocks of the developed autonomous driving system are introduced. As the pipeline consists of various different parts, only the most relevant components are introduced.

## 2.1 Diffusion Models

Diffusion models are generative models that have garnered increasing popularity in recent years due to their ability to generate high-resolution images with realistic details and excellent sample quality. They are inspired by non-equilibrium thermodynamics[19]. The most prominent one for generating imagery is the Denoising Diffusion Probabilistic Model (DDPM)[8] They work by iteratively transforming a simple distribution (such as a Gaussian distribution) into a target data distribution (e.g images) using a sequence of diffusion steps. Each diffusion step introduces noise into the current image, progressively transforming it into a more complex distribution that resembles the true data distribution. By applying multiple diffusion steps, the model learns to generate highly realistic and diverse images. Diffusion models are latent variable models that are formulated as Markov chains, where each image state only depends on the previous state.

#### Forward Diffusion

During forward diffusion, a given input image $\mathbf{x}_0$ is corrupted by gradually adding Gaussian noise through a series of $T$ noise steps. More formally, given $T$ timesteps and a variance schedule $\beta_1, \dots, \beta_T, \beta_t \in (0, 1)$, the approximate posterior $q(\mathbf{x}_{1:T}|\mathbf{x}_0)$, also known as *forward process* or *diffusion process* is given by:

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := p(\mathbf{x}_T) \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \qquad q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{2.1}$$

Using the reparameterization trick allows us to sample $\mathbf{x}_t$ at an arbitrary timestep $t$ in closed form. By setting $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$, we get:

$$q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1 - \bar{\alpha}_t)\mathbf{I}) \tag{2.2}$$

### 2.1.1 Reverse Diffusion

After a sufficiently large number of $T$ timesteps, the resulting image $\mathbf{x}_T$ should be nearly isotropic white noise. By reversing the forward process, i.e calculating $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$, it would be possible to generate images using pure Gaussian noise as input. As the reverse is intractable due to requiring knowledge of the data distribution, it would be prudent to approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$ using a neural network $p_\theta$. By applying the reverse process for all timesteps, we can go from $\mathbf{x}_T$ (pure Gaussian noise) to our target data distribution:

$$p_\theta(\mathbf{x}_{0:T}) = p_\theta(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) \tag{2.3}$$

Figure 2.1 shows an example of such a diffusion Markov chain.

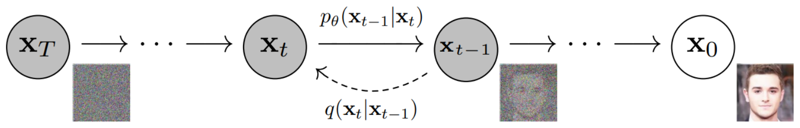
$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

Figure 2.1: Forward diffusion process $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ and reverse diffusion process $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ with $T$ timesteps. [8]

## 2.2 CARLA Benchmark

With autonomous driving rising in popularity over the last years, a safe method for testing the performance of autonomous driving pipelines is necessary. The CARLA (**CAR L**earning to **A**ct) benchmark[1][7] provides just that: a simulated environment that allows the deployment and benchmarking of autonomous driving systems. It is based on the Unreal Engine[2] and features 12 different townscapes, various weather conditions, traffic scenario simulation with autonomously moving vehicles and pedestrians, various autonomous driving sensors (such as LiDAR, Cameras, GPS, etc.) and much more. With its open-source codebase as well as its flexible API and extensive documentation, it is easy to develop and benchmark ADS. Next to the simulated environments, it also provides a leaderboard for its benchmark, where the ADS is tested on a predefined set of routes on townscapes only available for the benchmark. The benchmark scores the ADS using three different metrics:

- **Route completion**: Percentage of the route distance completed by an agent
- **Infraction penalty**: An aggregated metric that gets reduced the more infractions the ADS causes. These include collisions, running red lights/stop signs, minimum speed requirements etc.
- **Driving Score**: The product of the route completion and infraction penalty metrics, serves as the main metric of the leaderboard.

Autonomous agents need to follow a set of predefined routes, where they are given a starting point in the map, a destination and a number of intermediate coordinates. The routes will lead the agent through various areas such as freeways, urban scenes and residential districts.
During the route, the agent will face various traffic situations based on the NHTSA pre-crash typology[13]. This includes:

- Lane merging
- Lane changing

[1]https://carla.org
[2]https://www.unrealengine.com

- Negotiations at traffic intersections
- Negotiations at roundabouts
- Handling traffic lights and traffic signs
- Coping with pedestrians, cyclists and other elements

# Chapter 3

# Related Work

In this chapter, some of the relevant papers on autonomous driving and diffusion models are introduced.

## 3.1 InterFuser

The main starting point for this project was the InterFuser ADS[18]. It is currently leading the CARLA leaderboard in first place. Its approach is based on the fusion of multi-modal sensor data using vision transformers (ViT). Their approach uses the information from 3 forward-facing cameras (left, right, center) as well as LiDAR and a focused view (a cropped image of the center camera) using a transformer encoder to fuse the sensor data from different modalities into a singular feature space. Afterwards, the encoded values are decoded using three different queries to produce a set of future waypoints, an object density map (essentially a birds-eye view of the environment), as well as traffic rules (whether there is a stop sign, red light or intersection).
The output of these queries is precisely what makes the InterFuser pipeline interpretable. Yet at the same time, producing such training data requires costly annotations such as the bounding boxes for the object density map or information on signs and red lights at each timestep.
These intermediate features are then passed to a safety controller, which constrains the low-level actions (i.e steering and acceleration) into a safe set, after which a PID controller is used to generate the low-level actions.
Next to the costly annotations, another drawback of the Interfuser ADS is their reliance on hard-coded human heuristics while also not being fully end-to-end.

## 3.2 Denoising Diffusion Implicit Models

While DDPM already shows impressive results for generating images, one drawback is the large number of iterations that are needed during the reverse process to produce good images. The Denoising Diffusion Implicit Models (DDIM) paper[20] increases the speed of image generation by no longer relying on Markov chains for the reverse process. Instead, the reverse process is redefined via a class of non-Markovian diffusion processes that lead to the same training objective. This leads to a speed up of factor 10 to 50, while maintaining comparable image quality.

## 3.3 Diffusion in Reinforcement Learning

Next to image generation, diffusion models also show promise in decision-making problems, where they are used in combination with Reinforcement Learning (RL). Rather than denoising an image, the denoising process is applied to a number of state-action pairs. Two such applications are discussed in this section.

### 3.3.1 Diffuser

The Diffuser paper[11] aims to improve the decision-making process by enforcing a tighter coupling between the trajectory optimization pipeline and the model of the environment. They argue that models trained to approximate the dynamics of a system are often ill-suited for the planning algorithms that follow them, as they are often designed with ground-truth models in mind. Planning happens by sampling from the Diffuser model, which generates joint state-action pairs for a given time horizon. The distribution which is sampled from is perturbed using a function $h(\tau)$ that contains information about past observations as well as the goal that needs to be achieved. This serves as guidance for generating appropriate trajectories.
By setting the $h(\tau)$ function of a given trajectory $\tau$ as $h(\tau) = p(\mathcal{O}_{1:T}|\tau)$, where $\mathcal{O}_t$ is a binary random variable denoting the optimality of timestep *t* of a trajectory, one can exchange the RL problem into one of conditional sampling, where much previous work has already been done.
Under such a setting, the Diffuser decision-making model produces State of the Art (SOTA) results while also including some useful properties such as graceful handling of sparse rewards and the ability to plan for new rewards without retraining.

### 3.3.2 Decision Diffuser

The Decision Diffuser paper[2] also takes a look at sequential decision-making through the lens of conditional generative modeling, rather than RL. Unlike the Diffuser[11] approach that jointly diffuses state-action pairs, [2] only diffuses the state. They argue that action sequences are often discrete in nature, of high-frequency and less smooth, making them difficult to predict and model. The typically continuous nature of states better lends itself to generative modeling. After diffusing the state for the next timestep, the action is inferred using the inverse dynamics model[1]. Moreover, they also talk about conditioning of the trajectory data distribution for guided diffusion. Proper modelling of the conditioning variable, which may be e.g the return under a trajectory, can be used to maximize returns under a trajectory or satisfy a variety of constraints.

# Chapter 4

# Materials and Methods

In this chapter, the implementation of the network as well as the necessary preparation is talked about in more detail. Since the visualization of the CARLA environment is not possible on the provided Euler cluster, the environment was set up locally. As there were some issues with installation on Windows, an Ubuntu 20.04 dual-boot configuration was setup.

## 4.1 Data Collection using CARLA

As mentioned in section 2.2, the CARLA benchmark was used to test the model and collect the necessary training data.

### 4.1.1 Setup

Fortunately, the InterFuser[18] GitHub repository[1] already provides some scripts for the setup of the CARLA environment. The script automatically downloads the packaged 0.9.10 version of the CARLA benchmark, together with additional town maps. The `easy_install` feature of the `setuptools` package is used to install the CARLA `.egg` file. Afterwards, CARLA is used by starting up a CARLA server using a specified world port. Further programs can then be ran in the CARLA environment by using the same world port.

### 4.1.2 Data Collection

The InterFuser[18] repository also provides scripts for the generation of training data. The scripts first generate the necessary folder structure. Afterwards, the `.yaml` configuration files that specify the framerate, the weather, as well as how much the ground-truth waypoints of the route may be disturbed (a form of data augmentation essentially), are generated. Finally, the bash scripts that run all of the available scenarios for each town are generated.
For each town, there are a number of ”Tiny”, ”Short” and ”Long” routes available. Each scenario (i.e route length, town) is described by an `.xml` file, containing a set of routes described by a number of waypoints, where each waypoint describes the target x/y/z coordinates, as well as pitch/roll/yaw angles. Depending on the length of the route, a different number of waypoints is specified. While ”Long” routes contain more than 30 waypoints, ”Tiny” and ”Short” routes only contain two, the difference being the distance the waypoints are spaced apart.
Once the bash files are generated, they can be ran to collect data for a given scenario, provided a CARLA server is up and running. The Ego-Vehicle is controlled using an autopilot agent that is provided by the

[1]https://github.com/opendilab/InterFuser

| | Route Type | | |
|---|---|---|---|
| | **”Tiny”** | **”Short”** | **”Long”** |
| **Town01** | 0/292 | 21/23 | 5/10 |
| **Town02** | 0/129 | 12/15 | 3/6 |
| **Town03** | 136/442 | 4/22 | 3/30 |
| **Town04** | 33/358 | 7/24 | 2/39 |
| **Town05** | 338/338 | 32/32 | 4/26 |
| **Town06** | 4/294 | 0/18 | 0/50 |
| **Town07** | 0/142 | 0/14 | - |
| **Town10** | 117/117 | 0/13 | - |

Table 4.1: Table of collected routes out of the total number of routes for each town using `weather-0`

InterFuser repository. For each frame, the autopilot also generates a high-level CARLA command using a local planning module. This high-level command describes what the vehicle should do, i.e turn left, right, follow lane, change lane.
Since the collection of a single scenario takes up to 48 hours, only a single weather type (`weather-0`) was used for the data collection process. Moreover, as the autopilot occasionally runs into difficulties during the track that causes the autopilot to stop moving (e.g when turning while a vehicle is incoming from the opposite direction), mostly ”Tiny” and ”Short” routes are collected. While the autopilot eventually runs into a timeout and aborts the track, this often causes long periods of inactivity, proving problematic for the training. A total of around 60GB of data was collected for training. An overview of the routes that have been collected can be seen in table 4.1.

### 4.1.3 Data Preprocessing

The collected routes contain the following types of data for each frame:

- RGB images from left/right/center/rear cameras and topdown view
- Depth images from left/right/center cameras
- Segmentation images from left/right/center cameras and topdown view
- LiDAR measurements
- 2D/3D bounding boxes for surrounding objectives
- Measurement data (GPS, accelerometer, CARLA high-level commands, weather information, traffic information)

Since not all of the collected data is relevant for the training, some preprocessing was necessary to obtain only the required data. First of all, due to the aforementioned issues regarding timeouts of the autopilot, the frames where the vehicle is still need to be filtered. An exception is made for cases where the agent is waiting at a red light. Fortunately the InterFuser[18] repository also provides scripts for this inside the `tools/data` folder. First, a `.txt` file of all the routes that need to be filtered is generated using the `batch_stat_blocked_data.py`. After that, the blocked data is removed by passing the generated `.txt` file to the `batch_rm_haze_data.py` script. Since this causes gaps in the collected data process (e.g 100 frames of data, files are named `0.datatype-99.datatype`), they need to be renamed to make it easier to work with them again. Although the name would suggest otherwise, the provided `batch_recollect_blocked_data.py` script does just that.

Now that the timeout frames have been filtered out, we copy the data that we need (RGB center images, topdown view, measurements) into a separate folder structure for further usage. The copied images are also resized and cropped from their original 800x600 pixels to a resolution of 128x128px.
While the vehicle coordinates that are provided in the `measurements.json` are in world coordinates (x,y,$\theta$), what is relevant for the training procedure is their relative offset compared to the previous frame. The conversion from the world coordinates to the ego-vehicle coordinates is done via a rotational matrix.
To speed up the data loading process, this calculation is done beforehand. For frame at timestep $t$, they are deposited into a `ego_data/t.json` file. The ego-data `.json` contains:

- The ego-frame coordinates for the past and future 10 waypoints (or less if there are not enough past/future frames)
- The current high-level CARLA command
- The past and future 10 high-level CARLA commands

After the data preprocessing step, we are left with around 5GB of data, containing 80'000 usable frames.

## 4.2 Model Architecture

This section describes the overarching architecture of the developed system.

### 4.2.1 Initial Idea

The DiffuseDrive pipeline aims to use diffusion models to predict the future frames of a given RGB camera image by taking its past trajectory and the current high-level CARLA command into account. First, the last front-view image is encoded using the encoder of a variational autoencoder (VAE). Together with the last waypoint, they are added to a state buffer that stores the last $S$ timesteps. The waypoints and encodings of the past $S$ timesteps are then concatenated with the next $T$ future timesteps, which only contain noise. This concatenated tensor is subsequently fed into the diffusion model, together with the last $S$ timesteps as conditioning, to produce predictions of the future $T$ waypoints and frames.
Ideally, this output may then be decoded using the decoder of the VAE to reconstruct the images for human visualization and interpretation.
The diffusion model essentially produces a future trajectory for the agent. By sampling multiple times, multiple different trajectories may be produced. By using the visualization from the decoder, a combination of event prediction and scoring networks should then be used to score the different trajectories. The event prediction network consists of a feature extraction process (e.g using Convolutional Neural Networks (CNN) or Vision Transformers[6]), after which an MLP should classify infractions (such as collision with objects or traffic rule violations). Finally the event scoring network assigns each sampled trajectory a score, from which the best one is chosen. The future waypoints of the best trajectory are then targeted using a simple PID controller.
Figure 4.1 depicts the original idea of the DiffuseDrive pipeline.

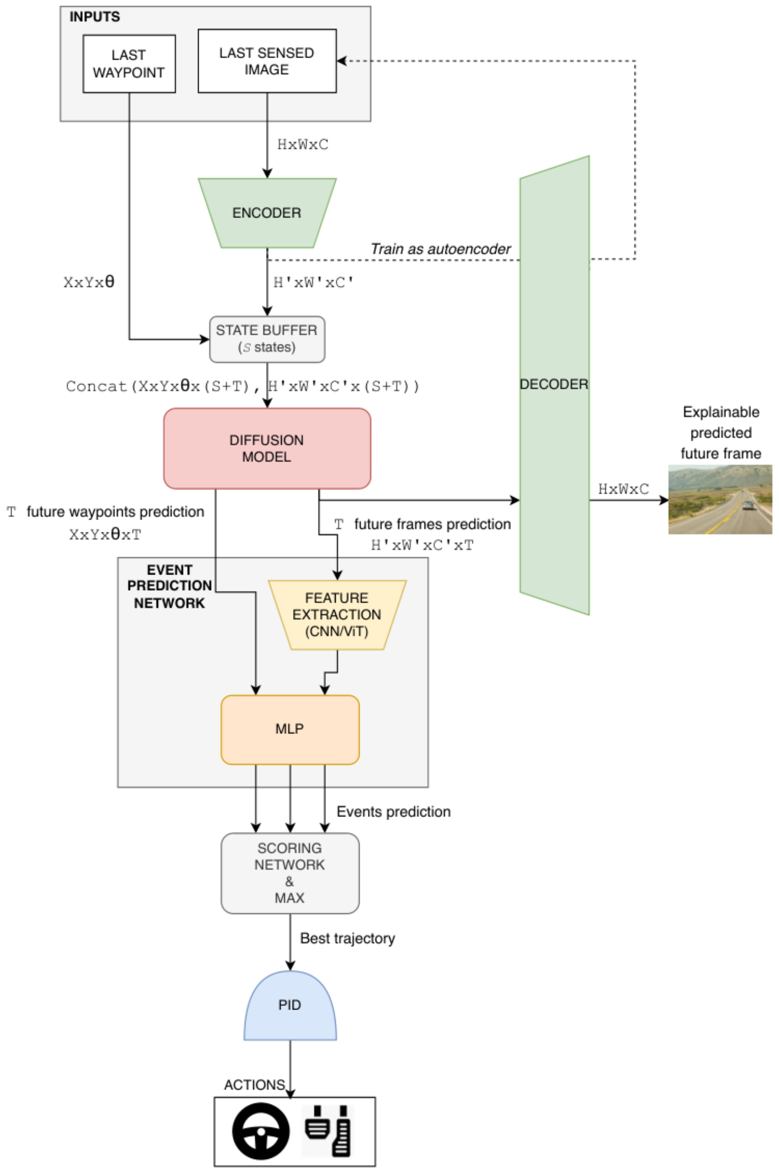

Figure 4.1: DiffuseDrive pipeline

### 4.2.2 Adjustments to the Architecture

Unfortunately, the diffusion of both image and waypoint trajectories proved to be too computationally expensive. As each trajectory consists of $T$ future images and multiple trajectories need to be sampled, it was necessary to reduce the dimensionality of the input/output to ensure that a high-enough framerate may be achieved. Instead of diffusing both the image as well as the waypoints, only the waypoint trajectory is diffused. This idea was heavily inspired by the Decision Diffuser paper[2]. The diffusion model is trained to denoise the part of the input tensor containing the future waypoints, while past and present waypoints act as conditioning. To ensure temporal consisteny, a Temporal U-net model[11] was used to fit the data as well as image conditioning. The Temporal U-net consists of 6 repeated residual blocks, each block consisting of two temporal convolutions, followed by a group norm[21] and Mish nonlinearity[12]. For further detail, please refer to the Diffuser paper[11] as well as their GitHub repository[2].
As additional conditioning, a segmented image of the current front view, as well as the current high-level CARLA command is also used. For the segmented image, the pretrained DeeplabV3Plus[3] with MobileNetV2 backbone was used. The model was pretrained on semantic segmentations from the Cityscapes[5] dataset. The current high-level CARLA command is one-hot encoded and fed through an MLP to yield an encoded representation.

[2]https://github.com/jannerm/diffuser

# Chapter 5

# Results

## 5.1 Improvement on training speed

As previously mentioned, the dataloading process was adjusted in order to increase the speed of the dataloader. Instead of generating ego-frame coordinates on the fly for the past/future waypoints, they are generated once during the dataset creation, after which they only need to be loaded during training time. To test the improvements in dataloading speed, 100 batches of 16 trajectories each were generated using both the new dataloading process as well as the original implementation. The results of this experiment are shown in Figure 5.1. It can be seen that the loading process takes around 8 seconds, compared the old implementation which takes around 27 seconds on average. Since the dataloading process proved to be a major bottleneck during training time, this adjustment lead to significant increases in training speed.

## 5.2 Future trajectory generation

For the evaluation of future trajectories, the model was trained for 90'000 steps. The initial learning rate of 0.0001 is decayed by a factor of 0.8 every 10'000 steps. A batch size of 16 was used, with each trajectory containing 4 past waypoints as well as 6 future waypoints. Training the model took 63 hours.
With this setup, the model is capable of simple tasks such as following lanes, lane change and turning. Some examples are shown in figure 5.2 Although during turning, the model generally predicts trajectories that are more conservative than the ground truth.
It is also worth mentioning that despite sampling multiple trajectories, the sampled trajectories heavily overlap after some training time. Only at the start of the training period do they differ in an way. Moreover, the trained model occasionally fails to turn correctly. Both these behaviors can be seen in figure 5.3

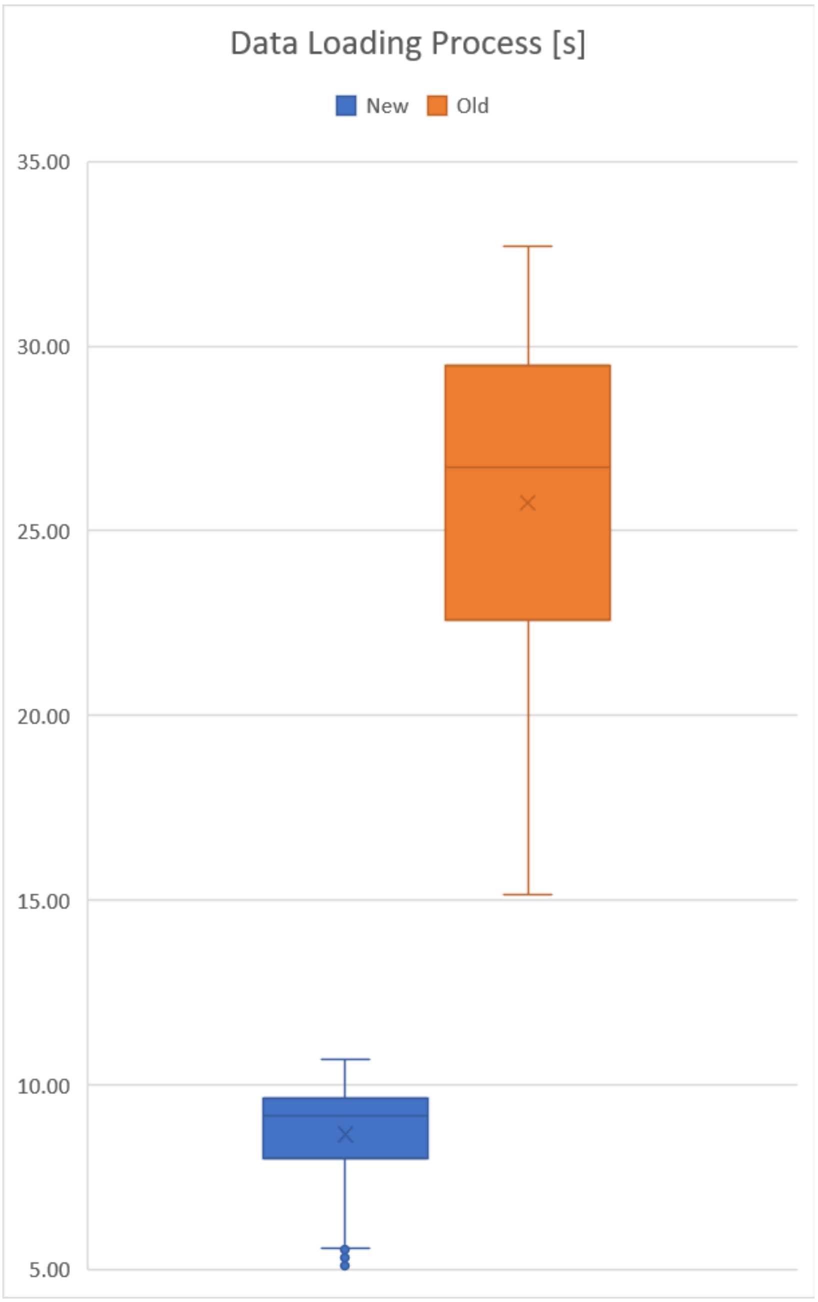

Figure 5.1: Boxplot showing the difference in data loading speed

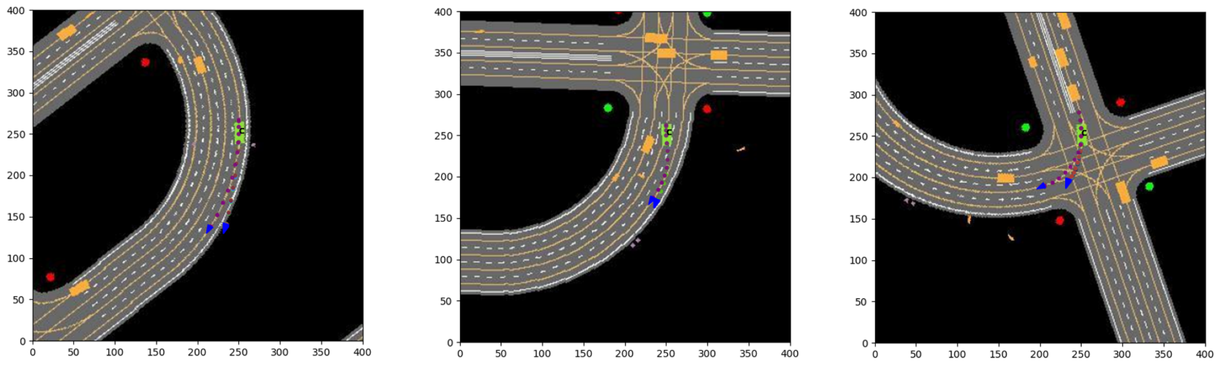

Figure 5.2: Trained model following the lane (left), changing lane (middle) and turning (right). Ground truth is shown in purple.

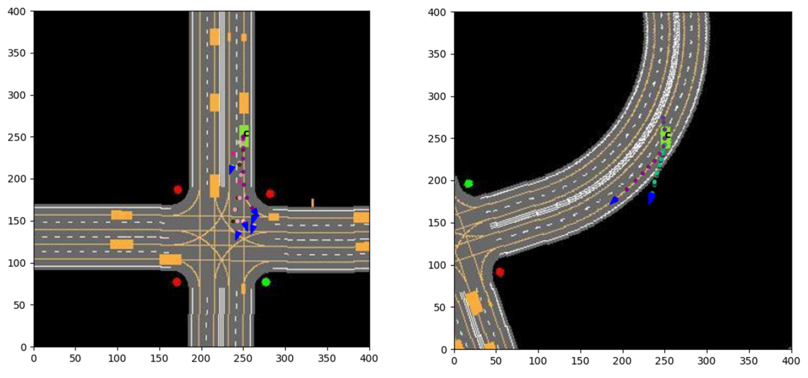

Figure 5.3: Untrained model with various sampled trajectories (left), Trained Model failing to follow lane (right). Ground truth is shown in purple.

# Chapter 6

# Conclusion

While the achieved results of this project are far from able to land it a spot in the CARLA leaderboard, it can be seen that Diffusion models show promise in their application for autonomous driving scenarios.
After setting up the CARLA simulation benchmark and downloading additional town maps, sunny weather data was collected for all of the towns available in the CARLA environment. This was done using the data collection scripts from the InterFuser repository, using their provided implementation of an autopilot agent. While far from perfect, requiring some additional preprocessing to eliminate undesired behavior such as standstills during timeout, around 80'000 frames of usable data were collected. To ensure compatibility with the used DeeplabV3Plus segmentation backbone and the rest of the pipeline, the images were also resized from 800x600px to 128x128px. A focus was placed on collecting extensive data for a single town, for subsequent evaluation on the same town. To speed up the data loading process, the provided GPS coordinates of each frame were also converted to "ego-frame" coordinates, depicicing their position in relation to previous and future waypoints of a trajectory. This preprocessing step was able to speed up the dataloading process by more than a factor of 3.
Training on this data, the model was able to achieve adequate results in ideal scenarios. The model shows signs of being able to follow the road, change lane and turn.

## 6.1 Future Work

Much work still needs to be done in order to allow the model to function in more complex traffic scenarios. Since so much of the collected data is removed during the filtering process, very little of the collected 60GB is usable. Therefore, it would be beneficial to collect even more training data. At some point this data should also contain additional weather types. As the autopilot also follows the road exactly and is always on track, the DiffuseDrive system struggles to cope with suboptimal positioning on the road. It would help to have more data containing scenarios where the agent needs to recover from such suboptimal positioning.
While the model is able to behave adequately in simple scenarios, it struggles when there is additonal traffic information that needs to be regarded, such as red lights, stop signs or other agents on the road. With the current architecture, where the only visual information used by the system is a segmented image of the front-camera, much of the needed information regarding the traffic is lost. Red lights and stop signs are simply "segmented out". A traffic event-classification network with additional conditioning of the diffusion model on such traffic information could lead to improved behavior in such situations. Next to this, it is difficult to balance the importance of the current front-view image and high-level CARLA command conditioninings. When using the image as additional conditioning, the CARLA high-level command is often ignored. Some additional tuning in this regard is necessary to achieve good planning results.
As the model only diffuses waypoint trajectories, the original goal of interpretability has taken a backseat.

With additional computational resources and some optimizations of the network architecture, it would be interesting to see the performance when future images are also diffused, lending itself to better interpretability.

# Bibliography

[1] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics, 2017.

[2] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making?, 2023.

[3] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation, 2018.

[4] Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. Transfuser: Imitation with transformer-based sensor fusion for autonomous driving. *Pattern Analysis and Machine Intelligence (PAMI)*, 2022.

[5] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

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

[8] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[9] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022.

[10] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[11] Michael Janner, Yilun Du, Joshua Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. In *International Conference on Machine Learning*, 2022.

[12] Diganta Misra. Mish: A self regularized non-monotonic activation function, 2020.

[13] NHTSA. U.S. Department of Transportation releases policy on automated vehicle development. `https://www.transportation.gov/briefing-room/us-department-transportation-releases-policy-automated-vehicle-development`, May 30 2013.

[14] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021.

[15] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models, 2022.

[16] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[17] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.

[18] Hao Shao, Letian Wang, RuoBing Chen, Hongsheng Li, and Yu Liu. Safety-enhanced autonomous driving using interpretable sensor fusion transformer. *arXiv preprint arXiv:2207.14024*, 2022.

[19] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015.

[20] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models, 2022.

[21] Yuxin Wu and Kaiming He. Group normalization, 2018.