POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Sim-to-Real Autonomous Driving Testing via Scene Reconstruction and Neural Rendering

Supervisor
prof. Maurizio Morisio
prof. Andrea Stocco

Candidate
Davide Jannussi

Academic Year 2025-2026

....

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Autonomous driving systems (ADS) represent the combination of hardware and software that enables a vehicle to drive without human intervention [1]. Nowadays, they are becoming increasingly popular and are attracting growing interest from both research and industry.

Since they operate in safety-critical scenarios, their misbehavior could put the safety of the people involved at risk [32]. For this reason, the validation of ADS represents a crucial step before their deployment in the real world. Extensive testing is necessary to ensure the safety and reliability of the system and to identify in advance possible failures or anomalous behaviors before deployment.

Real-world testing represents the most reliable and trustworthy form of evaluation, since the ADS behavior is validated in its deployment domain [26]. However, it comes with several drawbacks:

- **safety**: since it is important to test ADS in safety-critical conditions, validating an ADS in the real world could put people's safety in danger;
- **cost**: employees working on testing must be paid, crashes have an economic cost, and real-world experiments require significant resources in terms of time and money;
- **lack of reproducibility**: real-world conditions are not stationary. Weather, lighting conditions, and traffic may change from one run to another. For this reason, it is impossible to replicate exactly the same test twice in a real-world scenario. This makes it difficult to isolate whether a change in the ADS behavior is caused by the system itself or by changes in the surrounding environment.

To address these drawbacks, several other testing techniques have been employed. In particular, 2D driving datasets have been used to provide the driving model with realistic inputs obtained from real driving recordings [35]. However, this testing modality can only be used for model-level testing: only the driving model is evaluated, and the predicted driving commands do not control a real vehicle. Therefore, they cannot influence the generation of subsequent frames.

The use of simulators such as CARLA, SVL Simulator, and AirSim [48, 49, 51], in the context of ADS testing arose to overcome these limitations. Simulation-based testing allows reproducible evaluation of the entire system rather than only the driving model, making it possible to identify failures caused by cascades of errors that would not emerge in model-level testing alone. For example, in model-level testing, an incorrect driving command does not influence the subsequent frames. In system-level testing, instead, an incorrect output can change the future state of the vehicle and eventually lead the ADS to bad positions from which it cannot recover.

However, in simulation, testing scenarios are only a simplified approximation of real-world environments, as they rely on simplified physics and a game-engine visual appearance. Since ADS are sensitive to changes in the input domain [53], these differences may cause an ADS tested in simulation to exhibit behaviors that do not transfer to the real world [25]. This problem is known as the sim-to-real gap [45]. Different types of reality gap can be identified, including the perception gap, which arises from differences between synthetic and real sensor observations [46], the actuation gap, which is related to mismatches between simulated and real vehicle dynamics and control execution [55], and the behavior gap, which depends on differences in the behavior of other traffic participants in simulation compared with the real world [55]. In this work, we focus on the perception gap.

The use of novel view synthesis and generative AI techniques has been proposed to mitigate this problem [80, 65], with the aim of providing the driving model operating in simulation with generated inputs that better resemble the real world than raw synthetic data [80]. Among recent works, NVIDIA Cosmos [78] proposed a platform that allows the use of these techniques for both model-level and system-level testing.

In this work, we propose a framework that enables the use of driving data within 3D simulators for system-level testing with different neural rendering techniques, in order to compare their effectiveness in terms of the driving behavior of a self-driving model rather than relying only on image quality metrics, which have been shown to be insufficient [94].

We trained an end-to-end camera-based driving model based on the NVIDIA PilotNet architecture [21] using driving recordings that we collected. We then deployed the model on a full-size vehicle to record the driving data used in our evaluation. Starting from these recorded driving data, we used the framework to reconstruct the driving scenario inside the CARLA simulator and to train different neural rendering techniques to generate visual inputs for the driving model when tested in simulation.

We evaluate the tested techniques, in particular Stable Diffusion (SD) [62], Neural Radiance Fields (NeRF) [59], and 3D Gaussian Splatting (3DGS) [60], by comparing the behavior of the driving model in simulation with these techniques against its behavior in the real-world baseline recordings.

To guide this evaluation, we formulate three research questions.

RQ1) Scenario Reconstruction Fidelity. How accurate is the parked vehicle detection component of our framework at estimating parked vehicle coordinates in real-world driving scenarios? Does the framework successfully reconstruct real-world recorded driving scenarios in simulation? This determines whether the proposed framework faithfully recreates in simulation the recorded scenarios used as the basis of the evaluation.

RQ2) Generated Image Quality. Do Stable Diffusion, NeRF, and 3DGS produce images that resemble real-world ones? Which of these techniques generates the most similar images compared to their real-world counterparts? This measures the visual fidelity of the generated images through reconstruction metrics.

RQ3) System-Level Behavior. How does the driving model behave in system-level testing with synthetic images and different image generation techniques, compared to real-world driving in the same scenario? This shows which technique better aligns the driving behavior with the real-world baseline and whether the behavioral results are consistent with the image quality metrics.

Chapter 2

Background and Related Work

This chapter introduces the key concepts required for this work. It provides background on autonomous driving systems and the challenges involved in testing them, discusses simulation-based testing and the sim-to-real gap, and presents neural rendering techniques proposed to address this gap. In addition, the chapter reviews relevant related work, including approaches that are comparable to the methods explored in this thesis.

2.1 Autonomous Driving Systems

An autonomous driving system refers to the combination of software and hardware that enables a vehicle to drive without human intervention. ADS receive input data from onboard sensors and produce the commands required to navigate a road environment (e.g., steering, throttle, and braking) [1]. To perceive the environment, ADS generally employ cameras or LiDAR as their primary sensors[2, 3], either in isolation or in combination. The driving models are typically divided into two main architectural types: modular and end-to-end [5].

2.1.1 Modular Pipeline

Modular driving models decompose the autonomous driving task into multiple modules, each responsible for a specific function [6]. A typical modular pipeline consists of the following components:

- **Perception module**: processes data from onboard sensors such as cameras, LiDAR, or radar to detect and classify relevant elements of the environment. Typical perception tasks include object detection, lane detection, and traffic sign recognition. Common techniques include convolutional neural networks or Computer vision techniques for object detection [7, 8].
- **Localization module**: estimates the vehicle's position and orientation within a map of the environment. This module typically fuses information from multiple sensors such as GPS, inertial measurement units (IMU), LiDAR, or cameras. Techniques such as visual odometry, LiDAR-based localization, or simultaneous localization and mapping (SLAM) are commonly used [9].
- **Planning module**: determines how the vehicle should navigate the environment. Planning is often divided into two stages. A *global planner* computes a high-level route on a road map from the current location to the destination, while a *local planner* generates a feasible and safe trajectory that the vehicle can follow while avoiding obstacles and respecting dynamic constraints [10].
- **Control module**: converts the planned trajectory into low-level driving commands such as steering, throttle, and braking. Control algorithms such as PID controllers, model predictive control (MPC), or pure pursuit controllers are commonly used to ensure that the vehicle accurately follows the planned trajectory [11].

This architecture offers several advantages. Because each component performs a well-defined task, individual modules can be analyzed, validated, and improved independently [12]. However, modular pipelines also introduce limitations. Errors in upstream modules may propagate through the system and affect downstream decisions [14]. Communication between modules may introduce latency [15].

Popular examples of Modular ADS include Baidu's Apollo[16] and Autoware[17] which have been widely adopted and evaluated in both industry and Accademia [18, 19, 20].

2.1.2 End-to-End

A popular architectural choice in autonomous driving systems is the use of end-to-end deep neural network models [21]. These approaches aim to learn a

direct mapping from sensor observations to driving actions using data-driven methods.

In end-to-end architectures, the driving model directly predicts control commands such as steering, throttle, and braking from raw sensor inputs, typically camera images[22]. Because the system is trained to optimize the driving task as a whole, it avoids the need for explicitly designed intermediate modules such as perception, localization, or planning. This allows the model to jointly learn representations and decision policies that may capture complex relationships between perception and control [24]. End-to-end approaches have demonstrated promising results in several autonomous driving settings, including imitation learning systems and reinforcement learning-based driving agents [23].

However, the absence of explicit intermediate components also introduces important challenges. End-to-end models rely heavily on the statistical properties of the training data and can therefore be sensitive to distribution shifts in the input domain, such as changes in lighting, weather, or sensor characteristics [22]. In addition, because the internal decision process is not decomposed into interpretable modules, it is often more difficult to analyze and diagnose the causes of system failures compared to modular pipelines [24].

In this work, we focus on testing end-to-end driving models. These systems are widely studied in recent research and provide a relatively lightweight architecture compared to complex modular pipelines. Moreover, because perception constitutes the primary input to the model and directly influences the predicted control commands, perception-related errors can propagate through the entire system. This makes end-to-end models particularly sensitive to visual domain shifts, and therefore especially relevant for studying the sim-to-real gap which arises when testing real-world systems in simulation[25].

2.2 Testing Autonomous Driving Systems

Before deploying an autonomous driving system (ADS) on public roads, its behavior must be thoroughly evaluated to ensure safety and reliability [26]. Failures in autonomous driving systems can lead to serious real-world consequences, including traffic accidents, injuries, and loss of life.

Several recent incidents highlight the importance of rigorous testing. In September 2024, a crash involving a Tesla Model S equipped with Autopilot

and Full Self-Driving features resulted in the loss of three lives [32]. Legal scrutiny regarding such failures is also increasing. In 2025, a jury found Tesla partially liable for a 2019 accident in which self-driving software was active when a pedestrian was killed and another injured, citing the ADS's failure to alert the driver [33]. More recently, in January 2026, the National Highway Traffic Safety Administration opened an investigation after a Waymo ADS vehicle struck a child, causing minor injuries [34]. These events illustrate the potential risks associated with autonomous driving technologies and underline the need for systematic testing before deployment.

Several testing approaches have been proposed in both research and industry. In this section, we describe two main evaluation paradigms: model-level testing and system-level testing.

2.2.1 Model-Level Testing

Model-level testing evaluates the driving model in isolation using prerecorded datasets [35]. The model receives sensor observations as input and produces predicted control commands such as steering, throttle, and braking. These predictions are compared with reference commands representing the correct driving behavior for each sample using metrics such as mean squared error[36]. Public driving datasets containing sensor data and ground-truth driving commands are commonly used for this purpose [37].

This testing modality has several advantages. It is computationally efficient, simple to implement, and highly reproducible because the same dataset can be reused across experiments [35]. For these reasons, model-level testing is widely used for benchmarking and comparing different driving models. Popular datasets used for this purpose include Cityscapes [38], KITTI [39], the Waymo Open Dataset [40], and Comma2k19 [41]. These datasets provide recorded sensor observations together with annotations such as object bounding boxes, semantic segmentation labels, or LiDAR point clouds, and in some cases driving commands or vehicle trajectories collected during real-world driving.

However, model-level testing also has important limitations. Because the model processes prerecorded inputs, its predictions do not influence future observations. In real driving, incorrect actions affect the vehicle state and therefore change the observations received at the next time step. These feedback effects cannot be captured by dataset-based evaluation. As a result, strong performance at the model level is necessary but not sufficient to guarantee the correct behavior of an ADS [35].

2.2.2 System-Level Testing

System-level testing evaluates the ADS while it actively controls a vehicle and interacts with its environment [12]. In this setting, the control commands predicted by the model directly influence the motion of the vehicle, creating a closed feedback loop between the system and the environment.

This testing modality allows the evaluation of the overall behavior of the ADS over time and can reveal failures that are not observable in model-level testing [35]. For example, a small steering error may gradually move the vehicle away from its intended trajectory, producing new observations that the model may not have encountered during training [23]. At the same time, errors at one time step may sometimes be mitigated by subsequent corrective actions, making the closed-loop behavior of the system essential to evaluate [42].

System-level testing can be performed either with real vehicles in the physical world or with simulated vehicles in virtual environments.

Real-World Testing

Real-world testing evaluates the ADS on a physical vehicle operating in real environments [26]. This approach provides the most realistic validation because the system interacts with real sensor measurements, road conditions, traffic participants, and environmental variability.

Several works have nevertheless employed real-world testing to evaluate autonomous driving systems. Notable examples include studies by Guo et al. [27], Kessler et al. [28], and Testouri et al. [29], where the authors deploy open-source autonomous driving stacks on physical vehicles and evaluate their performance in real driving conditions, and Chen et al. [30], where the authors test an end-to-end driving model on real roads and analyze its behavior across highway and urban scenarios. Other works, such as Feng et al. [31], also conduct real-world experiments to validate autonomous vehicle testing methodologies using physical vehicles on dedicated test tracks. However, real-world testing also presents significant challenges. Deploying experimental driving systems on real vehicles is expensive, time-consuming, and potentially unsafe. In addition, it is difficult to reproduce specific scenarios or evaluate rare events such as dangerous traffic situations. These limitations make large-scale testing in the real world impractical during early stages of development.

In this work, we perform real-world testing of an ADS trained on real-world data on public roads. These experiments provide camera and LiDAR

recordings together with reference driving behavior. The collected sensor data are used to train the evaluated techniques, while the recorded driving behavior serves as a reference for evaluating how closely the simulated systems reproduce real-world driving.

Simulation-Based Testing

Simulation-based testing evaluates the ADS within a virtual environment where roads, traffic participants, and environmental conditions are digitally recreated [55]. The driving system controls a simulated vehicle that receives synthetic sensor data such as camera images or LiDAR point clouds generated by the simulator.

Simulation enables large-scale and safe experimentation. Dangerous scenarios can be reproduced without risk, and environmental conditions such as weather, traffic density, or lighting can be systematically varied. These characteristics make simulation an essential tool for evaluating autonomous driving systems before real-world deployment.

Several simulators have been developed for autonomous driving research, including CARLA [48], SVL Simulator [49], and AirSim [51]. Despite their advantages, simulators cannot perfectly reproduce real-world environments. Differences in visual appearance, sensor noise, and environmental dynamics may cause models that perform well in simulation to behave differently in real-world conditions. This discrepancy is commonly referred to as the *sim-to-real gap* [45].

CARLA Simulator CARLA is an open-source simulator designed specifically for autonomous driving research. It is built on top of the Unreal Engine game engine, which provides photorealistic rendering and realistic physics simulation.

CARLA offers a Python application programming interface (API) that allows users to control the simulation programmatically. Through this interface, it is possible to spawn vehicles and pedestrians, configure traffic scenarios, and retrieve sensor data generated by the simulated environment. The simulator supports several sensor types commonly used in autonomous driving systems, including RGB cameras, LiDAR, radar, GNSS (Global Navigation Satellite System), and IMU (Inertial Measurement Unit) sensors.

CARLA has become a widely used platform for autonomous driving research and benchmarking. It has been employed in numerous studies for tasks such as end-to-end imitation learning [23, 52], simulation-based ADS

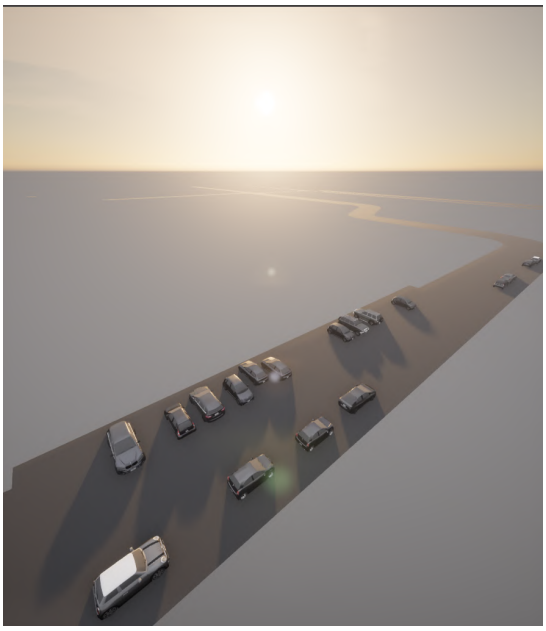
testing [42], and sim-to-real transferability analysis [25].

The simulator provides several predefined urban environments containing detailed 3D elements such as buildings, vegetation, and traffic infrastructure. CARLA also supports custom maps defined using the OpenDRIVE (XODR) road network format [61]. OpenDRIVE describes the geometric structure of roads and lanes but does not include detailed 3D scene elements.

As a result, environments generated from custom XODR maps typically contain only the road geometry and vehicles, while surrounding scene elements such as buildings and vegetation are absent. This leads to visually simpler environments compared to the predefined CARLA maps and increases the visual discrepancy between simulated and real-world scenes, as illustrated in Figure 2.1.

In this work, we use CARLA as the baseline simulation environment on which we apply the proposed techniques to reduce the sim-to-real gap between simulated and real-world driving scenes.

(a) Predefined CARLA map



(b) Custom XODR map

Figure 2.1: Comparison between a CARLA predefined map and a custom XODR map.

2.3 Sim-to-Real Gap Mitigation

The sim-to-real gap arises due to simulators inevitably simplifying many aspects of the real world. Real environments involve complex physical interactions, unpredictable traffic behavior, diverse weather conditions, and sensor noise that are difficult to model accurately in simulation [45]. As a result, the observations perceived by a driving system in simulation may differ from those encountered during real-world deployment, potentially leading to unexpected behavior.

One important component of the sim-to-real gap is the *perception gap*, which refers to the mismatch between real sensor data and simulated sensor data [46]. For example, images produced by game engines often differ from real camera recordings in terms of texture, lighting, and overall visual appearance. These discrepancies are particularly problematic for learning-based perception systems. Deep neural networks used in autonomous driving rely heavily on statistical patterns present in their training data. When the distribution of input data changes, a phenomenon commonly known as *domain shift*, model performance may degrade [53]. As a result, models trained or evaluated in simulation may fail to generalize correctly to real-world environments.

This issue is especially relevant for camera-based autonomous driving systems, where perception relies on visual features extracted from images. Differences in illumination, texture realism, and scene composition between simulated and real images can cause the perception system to produce incorrect detections or scene interpretations [54]. These perception errors can then propagate through the driving pipeline and lead to incorrect control decisions. For these reasons, evaluating autonomous driving systems purely in simulation can lead to misleading conclusions about real-world performance. Simulation-based evaluation may either underestimate or overestimate the capabilities of the system under real operating conditions [55].

In this work, we investigate techniques for mitigating the sim-to-real gap in simulation-based testing. In particular, we apply neural rendering techniques to reduce the discrepancy between simulated and real sensor observations and improve the reliability of simulation-based evaluation.

2.3.1 Neural Rendering Techniques

Several techniques have been proposed to reduce the perception gap between simulated and real sensor observations. Among the most widely adopted approaches are generative models and neural rendering methods, which aim to produce more realistic visual inputs than those generated directly by simulators [65].

Generative models are typically used to translate simulated images into more realistic ones while preserving their semantic structure [64, 58]. Neural rendering techniques instead reconstruct scenes from real-world observations and allow them to be rendered from new viewpoints [56, 57]. Both approaches aim to reduce the discrepancy between simulated and real data that affects the behavior of perception-driven autonomous driving systems.

In this section, we introduce the neural rendering techniques considered in this work. In particular, we describe Stable Diffusion[62] with ControlNet[71] as a generative image synthesis approach, and NeRF[59] and 3D Gaussian Splatting[60] as neural rendering techniques for novel view synthesis.

Stable Diffusion and ControlNet

Diffusion models [63] are a class of generative models that learn to synthesize images by reversing a progressive noise process. During training, Gaussian noise is gradually added to images until they become random noise. A neural network is then trained to reverse this process by iteratively removing the noise and reconstructing the original image distribution.

Stable Diffusion is a latent diffusion model that performs the diffusion process in a compressed latent space rather than directly in pixel space. Images are first encoded into a lower-dimensional representation using a Variational Autoencoder (VAE) [68]. A U-Net architecture [69] then performs the denoising process in this latent space, and the resulting representation is decoded back into an image by the VAE decoder. Operating in the latent space significantly reduces computational cost while maintaining image quality.

The generation process can be guided through textual prompts encoded by a CLIP text encoder [70]. These embeddings are injected into the U-Net through cross-attention layers, enabling the model to control the semantic content of the generated image. However, text conditioning alone provides limited control over the spatial structure of the generated scene.

ControlNet [71] extends Stable Diffusion by introducing spatial conditioning. It augments the original diffusion architecture with an additional network that processes structural inputs such as semantic segmentation maps,

edge maps, or depth maps. These inputs guide the generation process so that the resulting image follows the spatial layout of the conditioning signal while maintaining the realistic appearance produced by the pretrained diffusion model.

NeRF

Neural Radiance Fields (NeRF) [59] are a neural rendering method for synthesizing novel views of a scene. Given a set of images captured from known camera positions, NeRF learns a continuous representation of the scene that allows rendering images from new viewpoints.

The scene is represented by a neural network, specifically a multilayer perceptron (MLP)[72], that maps a 3D spatial position and viewing direction to two quantities: a volume density and a color value. The density indicates whether a point in space is occupied, while the color represents the appearance of the scene from a particular viewing direction.

To render an image, rays are cast from the camera through each pixel. Points are sampled along each ray and evaluated by the network to obtain their density and color values. These values are then combined using volume rendering techniques to compute the final color of each pixel.

NeRF models are trained using images of the scene together with their corresponding camera poses, which can be estimated using structure-from-motion tools such as COLMAP [74]. Because the rendering process is differentiable, the network can be optimized end-to-end by minimizing the difference between rendered images and real images.

A limitation of NeRF is that each trained model represents only a single scene[59]. In addition, rendering can be computationally expensive because the neural network must be evaluated at many points along each ray for every pixel[59].

3D Gaussian Splatting

3D Gaussian Splatting [60] is a recent neural rendering technique that represents a scene using an explicit set of 3D Gaussian primitives rather than a neural implicit representation.

In this approach, the scene is modeled as a collection of Gaussian elements positioned in 3D space. Each Gaussian has learnable parameters including its position, shape, opacity, and view-dependent color represented using spherical harmonics[60]. The initial set of Gaussians is typically obtained from

the sparse point cloud produced by structure-from-motion algorithms such as COLMAP[74].

During rendering, the Gaussians are projected onto the image plane and blended using rasterization. The rendered image is compared with the training images, and the Gaussian parameters are optimized to minimize the reconstruction error.

Compared to NeRF, Gaussian Splatting provides significantly faster rendering because it avoids repeated neural network evaluations along rays. Rendering can therefore achieve real-time performance while maintaining high visual fidelity.

2.3.2 Neural Rendering for Sim-to-Real Gap Mitigation

Several works have investigated the use of generative models and neural rendering techniques to reduce the visual discrepancy between simulated and real sensor observations [75, 77, 79, 80, 81]. These approaches aim to transform simulated data so that it more closely resembles real-world observations, allowing perception systems to operate under conditions that better match those encountered during deployment.

Existing approaches typically apply these techniques either offline on prerecorded datasets or online within simulated environments. Many works rely on generative models such as GAN-based image translation methods or other generative approaches to modify simulated images, sometimes operating on individual frames or short video sequences [75, 76]. Notable examples include studies by NVIDIA et al. [78], where generative models are used to translate simulated images into a more realistic visual domain. In most cases, these techniques are evaluated on static datasets or within simulation environments rather than in real-world deployments.

Evaluation in prior work also commonly relies on image similarity metrics such as SSIM or FID [96, 98]. However, recent studies have shown that improvements in image quality metrics do not necessarily translate into better driving behavior [94]. Images that appear visually realistic may still contain artifacts that cause failures in perception or control modules.

To the best of our knowledge, existing work has not validated these techniques as an automated approach for transforming static datasets into dynamic simulation environments, nor evaluated their effectiveness by directly comparing the behavior of a simulated system with that of a real autonomous driving system operating in the real world. In this work, we address this gap

by evaluating neural rendering techniques for sim-to-real gap mitigation using a behavioral evaluation. Rather than relying solely on image similarity metrics, we compare the behavior of an autonomous driving model operating on generated images with the behavior of the same model driving in real-world conditions. This allows us to assess which generative or neural rendering technique better supports reliable simulation-based testing of autonomous driving systems.

Chapter 3

Framework

In this work, we evaluate the advantages and limitations of different neural rendering techniques employed to reduce the sim-to-real gap in driving simulation, with the goal of supporting system-level testing of camera-based ADS. To this end, we propose a framework for simulation-based testing that enables the recreation of driving scenarios starting from real-world driving recordings. The framework supports the integration of camera-based driving models operating in a simulated scenario. The framework allows the use of different neural rendering techniques to generate input images for the driving model instead of raw synthetic images. We used the proposed framework for comparing how the driving model behaves when provided with images produced by three different neural rendering techniques: Stable Diffusion, NeRF and 3DGS. The effectiveness of each technique is evaluated by comparing the behavior of the driving model on the reconstructed real-world driving against the behavior on the real world scenario collected using a physical vehicle on which the ADS is deployed. This enables a direct comparison between the behavior of the same model in real-world conditions and in simulation, both with and without the application of neural rendering techniques.

3.1 Framework Overview

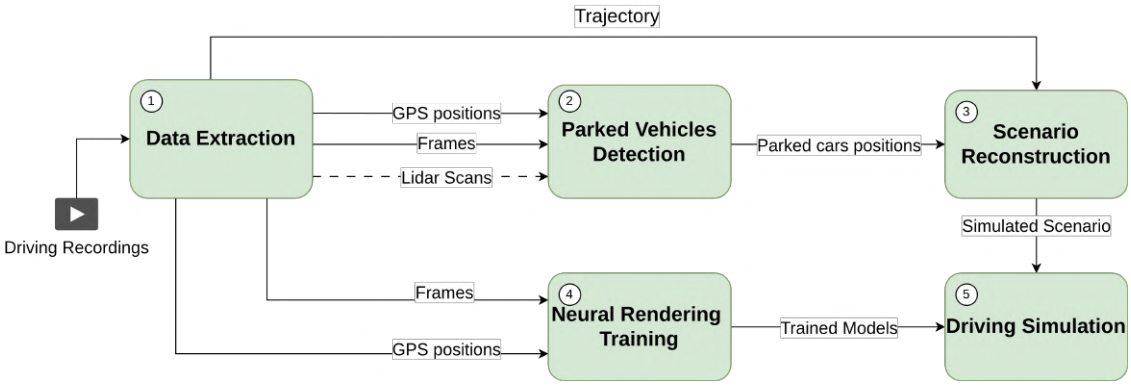


Figure 3.1: Overview of the proposed framework. LiDAR scans are optional and represented as a dotted line.

The framework is organized into five main components visualized in Figure 3.1:

1. Data Extraction. This component extracts the input data required to reconstruct the driving scenario in simulation. It takes as input real-world recordings from an ego vehicle and extracts the data needed for the next components. The output consists of camera frames, LiDAR point clouds (when available), and the ego-vehicle trajectory positions. Each element is associated with its original recording timestamp to ensure synchronization across data sources.

2. Parked Vehicles Detection. This component predicts the positions and orientations of parked vehicles in the recorded scenario, enabling their accurate placement in simulation to faithfully reconstruct the driving scenario. It identifies parked vehicles present in the recorded scenario and estimates their real-world positions and orientations. It takes as input the data extracted by the previous component and performs 3D object detection to predict parked vehicle's bounding boxes. Two approaches are supported: a camera-based method and a LiDAR-based method. The output consists of the positions and orientations of the detected parked vehicles expressed in real-world coordinates.

3. Scenario Reconstruction. This component reconstructs the recorded real-world driving scenario inside the CARLA simulator for system-level testing. It takes as input the ego-vehicle trajectory and the positions of parked vehicles detected by the two previous components. The road geometry is

obtained from OpenStreetMap data, converted to XODR format, and then loaded into the simulator. Parked vehicles are spawned at their estimated positions, and the component also supports manual correction of vehicle placement as well as the creation of modified scenarios for testing configurations beyond the original real-world setup.

4. Neural Rendering Training. This component trains the generative techniques that are the subject of our evaluation. Real-world data extracted in the first component are processed according to the requirements of each model. The framework supports three different techniques: diffusion-based image generation (Stable Diffusion with ControlNet), novel view synthesis from NeRF, and 3D Gaussian Splatting. The trained models are used during simulation to transform or replace synthetic camera images provided by CARLA as input to the driving model.

5. Driving Simulation. This component is used to test the driving model in simulation by executing the reconstructed driving scenario inside the CARLA environment. An ego vehicle is spawned in the CARLA environment, and the simulator manages vehicle dynamics and physics. During execution, the camera images rendered by the simulator can be transformed or replaced using the neural rendering models trained in the previous component. This component supports both trajectory replay, where the vehicle follows the recorded real-world path, and system-level simulation, where a driving model actively controls the vehicle. The former is used to obtain generated images along the recorded trajectory, enabling a direct comparison with their real-world counterparts, while the latter is used to compare the neural rendering techniques in terms of the driving behavior of the tested model. The output consists of the driving trajectories, camera images, and driving commands produced during simulation.

3.2 Data Extraction

The data extraction component takes as input a ROS bag, a file format used by the Robot Operating System (ROS) [87], a widely adopted middleware framework for robotics and autonomous systems, to store timestamped sensor recordings, and extracts the data necessary for the subsequent stages. The ROS bag contains sensor data collected from an ego vehicle driving in the real world. In particular, the framework requires front-facing camera images and vehicle localization data, while LiDAR point clouds are optional.

Each message in the ROS bag is associated with a UNIX timestamp indicating the moment at which it was recorded. These timestamps are essential for synchronizing the different data streams, which operate at different frequencies and are not aligned in time. An overview of the data extraction component is shown in Figure 3.2

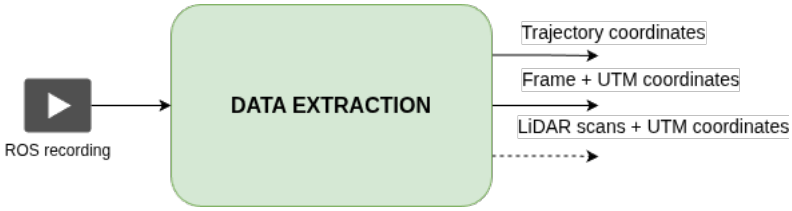


Figure 3.2: Data Extraction Component

3.2.1 Trajectory and Vehicle Pose Extraction

The ego-vehicle trajectory is essential for the entire framework. It is used together with LiDAR scans or camera images to estimate the coordinates of parked vehicles, to reconstruct the scenario in simulation, during inference for 3D rendering methods or for fine-tuning diffusion-based models. Most importantly, it is later used as the baseline for comparison between real-world and simulated driving behavior. The component extracts the vehicle's odometry messages that are in Universal Transverse Mercator (UTM) [88] format coordinates (x, y, z) for vehicle position and quaternions (x, y, z, w) [89] for vehicle orientation. In our case, we compute only the yaw angle from the quaternion, which represents the rotation around the vertical (z) axis and defines the heading of the ego vehicle in the x, y plane. This choice is based on the assumption of planar motion, meaning that the vehicle moves on a flat surface and pitch and roll variations are not relevant for our case. The conversion from quaternion to yaw is performed using the formula:

$$\psi = \text{atan2}\left(2(w \cdot z + x \cdot y), 1 - 2(y^2 + z^2)\right)$$

The resulting odometry messages, in the format (x, y, z, ψ), are saved together with their timestamps and represent the vehicle trajectory through a provided recording.

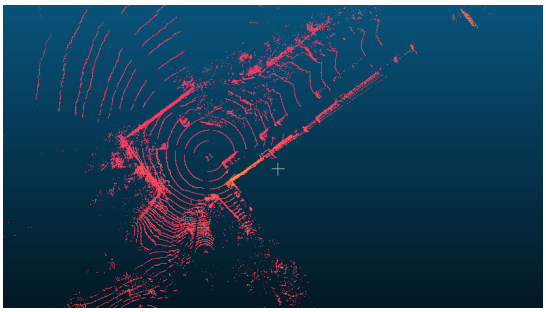
3.2.2 Camera Data Extraction and Synchronization

Images are acquired from the frontal camera mounted on the ego vehicle and employed for parked vehicle detection and for neural rendering model training. For these tasks, each image must be associated with the precise vehicle position at which it was captured, since real-world frame coordinates are needed to match neural rendering positions during both training and inference, and to project vehicle detections from camera reference into world coordinates. However, since camera frames and odometry messages are recorded at different frequencies and their timestamps do not match perfectly, we estimate the vehicle odometry for each recorded image using linear interpolation on the odometry samples recorded immediately before and after the image timestamp. This process results in a dataset where each camera frame is linked with its corresponding coordinates and angle.

3.2.3 LiDAR Data Extraction and Synchronization

The framework also provides the option of using LiDAR point cloud scans. These can be used for parked vehicle detection and during validation to evaluate the accuracy of the reconstructed scenario. However, requiring high-precision LiDAR sensors on the vehicle would be a too strict constraint for the usage of the framework; therefore, it is designed to also operate using only camera data. The acquired LiDAR data consists of dense point clouds, where each point is defined by its spatial coordinates (x, y, z) and a reflectance intensity value (i), which represents the strength of the returned laser signal measured by the sensor. Since LiDAR scans, like camera frames, are recorded at a different frequency than odometry, we associate each scan with its corresponding vehicle position through interpolation, as described for camera frames. An example of a front camera image and its corresponding LiDAR point cloud is shown in Figure 3.3.

(a) Front camera image



(b) LiDAR scan

Figure 3.3: Example of a front camera image and the corresponding LiDAR point cloud recorded by the ego vehicle.

3.3 Parked Vehicles Detection

The second component of the framework is parked vehicle detection. Parked cars along the road represent a crucial element of a driving scenario, since the driving model has to avoid them, also their presence and positions can affect the behavior of a driving model. To faithfully reconstruct a scenario in simulation, parked vehicles must be placed in the simulated environment in the same positions as in the real world. This component takes as input the data extracted in the previous stage and predicts oriented bounding boxes of parked vehicles. The world coordinates of each detected vehicle are estimated based on the ego vehicle position at the corresponding timestamp. Detections are then associated over time using clustering techniques to remove duplicate predictions and maintain a consistent vehicle identity across consecutive data samples. Two approaches are supported: a camera-based method that predicts bounding boxes from input frames, and a LiDAR-based method that relies on point cloud data when available.

An overview of the parked vehicle detection component is shown in Figure 3.4

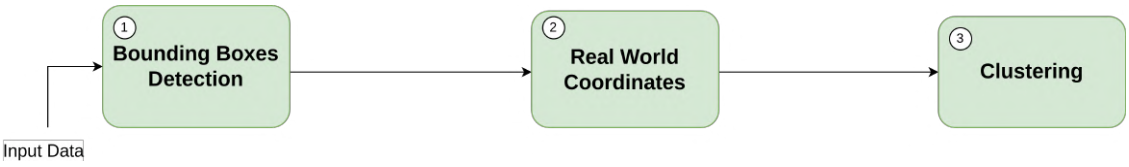


Figure 3.4: Overview of Parked Vehicles Detection component

3.3.1 Camera Based Detection

The camera based approach takes in input extracted camera frames from the first component of the framework and predicts 3D bounding boxes for each vehicle detected in the image, it also generates instance maps for each frame where every detected vehicle gets as color mask the color of the real car that will be used During StableDiffusion finetuning. The component works in a four-stage pipeline: single-frame detection, coordinate transformation, temporal tracking, and final clustering. The final output consists of the coordinates of detected vehicles along with their color and orientation.

Single-frame detection

This stage handles frame-by-frame detection. Given a single frame as input, it leverages deep learning models to detect vehicles in the image, predict their bounding boxes, estimate orientation, and extract the color of each detected car. For 3D object detection, we use FCOS3D[83], a monocular detection framework that consent given an image, detects vehicles and predicts their bounding box parameters: 3D centroid position in camera coordinates, orientation (yaw angle), and a confidence score. An output showing detection of the model is visible in figure 3.5

Figure 3.5: 3D bounding box detection example

The FCOS3D model pretrained trained on the nuScenes[84], a multimodal dataset for autonomous driving, collected using a specific camera configuration. Using the model on images captured with a camera with different intrinsic parameters (focal length, principal point) can introduce systematic errors in depth estimation. To mitigate this issue, the intrinsic calibration matrix of the target camera must be provided to the model at inference time.

To estimate the color of each detected vehicle, we employ YOLOv8 [85] with instance segmentation, which provides a separate pixel-level mask for each detected object. YOLOv8 is chosen for its real-time inference speed and low computational requirements, making it efficient to run alongside the detection pipeline without significant overhead. For each FCOS3D detection, we project its 3D centroid onto the image plane and check whether the projected point falls within any YOLO bounding box; if so, the corresponding instance mask is associated with the detection. The dominant color is then extracted from the lower 60% of the masked region, which predominantly contains the vehicle body, in order to exclude windows from color detection. To reduce noise, pixel values are quantized into discrete bins and the final color estimate is computed as an average of the most frequent quantized colors. An example of the process is shown in Figure 3.6. We classify each

predicted vehicle as "parallel" or "perpendicular" relative to the ego vehicle trajectory. We do this by comparing the predicted yaw angle against the direction of the ego vehicle trajectory; if the difference between the angles is less than 45° or greater than 135°, the vehicle is classified as "parallel", since in both cases the vehicle is approximately aligned with the road direction, either facing the same way or the opposite direction. Otherwise, it is classified as "perpendicular". This binary classification is applied to simplify the placement of detected vehicles in simulation.

Coordinate Transformation

In this stage, we compute coordinates transformation to get real world position of each detected vehicle from the previous stage. We leverage the ego-vehicle position at which the frame was captured from the first component of the framework.

First bounding box centroids are transformed from the camera frame to the ego-vehicle frame using the extrinsic calibration matrix $T_{\text{ego}\leftarrow\text{cam}}$, which encodes the static position and orientation of the camera with respect to the vehicle.

Then, we applied a dynamic transformation matrix $T_{\text{world}\leftarrow\text{ego}}(t)$ to move from the ego-vehicle frame to UTM coordinates, using the ego-vehicle pose at the corresponding timestamp.

The final world position is computed as:

$$p_{\text{world}} = T_{\text{world}\leftarrow\text{ego}}(t) \cdot T_{\text{ego}\leftarrow\text{cam}} \cdot p_{\text{cam}} \tag{3.1}$$

The yaw angle is also transformed to the global frame by passing the direction vector of the bounding box through the same transformation chain:

$$\mathbf{d}_{\text{world}} = T_{\text{world}\leftarrow\text{ego}}(t) \cdot T_{\text{ego}\leftarrow\text{cam}} \cdot \begin{bmatrix} \cos(\theta) \\ 0 \\ -\sin(\theta) \\ 0 \end{bmatrix} \tag{3.2}$$

$$\theta_{\text{world}} = \text{atan2}(d_y, d_x) \tag{3.3}$$

where θ is the predicted yaw in camera coordinates.

Temporal Tracking

The third stage of the vehicle detection pipeline tracks detected vehicles, assigning them a consistent ID over time. It takes as input the real-world UTM

coordinates of each detected vehicle. Since the same vehicle may appear in multiple frames and gets detected multiple times, this stage groups detections belonging to the same vehicle instance using a distance threshold. Each instance stores the current estimated position of the vehicle it refers to, a hit counter, and the list of detections associated with that instance. For each new bounding box detection, the Euclidean distance is computed against all existing instances. If the minimum distance is below a threshold of 2.5 m, empirically determined as the average distance between parked vehicles, the detection is assigned to the closest instance, and both the instance position and hit counter are updated. Otherwise, a new instance is initialized. To avoid false positives, only instances that were detected more than a certain number of times are saved.

Final Clustering

In the previous stage, confirmed vehicle instances contain multiple detections, each one associated with coordinates, orientations and confidence score. In this stage we compute for each instance its final parameters with a weighted average of all the detection parameters, where the detection confidence score serves as the weight:

$$\bar{p} = \frac{\sum_i w_i \cdot p_i}{\sum_i w_i} \tag{3.4}$$

where $\bar{p}$ is the final estimated parameter, p_i is the value of that parameter in the i-th detection, and w_i is the corresponding confidence score. Discrete attributes (binary orientation, color) are determined through majority voting across al the detection assigned to a specific vehicle.

A last clustering step is applied to resulting centroids by merging close instances that may represent the same physical vehicle but were tracked separately. The final output is the list of detected parked vehicles, each one described by: world position (x, y, z), yaw angle, orientation relative to the trajectory (parallel/perpendicular), color (RGB), and average confidence score.

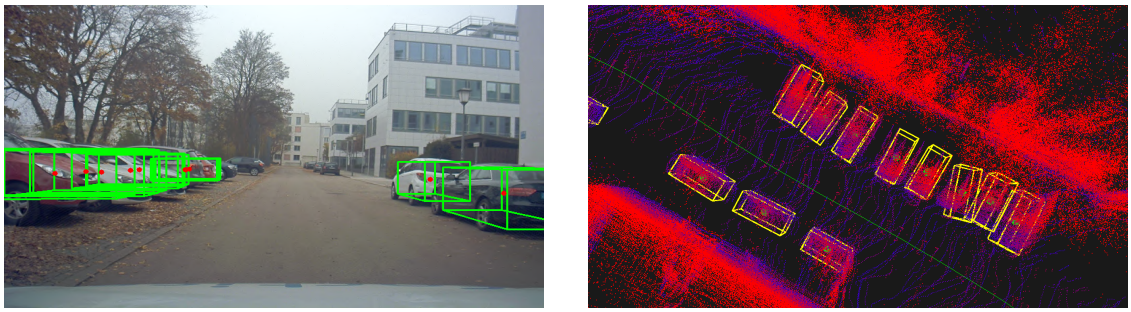
(a) Raw RGB frame (b) Instance map

Figure 3.6: Camera frame and the predicted instance map.

3.3.2 LiDAR Based Detection

The LiDAR-based detection approach follows a similar structure to the camera-based one. It takes as input point clouds generated by the LiDAR sensor instead of RGB images, along with GPS positions from the first component, detects vehicles by predicting 3D bounding boxes, and estimates their position in real-world coordinates. The detection pipeline is divided in three stages: single-scan detection, coordinate transformation, and final clustering. Compared to the camera detection pipeline, the tracking stage is bypassed since LiDAR provides much more precise localization, making a single clustering step enough to aggregate detections.

This approach achieves significantly better accuracy than the camera-based one, because monocular depth estimation from a 2D image is an ill-posed problem, and the model must infer depth from indirect cues like object size and perspective, while LiDAR instead knows exactly the distance of each point, providing precise depth information. Additionally, LiDAR handles occlusions more effectively than camera: for example, with multiple vehicles parked in a row, a front camera may see some vehicles only partially and for a fraction of time, making it difficult to detect their bounding boxes, while LiDAR does not have this problem, as shown in Figure 3.7. For these reasons, when LiDAR is available, this approach provides more reliable results.



(a) Camera detection (b) LiDAR detection

Figure 3.7: Vehicle detection comparison: camera limitations vs LiDAR accuracy in occluded scenes.

Single scan detection

In this first stage, the component takes as input a single LiDAR scan and uses deep learning methods to detect vehicle bounding boxes in the point cloud, outputting for each detected vehicle the bounding box parameters: position (x, y, z), and orientation. For 3D object detection on point cloud data, we use PointPillars [86], a model that encodes the 3D point cloud into a 2D bird's-eye view representation and predicts bounding boxes with confidence scores. The model is pre-trained on KITTI [39], a multi-modal dataset for autonomous driving containing synchronized camera and LiDAR data. The model expects input distributions matching the KITTI sensor setup. Using it with different input data will cause a domain shift leading to incorrect predictions, so to apply it on a different sensor configuration without finetuning, the input data must be normalized to match the training data distribution. To do so, we perform the following steps: In the KITTI dataset, LiDAR point intensity is stored in a float range $[0, 1]$, while common LiDAR sensors output intensity in $[0, 255]$. So to match the expected output we normalized our input data to be in the same range as KITTI data. Since LiDAR sensor in KITTI dataset is mounted at a fixed height of 1.73 meters above the ground, the model has learned features based on this absolute positioning. To account for variations in sensor mounting on different vehicles, we estimate the ground level by computing the 5th percentile of the z-coordinates in each scan, then shift the entire point cloud vertically so that the ground aligns with $z = -1.73$m. This prevents the detector from misclassifying objects due to height offsets.

The pointcloud obtain from LiDAR scan provided by the data extraction component3.2.3 is normalized and then passed to the PointPillars model,

which predicts bounding box parameters for each detected object: 3D centroid position (x, y, z), orientation (yaw angle), and a confidence score.

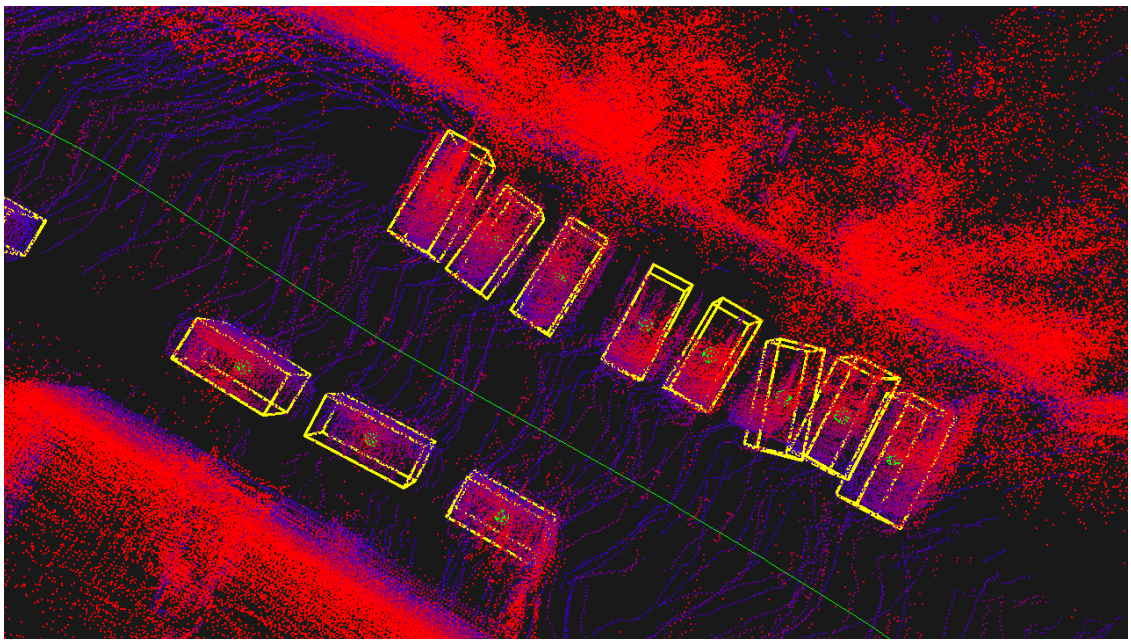


Figure 3.8: 3D bounding box detection in LiDAR point cloud

Coordinates Transformation

As in the camera-based approach, we need to project coordinates to the UTM reference frame. Each predicted centroid is first transformed from the LiDAR sensor frame to the ego-vehicle frame using the extrinsic calibration matrix $T_{\text{ego}\leftarrow\text{LiDAR}}$, which encodes the LiDAR position and orientation with respect to the ego-vehicle. Then, to obtain UTM coordinates, we apply the dynamic transformation matrix $T_{\text{world}\leftarrow\text{ego}}(t)$ from the ego-vehicle frame to world coordinates, using the vehicle pose at which the scan was recorded. The yaw angle is also transformed by rotating the direction vector through the same transformation chain.

Final Clustering

In this stage of the vehicle detection pipeline, we receive the predicted parameters of each detected bounding box in each scan. As in the camera-based approach, the same vehicles are detected multiple times in different scans, so to aggregate them, we use clustering. For each new detection, we compute the Euclidean distance against already detected clusters. If the distance

is below a certain threshold, the detection is merged into the closest cluster and the cluster centroid position is updated; otherwise, a new cluster is initialized.

We compute final bounding box parameters by the weighted average of all detections in the cluster, using confidence scores as weights. For the centroid position the weighted average is computed as:

$$\bar{p} = \frac{\sum_i w_i \cdot p_i}{\sum_i w_i} \tag{3.5}$$

where w_i is the confidence score and p_i is the positions of detection i. For the yaw angle, a circular mean is used to correctly handle angle wrapping:

$$\bar{\theta} = \text{atan2}\left(\sum_i w_i \sin(\theta_i), \sum_i w_i \cos(\theta_i)\right) \tag{3.6}$$

where θ_i is the yaw angle of detection i. This avoids errors that would occur with a simple arithmetic mean when angles are near $\pm\pi$. For example, averaging two angles of $-170°$ and $170°$ with a standard mean would yield $0°$, whereas the correct result is $\pm 180°$. The circular mean resolves this by operating on the sine and cosine components of the angles, correctly handling the discontinuity at $\pm\pi$. Binary orientation classification (parallel/perpendicular) is computed using the same logic as the camera-based approach. Only clusters with a minimum number of detections are retained in the final output.

The final output is the list of detected parked vehicles, each one described by: world position (x, y, z), yaw angle, orientation relative to the trajectory (parallel/perpendicular) and average confidence score.

3.4 Scenario Reconstruction

This component is responsible for recreating the recorded driving scenario inside the CARLA simulator. The component is divided into four stages: downloading the map corresponding to the driving recording, manual editing to correct parked vehicle positions, converting coordinates from UTM to the CARLA coordinate system, and loading the scenario into the simulator.

An overview of the scenario reconstruction component is shown in Figure 3.9.

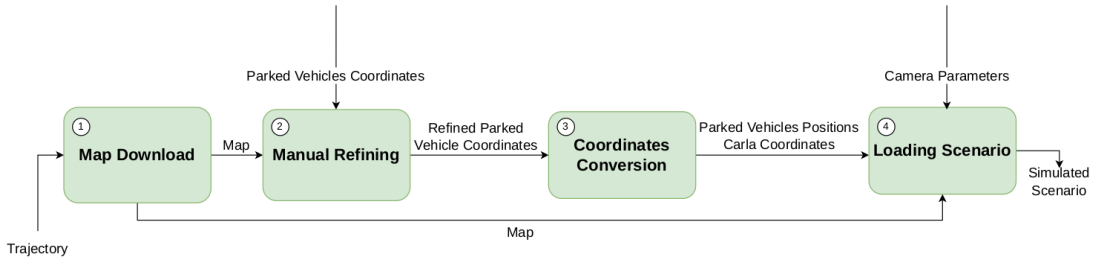


Figure 3.9: Scenario Reconstruction component overview

3.4.1 Map Download

To be able to recreate a driving scenario, we must provide the corresponding map to the next component of our framework, which will use it to recreate the road topology in simulation. To obtain the map of the driving scenario, we use OpenStreetMap's Overpass API [44]. Using the coordinates of the starting position of the ego-vehicle trajectory, we query the API to download the corresponding area as a .osm file. This file is then converted to XODR format using CARLA's built-in converter, as it is the format supported for custom maps in CARLA.

3.4.2 Manual Refining

The framework provides an interactive tool for editing the driving scenario in order to manually correct errors of parked position coming from the previous component 3.3. The ego-vehicle trajectory and detected parked vehicles are overlaid on the downloaded OSM map, allowing the user to manually correct detection errors by inserting new vehicles, removing false positives, or adjusting the position and orientation of existing ones.

An example of the tool is shown in Figure 3.10

Figure 3.10: Manual refining tool with trajectory and detected parked vehicles displayed over the OSM map

3.4.3 Conversion to CARLA coordinates

In this stage we convert parked vehicles' positions and trajectory coordinates from UTM, given by previous components, to the CARLA coordinate system to correctly load them in the simulation as we want to replicate the parked vehicles configuration observed in the driving scenario.

Coordinates must be projected from the curved Earth surface to a flat 2D plane, due to CARLA using planar metric coordinates. The conversion takes four steps:

1. **UTM → WGS84**: Universal Transverse Mercator (UTM) coordinates divides the world into 60 zones along meridians. The X value represents meters east from the central meridian of the zone, plus a 500 km offset to avoid negative values for locations west of the central meridian, the Y value represents meters north from the equator and the Z value indicate the current altitude above sea level. We need to convert UTM to WGS84

(latitude/longitude) as an intermediate step required by the XODR map that uses a different projection than UTM. The World Geodetic System 1984 (WGS84) [73] is the standard geographic coordinate system (latitude/longitude in degrees) that serves as a universal intermediate format from which any projection can be computed.

2. **WGS84 → XODR**: From WGS84 we project coordinates to the 2D XODR format supported by CARLA. We use the projection string specified in the `.xodr` file downloaded in the previous step for computing the projection.

3. **XODR → XODR Local**: Since CARLA does not use XODR coordinates directly but instead uses a local coordinate system, the projected coordinates are shifted by the offset values specified in the `.xodr` header, transforming them into the local coordinate system used by CARLA.

4. **XODR Local → CARLA World**: The Y-axis is flipped to match CARLA's left-handed coordinate system.

An overview of the conversion scheme is provided in Figure 3.11

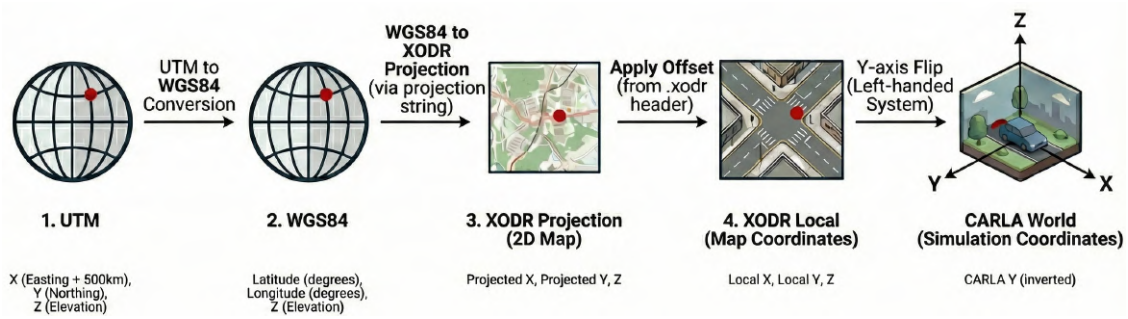


Figure 3.11: Coordinates conversion scheme

Z values are not converted since XODR format supported by CARLA does not provide altitude values.

For trajectory coordinates, an additional correction step is applied to account for the GPS sensor position relative to the ego-vehicle center. Since CARLA spawns vehicles with their center located at the given coordinates, passing the raw GPS coordinates would result in a shifted trajectory, as trajectory positions are relative to the GPS sensor location, not the center of the car. To correct this, the trajectory coordinates are shifted along the heading direction by the distance between the GPS sensor and the vehicle center. We

do not have the same problem with parked vehicles since their coordinates predicted by our previous component 3.3 already represent the center of the car.

The yaw angle ψ also requires conversion from UTM to CARLA coordinates. The coordinate conversion chain introduces a rotation of the reference axes, meaning that a given heading direction in UTM does not correspond to the same angle in CARLA. To determine this rotation, we convert two points through the full coordinate chain: a reference point and a second point displaced by one meter along the UTM East axis. By measuring the angle between the two resulting points in CARLA space, we obtain the angular offset A between the two coordinate systems. The UTM yaw angle is then converted to CARLA yaw as:

$$\psi_{CARLA} = A - \psi_{UTM} + \frac{\pi}{2} \tag{3.7}$$

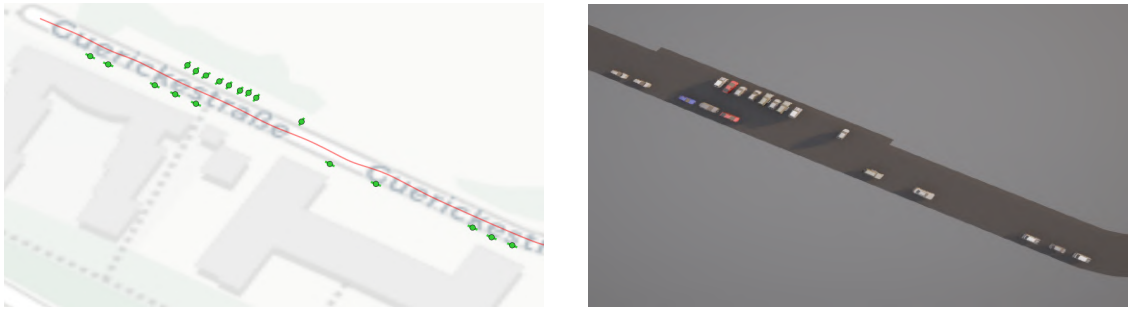
where the subtraction accounts for the reversal of rotation direction caused by the Y-axis flip, and the $\frac{\pi}{2}$ term corrects for the difference between the quaternion yaw convention used in the odometry data and the heading convention in CARLA.

3.4.4 Loading Scenario

This is the final stage of the scenario reconstruction component, which loads the driving scenario, comprising the map and the parked vehicles, into simulation which will be used for our evaluation. Using the provided CARLA Python API, this stage starts the CARLA server, which runs the simulation engine responsible for rendering the environment and managing vehicle physics, and loads the driving scenario into the simulation. It takes as input the map obtained in the previous stage, parked vehicle coordinates obtained in Parked Vehicles Detection component 3.3, and the ego-vehicle trajectory from the Data Extraction Component 3.2. First, it loads the map into CARLA. Then, it spawns the ego-vehicle at the first point of the trajectory and attaches three sensors: an RGB camera, a semantic segmentation camera, and an instance segmentation camera. The outputs of these sensors will serves for image generation in the next final component of our framework.

For each parked vehicle in input, the script spawns a car at the corresponding CARLA coordinates with the correct orientation matching the parked vehicles configuration observed in the driving scenario. The vehicle is spawned with the same color as detected in the real world from the detection component, and this color is also used as its mask in the instance segmentation

map, that will be used by further steps to recreate parked cars of the same vehicle as it was in the real world. Finally, the simulation is paused with all vehicles in position, ready for driving simulation, as shown in Figure 3.12



(a) OSM map (b) Reconstructed scenario

Figure 3.12: Scenario reconstruction comparison between OSM data and final reconstructed scenario in CARLA.

3.5 Neural Rendering Training

This component handles the training of the neural rendering models: Stable Diffusion, NeRF, and 3D Gaussian Splatting. It takes as input the recorded frames and their corresponding coordinates, and processes them according to each model's requirements to create the training datasets. The output consists of the trained models that will be used during simulation to produce more realistic input images for the driving model than raw synthetic images from the simulator. For all three techniques, the framework supports local-expert training [66], where the dataset is split into road segments and a separate model is trained for each segment. This allows each model to focus on a smaller portion of the driving scenario and become better specialized to the visual appearance of the road segment on which it is trained. Similar approaches have been adopted in large-scale neural rendering works such as Block-NeRF and Mega-NeRF [66, 67]. An overview of the pipeline is shown in Figure 3.13

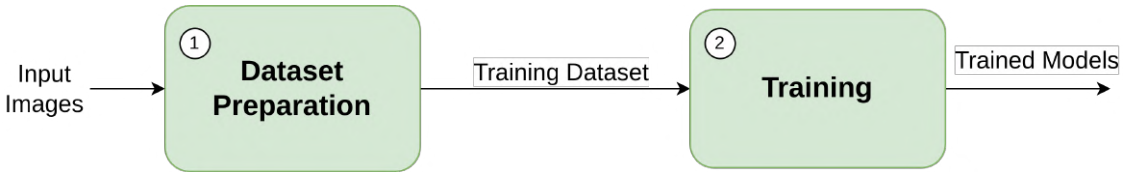


Figure 3.13: Models Training component overview

3.5.1 Stable Diffusion

We fine-tune Stable Diffusion 1.5 [62] using Low-Rank Adaptation (LoRA) [90], a parameter-efficient fine-tuning method, on frames from the recorded driving scenario. To guide the generation process, we fine-tune three ControlNets[71]: a Segmentation ControlNet[91], conditioned on semantic segmentation maps, which provides the main scene structure by specifying where to place roads and vehicles; an Instance ControlNet, conditioned on instance segmentation maps, which helps distinguish between individual vehicles; and a Temporal ControlNet[92], conditioned on the previous frame, which enforces temporal consistency between consecutive frames. At inference time, the Segmentation ControlNet is always required, while the Instance and Temporal ControlNets can be optionally enabled.

Dataset Preparation

Semantic segmentation maps are used to condition the Segmentation ControlNet on the scene structure. For each input frame, we generate the semantic segmentation map using SegFormer[91], a transformer-based segmentation model pre-trained on the Cityscapes[38] dataset. The model predicts a class label for each pixel, which is then recolored according to the Cityscapes color palette. Since CARLA with custom XODR maps only renders roads, road signs, and vehicles, as discussed in Section 2.2.3, we recolor to black all pixels not belonging to these classes. This ensures that the real-world segmentation maps used during training match the segmentation maps from the simulator used at inference time.

(a) Input image (b) Segmentation map

Figure 3.14: Example frame and its corresponding semantic segmentation map

Instance segmentation maps are obtained from the Parked Vehicle Detection component 3.3. Unlike semantic segmentation maps, where all vehicles share the same color, instance maps assign a unique color to each individual vehicle. This helps during the diffusion process to distinguish between separate vehicles, particularly when multiple cars are close together, preventing the model from merging them into a single object. Additionally, each vehicle's mask uses the real-world color of the corresponding car, guiding Stable Diffusion to generate vehicles with the correct colors.

Temporal Links.

For temporal consistency, each frame is paired with its preceding frame enabling Temporal ControlNet to learn correlation between consecutive frames during training.

Coordinate Captions.

A caption containing corresponding ego-vehicle position coordinates is associated to each input frame to help Stable Diffusion associate features with specific locations.

Image Preprocessing.

All images are resized to 512×512 pixels, the resolution on which Stable Diffusion 1.5 was trained. Segmentation and instance maps use nearest-neighbor interpolation to preserve exact pixel values when resized.

Dataset Compilation.

The final dataset is converted into HuggingFace[50] Arrow format, embedding all images (RGB, segmentation, instance, previous) and captions in a single binary file for efficient data loading during training.

Stable Diffusion Training

We used LoRA[90] to fine-tune Stable Diffusion to our downstream task. LoRA, is an approach originally developed for large language models[90]. It works by freezing the pre-trained model weights and injecting small trainable weight matrices into each transformer layer. Only these matrices are updated during training, reducing time and memory requirements. The result is a small set of updated weights (a few MB) that, when combined with the original model at inference time, produce fine-tuned results. Training uses a learning rate of 1e-4 with cosine scheduling for 10 epochs.

Segmentation ControlNet.

The Segmentation ControlNet is pre-trained on Cityscapes[38], then fine-tuned on our segmentation maps. Training on paired data (segmentation map → real image) teaches the model to generate realistic images that follow the scene layout specified by the segmentation input.

Temporal ControlNet.

The Temporal ControlNet is initialized from TemporalNet[92], then fine-tuned using frame pairs consisting of the current frame and the previous frame. This enforce the diffusion process to be consistent across consecutive frames.

Instance ControlNet.

The Instance ControlNet is trained from scratch on instance segmentation maps and corresponding frames. This allows the model to learn how to distinguish individual vehicles and preserve their real-world colors.

All ControlNets are trained for 10 epochs with a learning rate of 1e-5.

Local-Expert Training.

As described in the component overview, the framework supports training separate Stable Diffusion and ControlNet models for different road segments. Each model specializes in generating images of a specific segment, and during inference the model whose training segment is closest to the current vehicle position is automatically selected. In training we split the training dataset in of the driving scenario in three different subset with the same frames and we train a separate StableDiffusion, Segmentation, instance and Temporal ControlNet for each of them.

3.5.2 NeRF and 3D Gaussian Splatting

NeRF[59] and 3D Gaussian Splatting[60] share the same training pipeline. We use Nerfstudio[82], an open-source tool that provides APIs for training

different neural rendering models. We used nerfacto[82] and splatfacto[82], Nerfstudio's implementations of NeRF and 3D Gaussian Splatting respectively. Specifically, we employed nerfacto-big[82] and splatfacto-big[82], which are enhanced variants of nerfacto and splatfacto with more parameters for higher quality reconstruction.

Camera Pose Estimation

Camera positions and orientations are required for training. We estimate them using COLMAP [74], a structure-from-motion pipeline that extracts visual features from each frame, matches them across frames, and triangulates 3D points to estimate camera poses. COLMAP also generates a sparse point cloud, which is used by 3D Gaussian Splatting to initialize the gaussians.

Dataset Preparation

Input images are cropped from the bottom to remove the vehicle hood, which would otherwise interfere with the 3D reconstruction.

Training

Both models are trained with default Nerfstudio settings using the camera poses and point cloud generated by COLMAP.

Local-Expert Training

As with the Stable Diffusion pipeline, the framework supports training separate models for different road segments. For NeRF and 3D Gaussian Splatting, COLMAP is run independently for each segment, producing a separate set of camera poses and point cloud per local expert.

3.6 Driving Simulation

This component is the final stage of the framework. It is responsible for running the simulation and using the trained models to generate the images that serve as input to the driving model. It interfaces with the previously loaded scenario and supports two modes of operation: trajectory replay, in which the ego vehicle follows the trajectory recorded from the ADS in the

real world, and system-level simulation, in which the driving model actively controls the vehicle within the simulation.

3.6.1 Trajectory Replay

In this mode, the ego-vehicle follows the recorded real-world driving trajectory and captures the simulator outputs (RGB, semantic segmentation, and instance segmentation), together with the images generated by each of the considered generative models, at the positions corresponding to the original real-world frames. This enables a direct comparison between real-world images, simulator outputs, and model-generated images from the same viewpoint. Such a comparison is useful both for assessing the quality of the reconstructed scenario and for evaluating the different generative approaches in terms of image quality with respect to the corresponding real-world images.

3.6.2 System-Level Simulation

In this mode, the driving model is loaded and enabled to control the ego-vehicle. Rather than replaying the recorded real-world trajectory, the vehicle is initialized at the starting point of that trajectory, and its steering commands are then predicted by the driving model based on the input images provided during the simulation. Similarly to trajectory replay mode, the driving model can take as input either the simulator outputs or the images generated by each of the considered generation approaches, namely Stable Diffusion, NeRF, and 3D Gaussian Splatting.

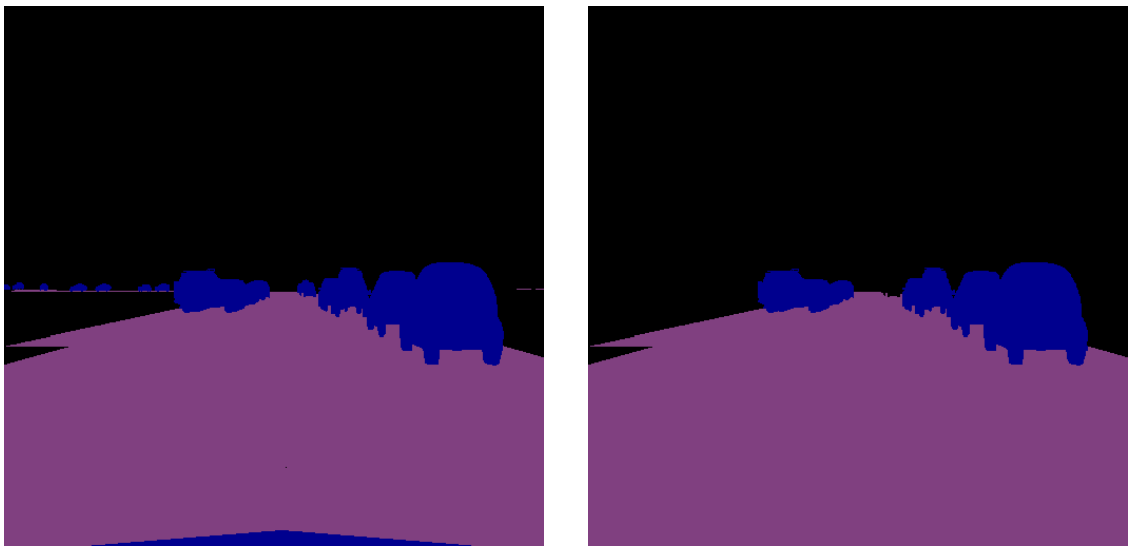
The simulation produces the trajectory generated under the control of the driving model. This trajectory can then be compared with the real-world trajectory recorded during the original driving session. In our evaluation, this comparison is used to assess which image generation technique leads to driving behavior that is most consistent with the real-world one, thus providing a system-level evaluation of the different approaches.

3.6.3 Stable Diffusion Inference

The generation process uses the LoRA weights and the ControlNets fine-tuned during the neural rendering training stage described in Section 3.5. Among these conditioning modules, the Segmentation ControlNet provides the main guidance for the Stable Diffusion generation, as it supplies the

structural and semantic information of the scene. At each simulation frame, the pipeline receives the segmentation map produced by the simulator and uses it as the primary conditioning input for image generation.

However, the simulator segmentation maps must be adapted so that they more closely match the real-world segmentation maps used to fine-tune the ControlNet. In particular, pixels at the bottom of the image corresponding to the front hood of the ego-vehicle are reassigned to the road class, consistently with the real-world segmentation maps. Additionally, when CARLA uses XODR-loaded maps, it does not render several external scene elements, such as buildings or vegetation. As a result, some vehicles that were occluded in the real-world images become visible in the simulator segmentation maps. To mitigate this mismatch, small isolated segmented regions below a given pixel threshold are removed from the segmentation map. An example of these adaptations is shown in Figure 3.15.



(a) Raw segmentation map (b) Cleaned segmentation map

Figure 3.15: Segmentation map before and after cleaning. Small isolated regions are removed and the vehicle hood area is set to road class.

The Instance ControlNet can be enabled to receive the instance segmentation map as input. This provides the distinction between individual vehicles that the semantic segmentation map does not capture, and helps guide the generation of correct vehicle colors. The Temporal ControlNet can also be

enabled, taking the previously generated frame as input to enforce temporal consistency between consecutive frames. The current ego-vehicle coordinates in the simulation are used as the text prompt. Since these coordinates follow the same format used during fine-tuning, the model can leverage the learned association between positions and visual features to generate location-appropriate images.

Several parameters can be configured during image generation:

- **Guess mode**: when enabled, the ControlNets operate without explicit text prompt guidance
- **Conditioning scale**: controls the influence of each ControlNet on the final output (range 0.0 to 1.0)
- **Guidance schedule**: defines the start and end points of each ControlNet's activation during the denoising process (range 0.0 to 1.0, where 0.0 is the beginning and 1.0 is the end of the denoising steps)
- **Negative prompt**: specifies undesired features to avoid in the generated image (e.g., "blurry, distorted")
- **Guidance scale**: controls the adherence to the text prompt

The selection of these parameters is subject to evaluation.

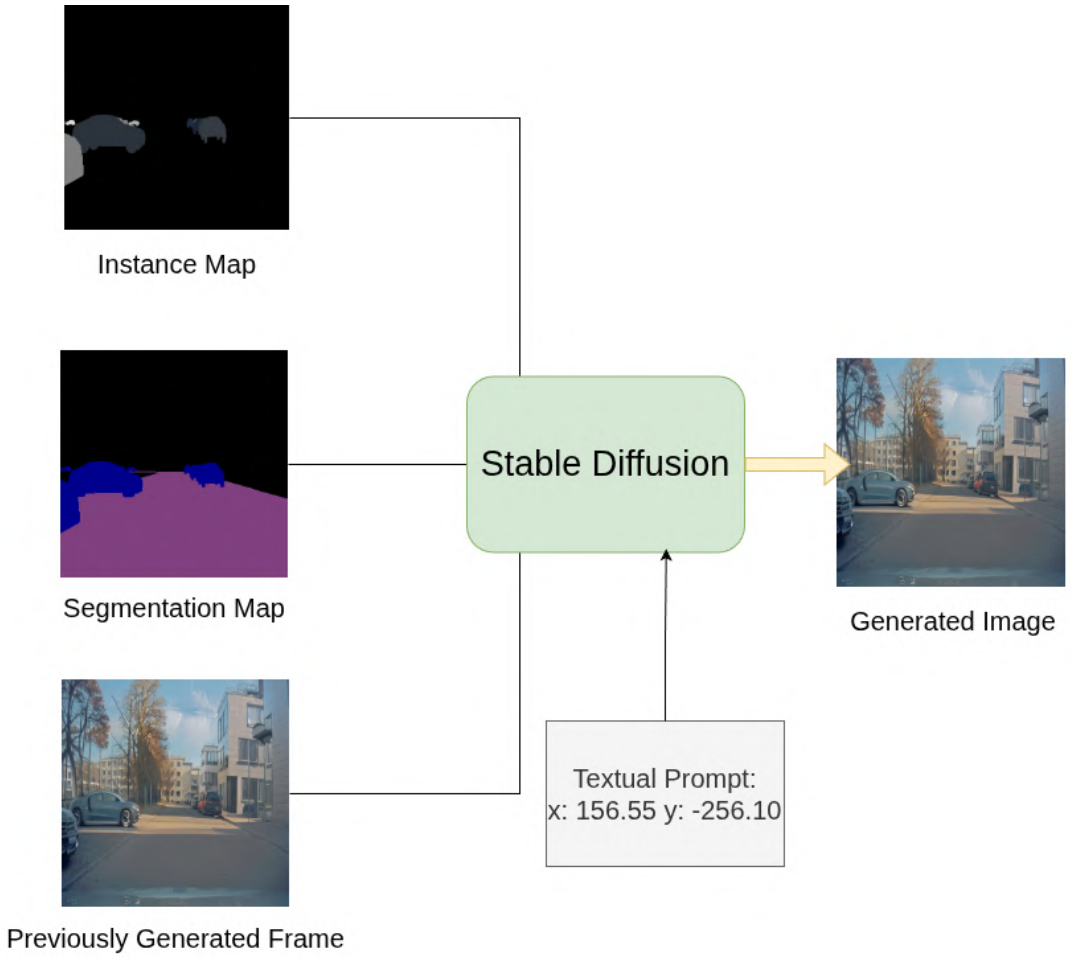


Figure 3.16: Overview of the Stable Diffusion inference pipeline. The segmentation map, and instance map from the simulator and previous generated frame are provided as conditioning inputs through their respective ControlNets, while the ego-vehicle coordinates are used as the text prompt. The Stable Diffusion model generates a realistic image conditioned on these inputs.

3.6.4 NeRF and Gaussian Splatting Inference

Unlike Stable Diffusion, which attems to transforms synthetic images into more realistic ones, NeRF and 3D Gaussian Splatting render views directly from their trained 3D representations based on a given camera position and orientation. At each simulation step, the ego-vehicle position and orientation in CARLA are converted to the Nerfstudio coordinate system, and the corresponding model renders the image from that viewpoint. Both techniques

used Nerfstudio and share the same inference pipeline. An overview of the NeRF and Gaussian Splatting Inference pipeline is shown in Figure 3.17.

Coordinate Alignment

Since COLMAP estimates camera poses from visual feature matching rather than using the original GPS/UTM coordinates, the resulting poses may differ from the ground-truth positions. Additionally, Nerfstudio applies internal transformations to COLMAP data: it centers the scene by moving the centroid to the origin, normalizes the scene size by applying a scale factor, and may reorder or flip coordinate axes. As a result, no direct conversion exists between simulation coordinates and Nerfstudio coordinates. To address this, we compute a set of transformations that map simulation coordinates to the Nerfstudio model space.

The alignment is performed by loading the trained Nerfstudio model and extracting the camera positions and orientations used during training. Each training camera is matched to its corresponding frame in the original recording using frame IDs, and the UTM coordinates are loaded from the trajectory data.

Position Alignment. For the horizontal position, we compute a 2D similarity transform using Umeyama's method, which estimates a scale factor s, rotation matrix R, and translation vector t that minimize the alignment error across all matched points:

$$\min_{s,R,t} \sum_{i} \|P_{ns,i} - (s \cdot R \cdot P_{utm,i} + t)\|^2 \tag{3.8}$$

where $P_{ns,i}$ is the camera position in Nerfstudio coordinates of frame i, and $P_{utm,i}$ is the corresponding UTM position. The resulting transform is applied at runtime to convert simulation coordinates to Nerfstudio coordinates:

$$P_{nerfstudio} = s \cdot R \cdot P_{utm} + t \tag{3.9}$$

Height Estimation. CARLA with custom XODR maps does not provide elevation information, so the Z coordinate cannot be obtained from the simulator. Instead, it is interpolated at runtime from the training camera positions in Nerfstudio space. Given the current vehicle position in Nerfstudio X/Y coordinates, we use Delaunay triangulation on the training camera positions to interpolate the corresponding Z value.

Yaw Alignment. The yaw angle, which represents the vehicle rotation along the horizontal plane, is aligned separately from the position transform.

For each matched frame, we extract the yaw from the Nerfstudio camera-to-world matrix and compare it with the corresponding UTM yaw from the odometry data. We determine the mapping by testing two hypotheses ($\psi_{ns} = +\psi_{utm} + \beta$ and $\psi_{ns} = -\psi_{utm} + \beta$), where ψ_{ns} denotes the yaw angle in the Nerfstudio reference frame, ψ_{utm} denotes the yaw angle in the UTM reference frame obtained from the odometry data, and β denotes a constant angular offset between the two yaw conventions, and selecting the one with the smallest residual variance. The offset β is computed as the circular mean of the per-frame differences.

Camera Pitch and Roll. Since the camera is physically mounted at a fixed position on the vehicle, its pitch and roll remain constant throughout the recording. We extract these angles as the median pitch and roll across all training cameras positions in the Nerfstudio model, filtering out small variations due to COLMAP estimation noise. These angles are applied during rendering to match the real camera orientation.

Figure 3.17: Overview of the NeRF and 3DGS inference pipeline. The ego-vehicle position and orientation in CARLA are converted to the Nerfstudio coordinate system through the alignment transformations, and the corresponding model renders the image from the resulting viewpoint.

3.6.5 Local Expert Selection

As described in the training component, the framework supports splitting the driving scenario into different segments and training multiple models, one for each segment. During inference, the model trained on the segment of the road where the ego-vehicle is currently located is automatically selected based on the current ego-vehicle position in the simulator. This applies to all three neural rendering pipelines.

Chapter 4

Evaluation

The objective of this chapter is to evaluate the advantages and limitations of different neural rendering techniques for image generation in system-level testing of autonomous driving systems. To this end, we compare a real-world driving scenario with the same scenario reconstructed in simulation using the proposed framework. The evaluation begins by examining whether the real-world scenario can be reconstructed faithfully in simulation, with particular attention to parked vehicle placement and overall scene consistency. It then considers whether the generated images produced by the neural rendering techniques resemble the corresponding real-world observations. Finally, it investigates whether differences at the reconstruction and image levels affect the behavior of the driving model when it is tested in the reconstructed scenario. The study is conducted using recordings collected with a real vehicle in the target scenario under multiple weather conditions. Overall, the evaluation is organized around three research questions addressing reconstruction fidelity, image quality, and system-level behavior.

4.1 Evaluation Overview

We use the proposed framework to recreate the selected driving scenario in simulation under three weather configurations. Since the recordings were collected on different days, the arrangement of parked vehicles varies across conditions, making each reconstructed scenario slightly different. For each weather condition, one representative run is used to reconstruct the scene and train the corresponding neural rendering models. This setup allows us to evaluate the framework from three complementary perspectives. First, we study how accurately the recorded real-world scenario is reconstructed

in simulation. We then examine how closely the images generated by the considered neural rendering techniques match the corresponding real-world observations. Finally, we analyze how the driving model behaves when operating on these generated images in simulation and compare this behavior with that observed in the corresponding real-world runs. In this way, the evaluation progresses from the reconstruction of the environment, to the realism of the generated sensor data, and ultimately to their effect on autonomous driving behavior.

(RQ1) Scenario Reconstruction Fidelity

How accurately does the parked vehicle detection component of the framework estimate the coordinates of parked vehicles in real-world driving scenarios? To what extent does the framework reconstruct the recorded real-world driving scenario in simulation?

(RQ2) Generated Image Quality

Do Stable Diffusion, NeRF, and 3D Gaussian Splatting (3DGS) generate images that resemble real-world observations from the same scenario? Which of these techniques produces images that are most similar to their real-world counterparts?

(RQ3) System-Level Behavior

How does the driving model behave in system-level testing when synthetic images generated by different neural rendering techniques are used, compared with its behavior in real-world driving in the same scenario?

4.2 Driving Scenario

We define a driving scenario as a stretch of road where a vehicle can operate. The same driving scenario can present different configurations, with varying parked vehicle placements, weather conditions, and traffic actors. Our driving scenario is located on Guerickestraße, a public urban street in Munich, Germany. It is a two-way street with no lane markings, approximately 400 meters long with an average width of 7 meters. The red line in Figure 4.1 represents the 365-meter stretch used for our experiments. The street has sidewalks on both sides and two parking areas, one at the beginning and one

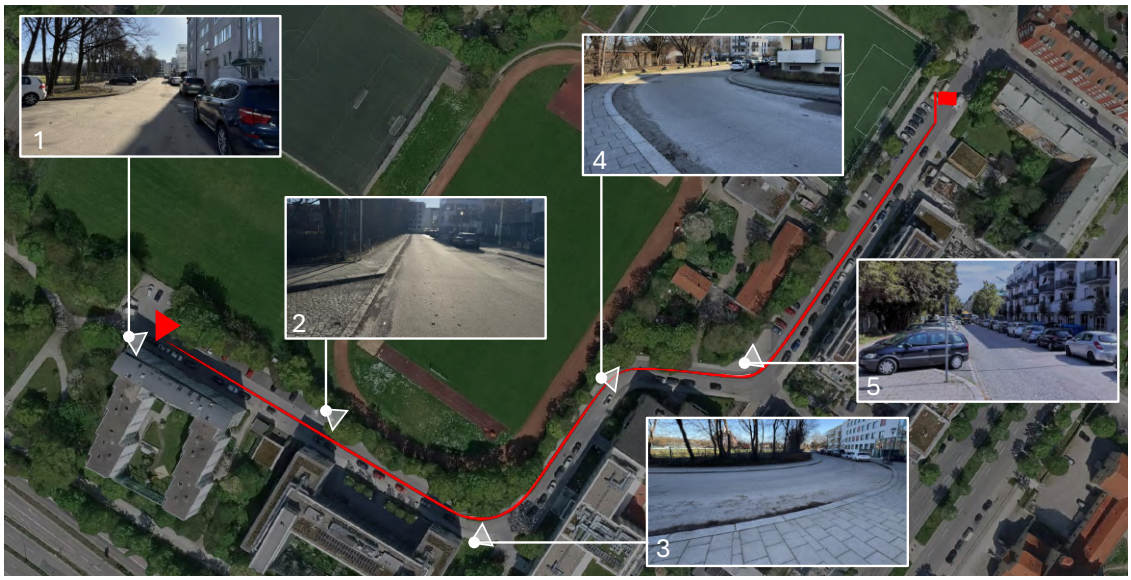


Figure 4.1: Top down overview of the scenario used in our implementation.

at the end, both on the left side in our driving direction. On the right side, cars are frequently parked along the road edges. The street has three curves, no pedestrian crossings, and no possible turns. It was selected because it has a speed limit of 30 km/h and low traffic, ensuring safe conditions during testing, ...say its still not trivial with interesting conditions suche as cars parked on the sidews and being difficult to navigate... do not invenet anything new just extrapolatre info from this section

4.3 Vehicle Platform

For collecting our datasets we used Fortuna [93], a modified Volkswagen Passat GTE prototype vehicle developed by fortiss[13] for the project PROVIDENTIA [47]. The vehicle is equipped with a front-facing camera with 60° horizontal Field of View (FOV), three Velodyne [43] LiDARs mounted on the roof rack (one VLP-32C with 32 layers and 200m range positioned centrally, and two VLP-16 with 16 layers tilted at each side to cover lateral areas, providing 360° FOV), and an iMAR [104]iNAT FSSG-1 GNSS/INS system providing RTK positioning with up to 2cm accuracy. Sensor data were recorded using ROS.

Figure 4.2: Passat GTE Fortuna

4.4 Datasets

To answer RQ1 and RQ2, we need a dataset containing recordings of an ego-vehicle driving in the real world, with front-facing camera images and trajectory coordinates. For RQ3, we additionally require that the recordings come from an ego-vehicle controlled by a driving model rather than a human driver, and we need access to the same driving model to test it in simulation. This is necessary to establish a baseline for evaluating how different image generation techniques impact the behavior of the driving model at system level. While there are numerous public datasets for autonomous driving, they are primarily designed for machine learning training rather than testing. Most of them consist of recordings where a human drives the vehicle, not a driving model. We were not able to find a dataset with all the characteristics necessary to answer RQ3.

Existing datasets such as nuScenes [84], KITTI [39], and Waymo Open Dataset [40] provide sensor data and annotations but do not provide steering data and focused on perception tasks. The Comma.ai dataset [41] includes steering data but is recorded by human drivers.

For these reasons, we collected a training dataset that will be used for training a driving model, and a testing dataset in which the trained driving model autonomously controls the vehicle in the selected driving scenario. The resulting testing dataset will be used for our empirical evaluation. We chose not to use a pre-trained model because we need a model that drives consistently on a specific road but does not generalize to different environments. This allows us to isolate the effectiveness of the neural rendering

techniques at reconstructing the scenario in simulation, rather than evaluating the model's ability to generalize to unseen data. Since the model needs to reliably drive on our driving scenario, the training dataset was collected on the same road.

4.4.1 Training Dataset

The training dataset consists of 150 recordings sessions collected by the Fortuna ego vehicle driven by a human driver on the selected driving scenario across 4 days in October between 09:00 and 16:00, under two different weather conditions: sunny and cloudy. This variability was necessary for training a model able to generalize across different lighting conditions and parked car configurations, ensuring consistent driving behavior in conditions similar to those encountered during training. We collected images from the front-facing camera along with the steering angle applied by the human driver and vehicle speed. Examples of collected images are shown in figure 4.3. Images were recorded at a frequency of 30 fps with a resolution of 800×503 pixels. Data were collected at an average speed of 10.5 km/h. For each frame, we saved a JSON file containing the corresponding steering angle and speed values. Steering values are expressed in radians, ranging from -3π to $+3\pi$

The dataset is characterized by two types of recordings:

- **Nominal runs** (20 recordings): the driver covers the full length of the driving scenario, staying as close to the center as possible. The average number of frames for each nominal run is 2,760, for a total of 55,198 recorded frames.

- **Recovery runs** (130 recordings): the driver starts from a poor position (e.g., too close to parked cars) and steers back toward the center of the road. The average number of frames for each recovery run is 180.5, for a total of 23,682 recorded frames.

Recovery runs are necessary for driving model training since nominal recordings, where the human driver stays in the center of the road, do not provide information on how to recover from a bad position. Without recovery data, the model may be unable to correct its trajectory during inference when it deviates even slightly from the center, resulting in a failure. It is important that recovery runs start from the moment the driver begins to recover, not before, to avoid biasing the model toward drifting off the road. Recovery runs were taken at regular intervals along the whole length of the street.

(a) Nominal sunny (b) Recovery cloudy

Figure 4.3: On the left a nominal frame in sunny weather, on the right recovery frame on cloudy weather

The plot in Figure 4.4 shows the distributions of steering values across all the recorded training data.

The plot in Figure 4.5 shows the distributions of vehicle speed at which we collected our training dataset across all the recorded training data.

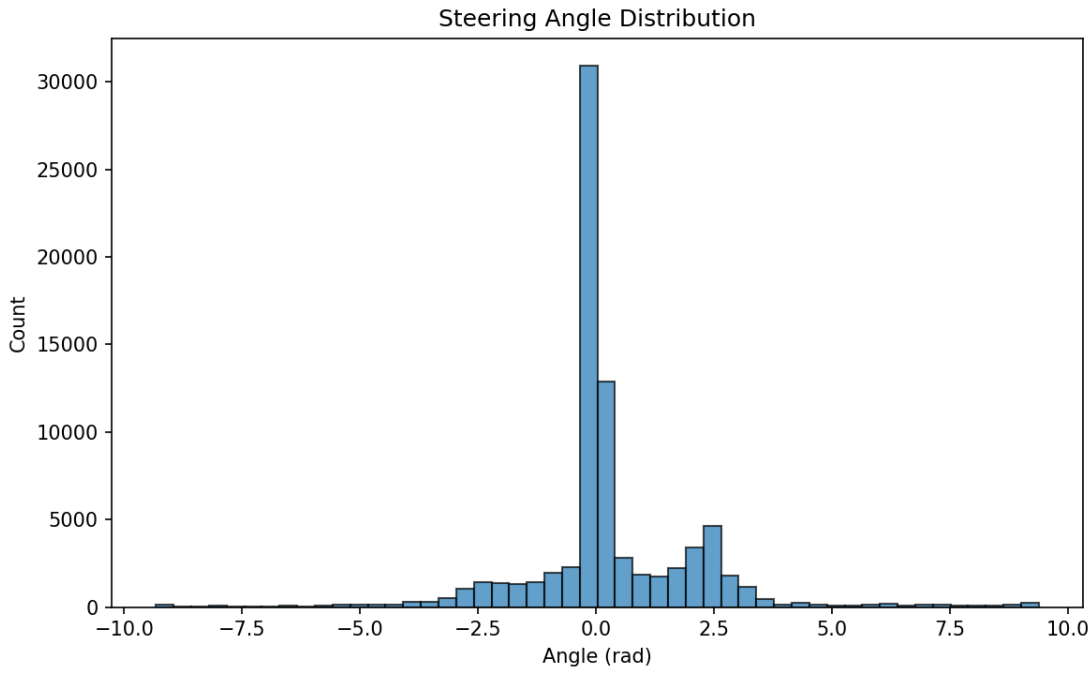


Figure 4.4: Steering angle distribution in the training dataset

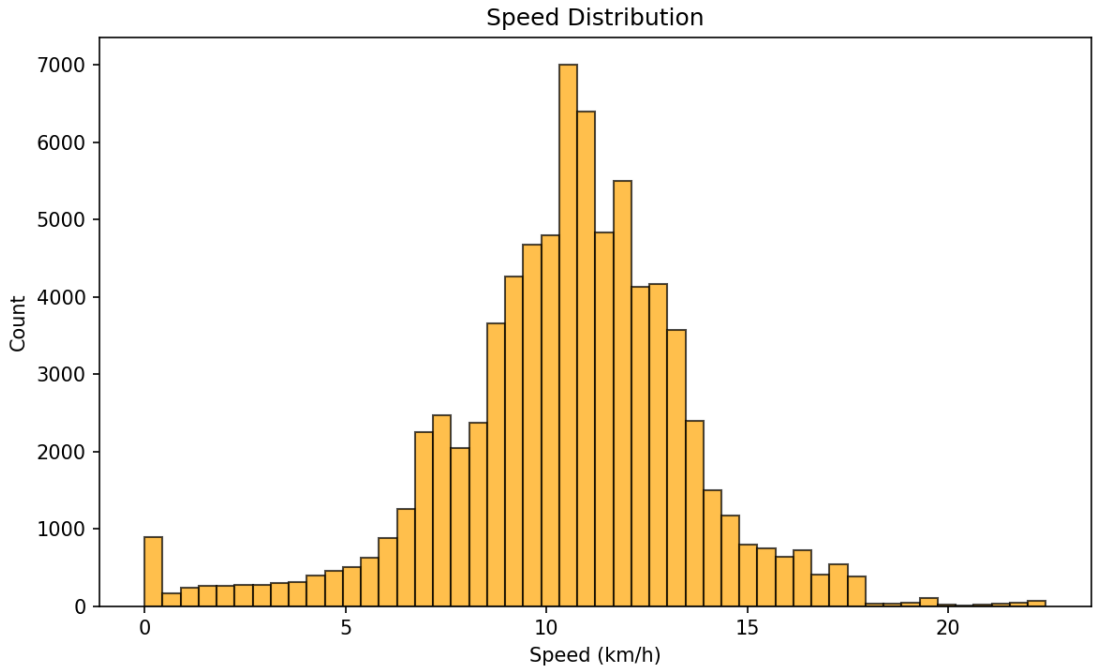


Figure 4.5: Speed distribution in the training dataset

The final training dataset consists in 78880 frame recorded for a total amount of 43.8 minutes of driving recordings.

4.4.2 Testing Dataset

Once the driving model was trained and validated, we deployed it on the Fortuna vehicle to collect our testing dataset. In testing dataset recordings the driving model controls the vehicle steering autonomously predicting the driving commands from camera images in real time. We collected the testing dataset in the driving scenario under three different weather conditions: sunny, cloudy, and snowy. Sunny and cloudy conditions are within the training data distribution, while the snowy condition, with snow covering the ground, represents an out-of-distribution case, as no snow data was present during training.

(a) Sunny (b) Cloudy (c) Snowy

Figure 4.6: Example frames from each testing condition.

Figure 4.6 shows example frames from each condition. For each weather condition, we performed three consecutive runs of the model driving along the scenario. That was necessary to account for the non-deterministic nature of the driving model and to obtain a distribution of plausible trajectories rather than relying on a single execution for our baseline. Under sunny and cloudy conditions, the driving model successfully completed the entire length of the scenario in all runs. Under snowy conditions, the model was unable to stay on the road, requiring manual braking intervention by the human driver to prevent the vehicle from going out of the road. We consider a run as failed when such intervention is needed. Figure 4.7 shows an example of the trajectory of the driving model in a testing recording in our driving scenario.

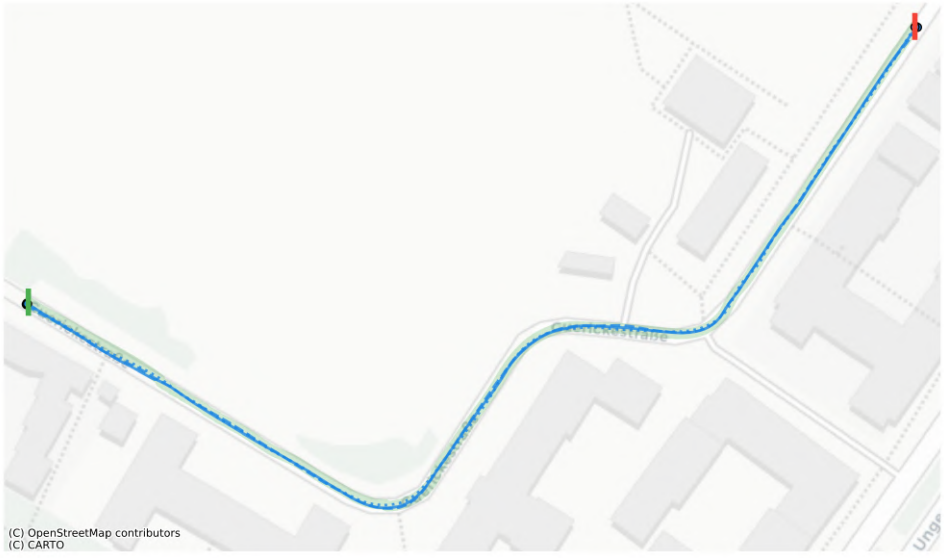


Figure 4.7: Trajectories recorded in the sunny testing condition.

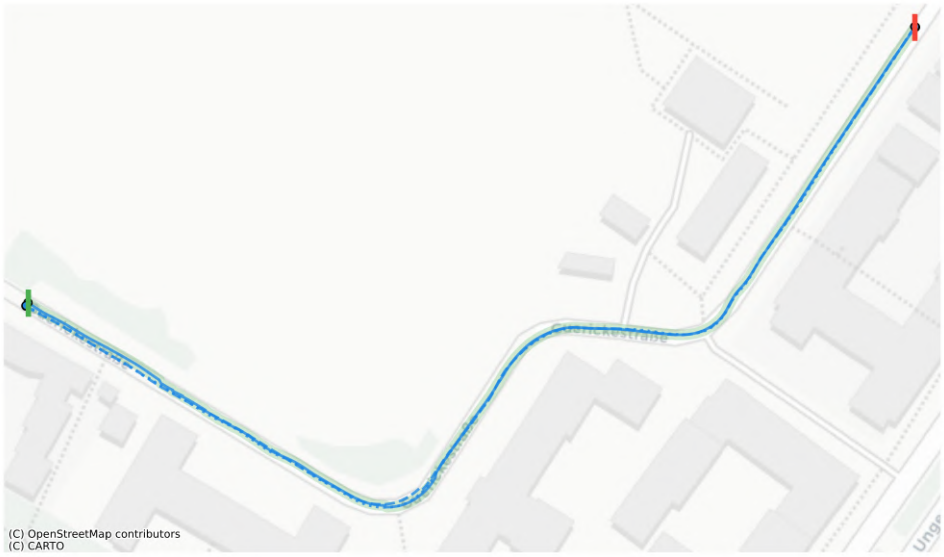


Figure 4.8: Trajectories recorded for the cloudy testing condition.

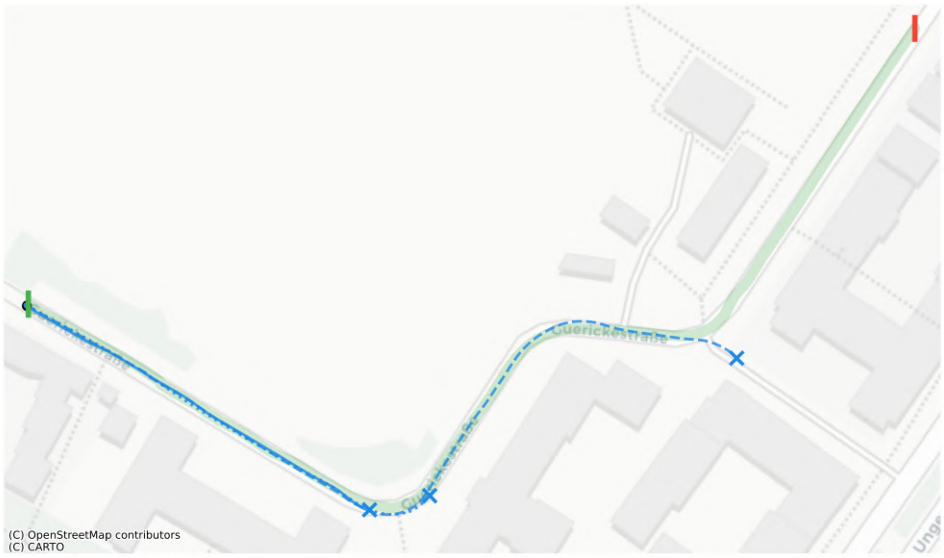


Figure 4.9: Trajectories recorded for the snowy testing condition. Crosses indicate failure points.

The testing dataset consists of 9 recordings (3 conditions × 3 runs), we take multiple runs for every scenario to asses a consistent behavior of the driving model that we can use as baseline for our evaluation. For each recording,

we saved front-facing camera images, LiDAR scans, steering commands predicted by the model, and trajectory coordinates. Additionally, we recorded an extra run under cloudy conditions in another day, which will be used exclusively for validation during parameter selection of the neural rendering models. This recording was not used in the final evaluation to avoid data leakage between configuration selection and testing. The final size of the two datasets exceeds 300,GB.

4.5 System Under Test

For our evaluation, we use an end-to-end camera-based driving model inspired by the DAVE-2 architecture proposed by NVIDIA [21]. The model takes camera images (66×200×3) as input and directly outputs steering commands.

4.5.1 Model Architecture

The original DAVE-2 is a CNN-based model consisting of five convolutional layers followed by three fully connected layers. In our implementation, we added Batch Normalization layers for training stability and Dropout layers for generalization and inference time uncertainty estimation.

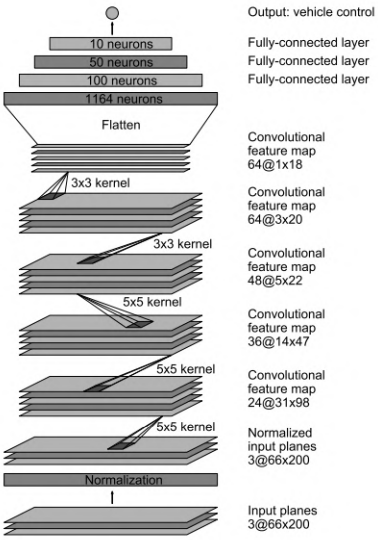


Figure 4.10: NVIDIA DAVE2 Architecture

4.5.2 Training

Pre Processing

Training images were preprocessed according to what the model expects. 204 pixels were removed from the top of the image to exclude sky and buildings, since we want the model to learn to drive based on what it sees on the street. Additionally, 35 pixels were removed from the bottom to exclude the vehicle front hood. This results in an image of 800×264 pixels, which has the same aspect ratio as 200×66, the input dimensions expected by the model, ensuring correct proportions during resizing. Pixel values are normalized to the range $[0, 1]$.

Data augmentation

The only data augmentation technique applied was horizontal flipping of the image along the vertical axis, with the inversion of steering angles. This is a common technique used in ADS training to double the dataset size and avoid bias in turning directions. In our case, since the road has two left turns and one right turn, without image flipping the model would be biased towards one direction. Figure 4.11 and Figure 4.12 show the steering angle distribution before and after data augmentation.

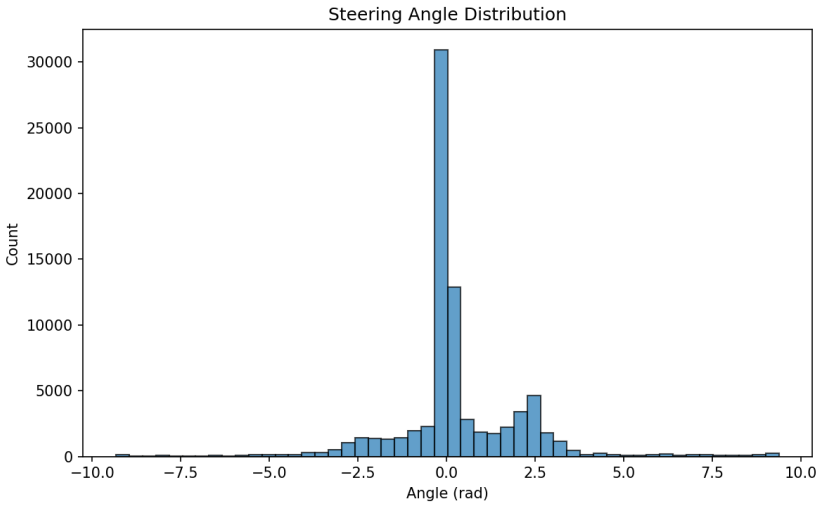


Figure 4.11: Distribution of steering values before augmentation.

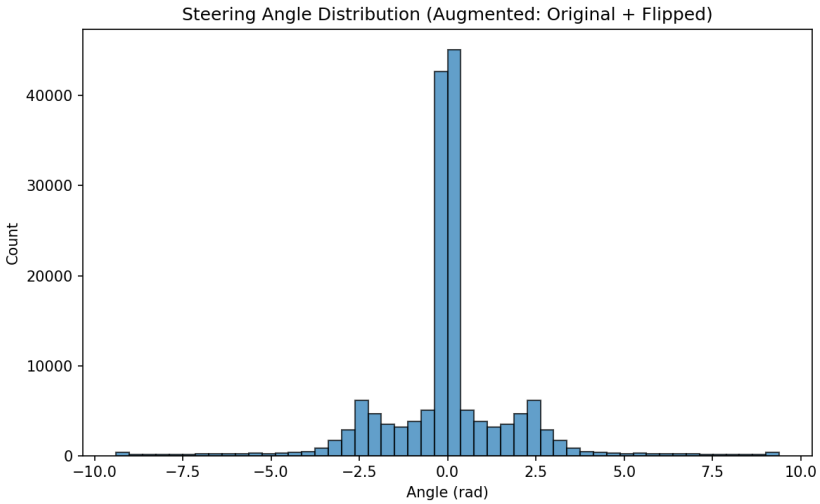


Figure 4.12: Distribution of steering values after augmentation.

Training Configuration

The resulting dataset is randomly split into 80% training and 20% validation. The frame's ids used in each split are saved in JSON files that contains all the frames separated for validation and trainig to ensure reproducibility.

The model is trained using the Adam optimizer with a learning rate of 10^{-4} and Mean Squared Error (MSE) loss, defined as:

$$\mathcal{L}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{\theta}_i - \theta_i)^2 \tag{4.1}$$

where N is the number of samples, $\hat{\theta}_i$ is the predicted steering angle, and θ_i is the ground-truth steering angle.

Training is performed with a batch size of 16 and an epoch limit of 100. Early stopping with a patience of 10 epochs is applied based on validation loss, for stopping training if no improvements are observed. A fixed seed is used to ensure reproducibility. Figure 4.13 shows training loss evolution over the 25 training epochs.

4.5.3 Model-Level Validation

Before deploying the model in the real world to collect the testing dataset, we performed model-level testing on recorded bags. This was done to verify

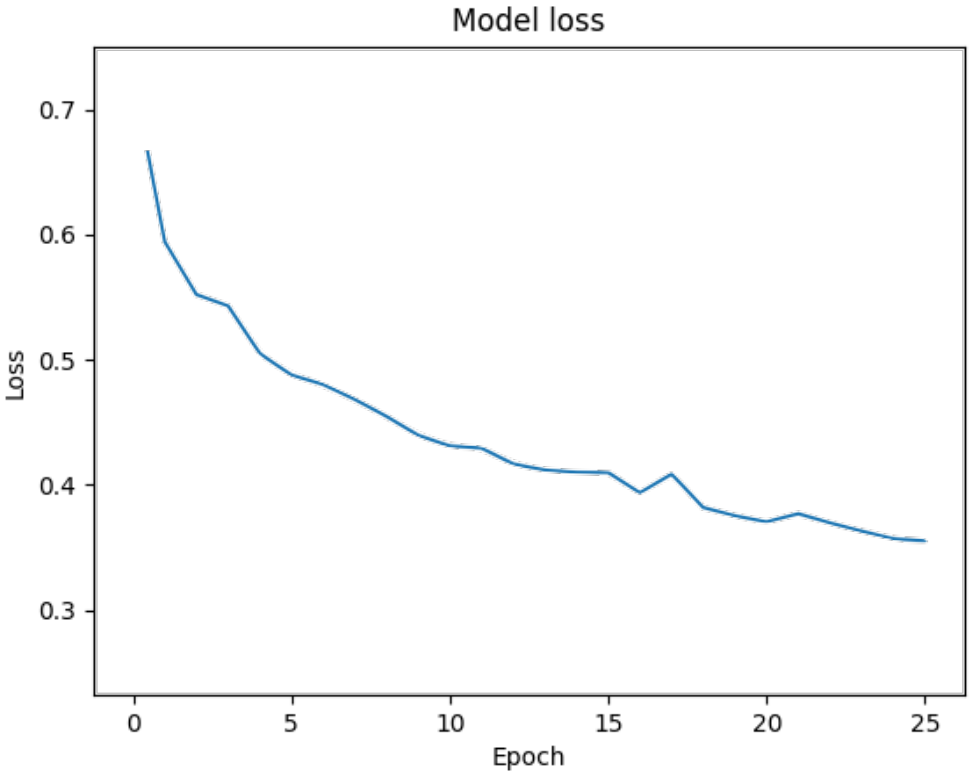


Figure 4.13: Model's loss over epochs

that the model produces coherent outputs before testing it in the real-world driving scenario.

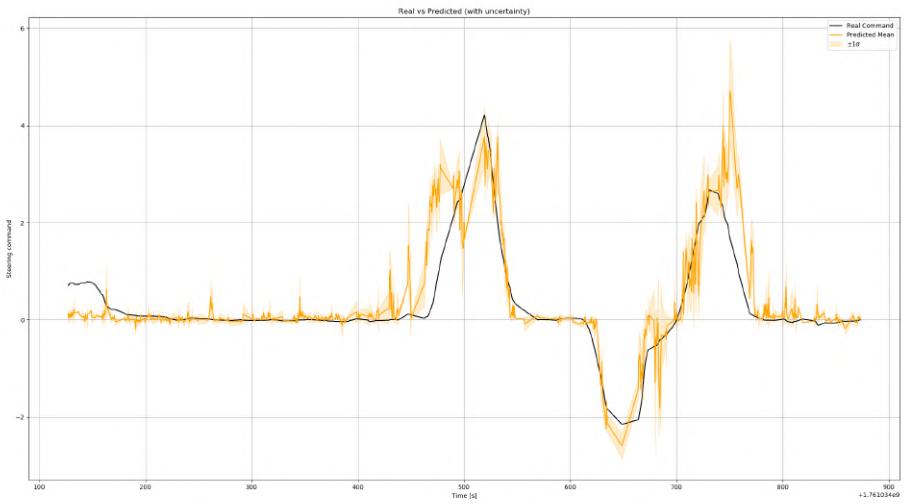


Figure 4.14: Model's steering output vs Human driver ground truth steering values with MonteCarlo Dropout

During open-loop testing the model predicts steering angles which are compared against the ground-truth steering values recorded by the human driver. We employed Monte Carlo Dropout during inference, performing N=5 stochastic forward passes with dropout active. The Figure 4.14 shows in yellow the mean prediction used as steering output, and variance of steering values for estimate the model's prediction uncertainty, and in blue the ground truth.

4.6 RQ1: Scenario Reconstruction Fidelity

How accurate is the parked vehicle detection component of our framework at estimating parked vehicles coordinates in real-world driving scenarios? Does the framework successfully reconstruct real-world recorded driving scenarios in simulation? Since the reconstructed scenario will serve as the basis for all our experimentation, it is essential to assess that it faithfully matches its real-world counterpart.

We evaluate reconstruction fidelity at two levels: parked vehicles detection accuracy, and semantic similarity between simulated and real-world scenes measured through segmentation map comparison.

4.6.1 Metrics

Parked Vehicles Detection Accuracy.

We evaluate the performance of both LiDAR-based and camera-based detection using precision, recall, F1-score, position error and orientation error.

Precision Measures how many of the predicted vehicles corresponds to real vehicles:

$$P = \frac{TP}{TP + FP} \tag{4.2}$$

where TP is the number of true positives and FP is the number of false positives. For our task, a value close to 1 indicates that most predicted detections correspond to actual vehicles. **Recall.**

Recall measures how many of the real vehicles are predicted:

$$R = \frac{TP}{TP + FN} \tag{4.3}$$

where FN is the number of false negatives. For our task, a value of 1 indicates that all real vehicles in the scenario were successfully detected.

F1-Score.

The F1-score is the harmonic mean of precision and recall:

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{4.4}$$

It is important to consider this metric because precision and recall alone do not provide a balanced view of detection performance. For example, a detector could achieve very high recall by predicting many objects as vehicles, while obtaining low precision due to a large number of false positives. Conversely, it could achieve perfect precision by making very few predictions, at the cost of a low recall. The F1-score addresses this trade-off by combining both metrics into a single measure, giving a more balanced assessment of detection quality.

Position Error.

We compute statistics on the Euclidean distance between detected and ground-truth vehicle positions:

$$e_{pos,i} = \|\hat{P}_i - P_i\| \tag{4.5}$$

where $\hat{P}_i$ is the detected position and P_i is the ground-truth position. To provide a comprehensive view of detection performance, we report the

mean, standard deviation, and maximum position error across all matched detections:

$$E_{pos}^{mean} = \frac{1}{N}\sum_{i=1}^{N} e_{pos,i} \tag{4.6}$$

$$E_{pos}^{std} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(e_{pos,i} - E_{pos}^{mean})^2} \tag{4.7}$$

$$E_{pos}^{max} = \max_i \; e_{pos,i} \tag{4.8}$$

Orientation Error.

We compute statistics on the absolute angular difference between detected and ground-truth yaw angles to assess how accurately the detection component estimates bounding boxes orientation:

$$e_{yaw,i} = |\hat{\theta}_i - \theta_i| \tag{4.9}$$

where $\hat{\theta}_i$ is the detected yaw and θ_i is the ground-truth yaw. To provide a comprehensive view of orientation estimation performance, we report the mean, standard deviation, median, and maximum orientation error across all matched detections:

$$E_{yaw}^{mean} = \frac{1}{N}\sum_{i=1}^{N} e_{yaw,i} \tag{4.10}$$

$$E_{yaw}^{std} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(e_{yaw,i} - E_{yaw}^{mean})^2} \tag{4.11}$$

$$E_{yaw}^{max} = \max_i \; e_{yaw,i} \tag{4.12}$$

Semantic Similarity

We assess simulated scenario similarity using Intersection over Union (IoU) on semantic segmentation maps extracted from real-world images and their corresponding simulated images at the same positions. The segmentation maps are obtained using a SegFormer-B0 model [91] pretrained on Cityscapes [38]. This metric simultaneously evaluates trajectory alignment and sensor consistency: by comparing segmentation maps captured at identical coordinates

in both the real world and simulation, we verify that the ego-vehicle follows the correct trajectory and that the simulated sensors perceive the same scene structure as the real sensors. We compute IoU for each semantic class (background, vehicle, road) and report the mean IoU (mIoU) across all classes:

$$IoU_c = \frac{|GT_c \cap Pred_c|}{|GT_c \cup Pred_c|} \tag{4.13}$$

where GT_c and $Pred_c$ are the sets of pixels belonging to class c in the ground-truth and predicted segmentation maps, respectively.

$$mIoU = \frac{1}{C} \sum_{c=1}^{C} IoU_c \tag{4.14}$$

where C is the number of semantic classes.

A value close to 1 indicates that the simulated scenario closely matches its real-world counterpart in terms of semantic scene structure.

4.6.2 Procedure

Ground Truth Annotation

Ground truth 3D bounding boxes were obtained by manually annotating point cloud of the road given by LiDAR scans. Ground-truth bounding boxes are used as the reference for evaluating both the LiDAR-based and camera-based approaches in the task of detecting parked vehicles.

Detection Matching

Since vehicle positions are predicted as continuous float coordinates, an exact correspondence with the ground-truth annotations is generally not expected. We need then to associate detections with ground-truth annotations, to do so we employed a greedy algorithm based on Euclidean distance. First, we compute all pairwise distances between detections and ground-truth vehicles and filter out pairs exceeding a maximum distance threshold of 4 meters. The remaining pairs are sorted by distance in ascending order. Starting from the shortest distance, we assign each detection to its corresponding ground-truth vehicle. Once a match is made, both the detection and the ground-truth vehicle are removed from further consideration, preventing any other detection from being assigned to the same ground-truth and vice versa. This greedy approach ensures that the closest detection-GT pairs are

Figure 4.15: Example of predicted vehicle positions and ground-truth annotations overlaid on an OSM map. Green dots indicate correctly matched vehicle detections, while red dots indicate detections that could not be associated with any ground-truth vehicle and are therefore counted as false positives. Yellow squares represent ground-truth vehicle positions; those highlighted with a red circle indicate false negatives.

matched first, preventing suboptimal assignments where a distant detection could claim a ground-truth vehicle that is closer to another detection. Figure 4.15 shows an example of the vehicle detection results, including true positives, false positives, and false negatives. Position and orientation errors are computed only for true positive detections.

Semantic Map Comparison

To measure how closely the scenario reconstructed in simulation matches its real-world counterpart, we compare their semantic scene structure for the classes considered in this work, namely road and vehicles. After the

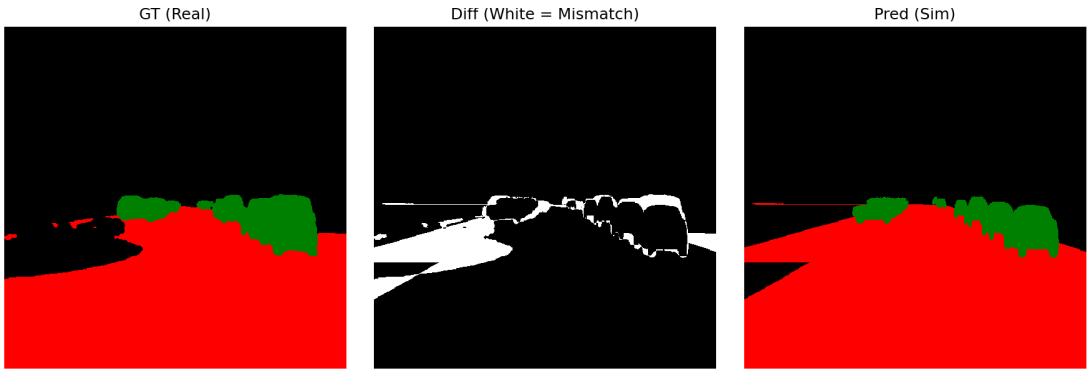


Figure 4.16: Comparison between real-world and simulated segmentation maps. Left: real-world segmentation map. Center: pixel-wise difference, where black pixels indicate matching classes and white pixels indicate mismatches. Right: simulated segmentation map from CARLA.

detection step, we load the scenario in simulation and use the manually annotated ground-truth vehicle positions to place the parked vehicles in the reconstructed environment. We then perform trajectory replay in the CARLA scenario, as described in Section 3.6.1, and capture semantic segmentation maps at each recorded position. These are compared against segmentation maps extracted from the corresponding real-world frames using SegFormer, as shown in Figure 4.16.

4.7 RQ2: Generated Image Quality

Do Stable Diffusion, NeRF, and 3DGS produce images that resemble real-world ones? Which of these techniques generates the most similar images compared to their real-world counterparts? To answer these research questions, we compare the images generated by each method with their corresponding real-world images and study how differences in image quality may influence the behavior of driving models when tested at the system level.

We first fine-tune each method to identify its best parameter setting, so that all three are evaluated under their best-performing configuration. Then, using the validation dataset defined in Section 4.4.2, we perform trajectory replay as described in Section 3.6.1 and generate images with each method. These generated images are then compared with their real-world counterparts.

For the comparison, we use image quality metrics to measure how close

the generated images are to the real-world ones. We also use vehicle metrics to check whether vehicles are represented consistently in both generated and real images. Finally, we include temporal metrics to assess the consistency of the generated images across consecutive frames.

4.7.1 Metrics

Image quality metrics

Image quality metrics are numerical measures used to assess how realistic AI-generated images are by comparing them with corresponding reference images. In our case, however, the goal is not only to measure how similar the generated images are to the reference ones. We also want to understand whether images that obtain good image quality scores lead to driving behavior of the SUT similar to the one observed when the SUT is tested in real world. This is important because image quality metrics do not always capture the aspects of realism that matter in driving scenarios, as shown in [94].

Image quality metrics can be divided into two main categories: single-image metrics and distribution-level metrics. Single-image metrics evaluate each generated image by directly comparing it with its corresponding real-world counterpart. Distribution-level metrics, instead, compare the overall distribution of generated images with the distribution of real images in a reference dataset. These metrics are useful for assessing whether a set of generated images has a similar apparence compared to real-world ones and for evaluating images when there is no specific counterpart.

Single Image Metrics Single image metrics measure the similarity between paired ground truth and generated images. Each metric is computed per image pair and averaged across the dataset. In this work we consider five different single image metrics.

Mean Squared Error (MSE)[36] computes the average squared difference between pixel values of the ground truth and generated images:

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2 \tag{4.15}$$

where x_i and y_i are pixel values of the ground truth and generated image respectively, and N is the total number of pixels across all channels. Lower

MSE indicates higher fidelity. It measures the average pixel-level error between a generated image and its ground-truth counterpart. Therefore, it shows how different the generated image is from the ground-truth image at the pixel level.

Peak Signal-to-Noise Ratio (PSNR)[95] measures the ratio between the maximum possible signal and the noise:

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{MAX^2}{MSE} \right) \tag{4.16}$$

where MAX is the maximum possible pixel value (255 for 8-bit images) and MSE is the mean squared error between pixel values. Higher PSNR indicates better quality. It captures how much noise is present in the generated image with respect to the ground-truth image.

Structural Similarity Index (SSIM) [96] measures perceived image quality based on luminance, contrast, and structure:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{4.17}$$

where μ_x, μ_y are the mean values, σ_x^2, σ_y^2 are the variances, σ_{xy} is the covariance, and c_1, c_2 are constants for numerical stability. SSIM ranges from -1 to 1, where 1 indicates identical images. Evaluates similarity in terms of luminance, contrast, and structural information, which makes it more aligned with human visual perception than pixel-wise error measures such as MSE or PSNR.

Classifier Perceptual Loss (CPL)[97] measures high-level perceptual similarity by comparing feature maps extracted from a pre-trained classification model. Both the generated image and the reference image are passed through the model, and the MSE between the corresponding feature representations is computed:

$$\text{CPL} = \frac{1}{M} \sum_{j=1}^{M} (f_j^{\text{gt}} - f_j^{\text{gen}})^2 \tag{4.18}$$

where f^{gt} and f^{gen} are the feature representations of the ground-truth and generated images, respectively, and M is the feature dimensionality. It compares images in a learned feature space rather than at the pixel level. Therefore, it captures higher-level perceptual information such as shapes, textures, and objects, making it more suitable for evaluating whether a generated image preserves visually meaningful structures from the reference image. Lower CPL indicates more perceptually similar images.

Segmentation Score (SegScore)[94] evaluates semantic consistency between ground truth and generated images. Both images are independently segmented using SegFormer-B0[91] a into Cityscapes[38]class maps (19 classes including road, vehicle, building, etc.), and the MSE between the predicted class ID maps is computed:

$$\text{SegScore} = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} (c_{ij}^{\text{gt}} - c_{ij}^{\text{gen}})^2 \tag{4.19}$$

where c_{ij}^{gt} and c_{ij}^{gen} are the predicted semantic class IDs at pixel (i, j) for the ground truth and generated image respectively. It is useful to asses if the semantic scene is preserved in generated images with lower values indicates closer matching with ground truth.

Distribution Level Metrics **Fréchet Inception Distance (FID)** [98] measures the distance between the feature distributions of real and generated images:

$$\text{FID} = \|\mu_{\text{real}} - \mu_{\text{gen}}\|^2 + \text{Tr}\left(\Sigma_{\text{real}} + \Sigma_{\text{gen}} - 2(\Sigma_{\text{real}}\Sigma_{\text{gen}})^{1/2}\right) \tag{4.20}$$

where μ_{real} and μ_{gen} are the mean feature vectors, and Σ_{real} and Σ_{gen} are the covariance matrices of features extracted from real and generated images using a pre-trained Inception network. These features are modeled as multivariate Gaussian distributions, and their similarity is measured through the Fréchet distance [99]. Lower FID should indicate better matching between real and generated images.

Kernel Inception Distance (KID) [100] estimates the squared Maximum Mean Discrepancy between the Inception features of real and generated images using a polynomial kernel:

$$\text{KID} = \mathbb{E}[k(x, x')] + \mathbb{E}[k(y, y')] - 2\mathbb{E}[k(x, y)] \tag{4.21}$$

where $k(a, b) = (\gamma a^\top b + 1)^3$ is a cubic polynomial kernel, and x and y are Inception feature vectors extracted from real and generated images, respectively. Unlike FID, KID does not assume that feature distributions are Gaussian and provides an unbiased estimator of distribution similarity. Lower KID values indicate that the generated images are more closely aligned with the real image distribution.

Inception Score (IS) [95] employs a pretrained Inception network‘[101] as a feature extractor to assess both the quality and diversity of generated

image distributions. It measures the KL divergence between the conditional class distribution $p(y|\mathbf{x})$ and the marginal distribution $p(y)$:

$$\text{IS} = \exp\left(\mathbb{E}_{\mathbf{x}}\left[D_{\text{KL}}(p(y|\mathbf{x})\|p(y))\right]\right) \tag{4.22}$$

A high IS indicates that generated images are both individually recognizable and diverse across the set.

Maximum Mean Discrepancy with RBF kernel (MMD-RBF) [102] measures distributional distance using a Gaussian radial basis function kernel in Inception feature space:

$$\text{MMD}^2 = \mathbb{E}[k(\mathbf{x}, \mathbf{x}')] + \mathbb{E}[k(\mathbf{y}, \mathbf{y}')] - 2\mathbb{E}[k(\mathbf{x}, \mathbf{y})] \tag{4.23}$$

where $k(\mathbf{a}, \mathbf{b}) = \exp(-\|\mathbf{a} - \mathbf{b}\|^2/2\sigma^2)$ with bandwidth σ set to the median pairwise distance. It compares the overall distribution of generated images with that of real images without assuming a specific parametric form, such as a Gaussian distribution like FID does. Lower MMD-RBF indicates closer alignment between generated and real image distributions.

Precision, Recall, Density, and Coverage (PRDC) [103] provide a detailed analysis of how much the generated and real image distributions overlap in feature space. Unlike single-number scores such as FID or KID, which combine fidelity and diversity into one value, PRDC separates them into four independent values, giving a better understanding on the generation quality. Given real samples $\{X_i\}_{i=1}^N$ and generated samples $\{Y_j\}_{j=1}^M$ embedded in Inception feature space, i.e. intermediate activations extracted from a pretrained Inception-v3 network [101], a manifold is estimated via k-nearest neighbor (k-NN) [105] spheres:

$$\text{manifold}(X_1, \ldots, X_N) := \bigcup_{i=1}^{N} B\Big(X_i, \text{NND}_k(X_i)\Big) \tag{4.24}$$

where $B(x, r)$ is the hypersphere centered at x with radius r, and $\text{NND}_k(X_i)$ is the distance from X_i to its k-th nearest neighbor within $\{X_i\}$. In our implementation, $k = 3$.

Precision measures the fraction of generated samples that fall within the real manifold, telling us whether the generated images look realistic:

$$\text{Precision} := \frac{1}{M}\sum_{j=1}^{M} \mathbb{1}_{Y_j \in \text{manifold}(X_1,\ldots,X_N)} \tag{4.25}$$

A low precision means the generator is producing images that do not look like real ones.

Recall measures the fraction of real samples that fall within the generated manifold, telling us whether the generator covers the full variety of real scenes:

$$\text{Recall} := \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{X_i \in \text{manifold}(Y_1,\ldots,Y_M)} \tag{4.26}$$

A low recall means the generator is missing parts of the real distribution.

Density answers the same question as precision but in a more stable way. Instead of a binary yes/no decision, it counts how many real neighborhood spheres contain each generated sample, making it less sensitive to outliers:

$$\text{Density} := \frac{1}{kM} \sum_{j=1}^{M} \sum_{i=1}^{N} \mathbb{1}_{Y_j \in B(X_i, \text{NND}_k(X_i))} \tag{4.27}$$

Coverage does the same for recall, measuring the fraction of real samples whose nearest generated neighbor falls within their k-NN sphere:

$$\text{Coverage} := \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{\|X_i - \text{NN}_Y(X_i)\| \leq \text{NND}_k(X_i)} \tag{4.28}$$

where $\text{NN}_Y(X_i)$ is the nearest neighbor of X_i in $\{Y_j\}$. Higher values are better for all four metrics. For example, a generation method might produce very realistic images but only for certain scenes (high precision, low recall), while another might cover all conditions but with lower quality (low precision, high recall). PRDC makes these differences visible.

Vehicle Consistency Metrics

Vehicle consistency metrics evaluate whether vehicles detected in the ground truth image are also detected in the corresponding generated image. Both images are processed with YOLOv8n [106], pretrained on the COCO dataset [107], which contains 80 object classes including vehicle categories relevant to our scenario: cars, trucks, buses. The model takes each image as input and outputs a set of bounding boxes, each with a class label and a confidence score. We filter out detections with a bounding box area below 600 pixels to discard small, unreliable detections. The remaining bounding boxes from ground truth images and generated images are then matched using greedy IoU matching with a threshold of 0.5: for each ground truth vehicle, the generated detection with the highest IoU is selected as its match, provided the overlap exceeds the threshold. IoU represent the intersection area of the predicted bound boxes.

Vehicle Recall measures the fraction of ground truth vehicles that were successfully matched in the generated image:

$$\text{Recall} = \frac{|\text{matched}|}{|\text{GT vehicles}|} \tag{4.29}$$

It tells us how many of the vehicles in the real scene are also present in the generated image.

Vehicle Precision measures the fraction of detected veicles present in the generated images that also appearse in the ground truth.

$$\text{Precision} = \frac{|\text{matched}|}{|\text{Gen vehicles}|} \tag{4.30}$$

It is useful cause we want to know if the model predicted actually consitent values.

Vehicle Average IoU is the mean Intersection over Union of all matched vehicle pairs:

$$\text{Avg IoU} = \frac{1}{|\text{matched}|} \sum_{k=1}^{|\text{matched}|} \text{IoU}(b_k^{\text{gt}}, b_k^{\text{gen}}) \tag{4.31}$$

It tells us how well the position of vehicles in the generated image corresponds to their position in the ground truth. Higher values for all three metrics indicate better preservation of vehicle appearance and positioning.

Temporal Consistency Metrics

We additionally employ single image metrics such as SSIM, PSNR, MSE, and CPL for assessing temporal consistency. This is important because we want each newly generated frame to be visually coherent with the previous one, avoiding visual inconsistencies between consecutive frames. We compute these metrics on each pair of consecutive generated frames (I_t, I_{t+1}), using the same formulations as the single-image metrics but applied between adjacent generated frames rather than between generated and ground truth images.

4.7.2 Neural Rendering Techniques Parameter Selection

Each neural rendering technique under evaluation presents different parameters that can affect the quality of the generated images. Using different

parameter configurations can lead to images that match the real world more or less accurately. Before evaluating the ability of each technique to translate simulated data into data that resembles the real world, it is therefore necessary to identify, for each technique, the parameter configuration that achieves the most promising quality on generated images. We achieve this by performing a study of different parameter configurations for each technique. This study is conducted on a subset of the validation set described in Section 4.4.2.

Stable Diffusion Parameter Selection

Stable Diffusion is highly sensitive to inference parameters. Incorrect control guidance values can produce images with severe artifacts or unrealistic appearance, making parameter selection essential before evaluation. In this study we have considered the following parameters: *i) Control guidance*, consisting of a start and end value for each conditioning input (semantic segmentation, instance segmentation, and temporal conditioning), which determine the portion of the denoising diffusion process each ControlNet can influence. *ii) Guess mode* [71], which can either be enabled or disabled. When enabled, ControlNet relies more on the structural conditioning inputs, so in our case semantic segmentation maps, instance maps, and the previously generated frame, rather than on the text prompt to guide the generation. *iii) Seed*, which determines the initial noise pattern for the generation process. It can be set to a fixed value for reproducibility or randomized across frames.

Due to the large parameter space, we adopted a multi-stage selection process. In the first stage, a preliminary exploration is performed prior to the research question evaluation to identify the subset of the parameter space that produces acceptable visual quality. In the second stage, configurations near the best candidates identified in the first stage are selected and their results are used in the evaluation of the second research question.

Preliminary Parameter Exploration

In the preliminary exploration, since it is infeasible to evaluate all possible combinations of control guidance values, we discretize the parameter space by sampling start and end values at 0.00, 0.33, 0.66, 1.00 of the denoising process. This yields 6 valid control guidance combinations per ControlNet channel (where end $>$ start), resulting in $6^3 = 216$ total combinations across three channels. Imposing the constraint that at least one ControlNet must be active at every timestep of the denoising process reduces this to 163 valid

configurations. To efficiently explore this space, we apply farthest-point sampling [108]: the first configuration is chosen randomly, and each subsequent one is selected as the point that maximizes the minimum Euclidean distance to all previously selected points in the 6-dimensional parameter space (3 start values + 3 end values). This greedy strategy ensures that each new sample is as far as possible from the entire set of already selected configurations, providing diverse and uniform coverage of the valid parameter space. Using this method, we selected 25 representative configurations. Each sampled configuration is evaluated with fixed or random seed and guess mode set to either enabled or disabled, resulting in a total of 100 configurations. For each configuration, 10 frames are generated using the Stable Diffusion and ControlNet models trained on the validation run. The number of generated frames is intentionally limited to reduce computational cost, as the purpose of this stage is only to determine whether a configuration produces visually plausible images and preserves the structural information provided by the ControlNet inputs. In practice, a small number of frames is sufficient to detect configurations that generate clear artifacts or fail to maintain the expected scene layout.

All 100 generated sequences are then evaluated through manual inspection. Three human evaluators are asked to answer a binary question for each sequence: *does the generated output represent an acceptable representation of the scene?* The evaluation focuses only on whether the output is structurally correct and free from major artifacts, rather than on subtle perceptual differences. This step allows us to discard configurations that clearly fail to produce valid outputs. The configurations are also ranked using the image quality metrics described in Section 4.7.1. These metrics provide an automatic measure of similarity between generated images and their real-world counterparts. However, following the observation by Lambertenghi et al. [94] that image quality metrics do not always correlate with correct driving behavior or perceptual validity, metric scores alone are not considered sufficient to select the best configurations. The final selection is therefore performed by combining both criteria. Among the configurations that are deemed acceptable by human evaluators, we select the three configurations with the best ranking according to the image quality metrics. These top three configurations define the promising regions of the parameter space that will be further analyzed in the subsequent stage of the evaluation.

Focused Parameters Exploration

Once the preliminary parameter exploration is completed, its results are used to guide a second, more detailed exploration of the parameter space. In particular, we analyze the three most promising configurations identified during the preliminary exploration in order to extract general rules that define the focused exploration stage. The analysis focuses on four aspects. First, we determine whether I) the random seed should be fixed or randomized, and II) whether guess mode should be enabled or disabled. Once these global parameters are determined, we analyze the ControlNet activation patterns to derive rules for the scheduling of each model during the denoising process. For each ControlNet, we extract III-IV-V) the start and end region of the denoising process in which the ControlNet should be active, and the fraction of the denoising process during which the ControlNet should remain active. Using these rules, we define a reduced parameter space centered around the promising configurations identified during the preliminary exploration. In this stage, the configuration of the random seed and guess mode parameters is fixed according to the settings observed in the three best configurations of the preliminary exploration. Then the discretization of the ControlNet activation parameters is refined. While the preliminary exploration discretized the start and end activation values at $\{0.00, 0.33, 0.66, 1.00\}$ of the denoising process, the focused exploration allows these parameters to vary at each percentage point of the denoising timeline, enabling a finer search within the relevant regions of the parameter space.

From this refined parameter space, we randomly sample 50 configurations. For each configuration, we generate 384 frames corresponding to 1/10 of the validation scenario and evaluate the generated images using the same image quality metrics described in Section 4.7.1. These metrics allow us to compare the generated images against their real-world counterparts and obtain a ranking of the configurations. To obtain a single score for each configuration representative of all the metrics, we need to aggregate them. However, we could not employ a direct average because different metrics use different scales and units, which would result in metrics with larger magnitudes dominating the final score, regardless of their actual importance.

To address this, we normalize each metric using the z-score, a standard statistical transformation defined as:

$$z = \frac{x - \mu}{\sigma} \tag{4.32}$$

where x is the metric value for a given configuration, μ is the mean of that

metric across all configurations, and σ is its standard deviation. After the normalization, each metric is expressed on a common scale: a z-score of $+1$ indicates that the configuration is one standard deviation above the average for that metric, while a z-score of -1 indicates one standard deviation below. This allows metrics with different units and ranges to be compared on equal terms.

Since some metrics are better when higher (e.g., PSNR, SSIM) and others are better when lower (e.g., MSE, FID, CPL), we negate the z-scores of lower-is-better metrics so that a higher value always indicates better performance. The overall score for each configuration is then computed as the mean of all its normalized metric values. Configurations are ranked by this overall score, with higher values indicating better image generation quality.

In this phase of the selection we did not use distribution-level metrics for assessing the quality of generated outputs. This choice was made following the findings of Chong and Forsyth [109] and Jayasumana et al. [110], who show that distribution-level metrics such as FID and KID produce unreliable estimates when computed on small sample sizes, as they require large image sets to capture meaningful statistical properties. Since we generates only 384 frames per configuration, we considered the sample size as insufficient to produce meaningful distribution-level comparisons.

Based on this metric-based ranking, we select the top 15 configurations for further evaluation. A human evaluation study is then conducted in order to assess the overall visual quality and structural plausibility of the generated outputs. For each configuration, the evaluators inspect videos of the 384 generated frames played at 30 fps, resulting in approximately 13 seconds of video per configuration. For the human study, we asked three independent evaluators to rank the generated videos of the top 15 configurations according to their preferences and perceived quality. The ground truth video from the validation set was also included among the configurations to verify that the evaluators were providing reliable and attentive judgments.

We decided to compare videos instead of single images to assess both the quality of individual generated frames and the temporal consistency among subsequent frames. We adopted a side-by-side layout to enable direct comparison between configurations, rather than asking evaluators to remember previously watched videos. This is important in this stage of the evaluation, where differences between configurations are smaller and a direct comparison provides a more reliable way to distinguish them. The evaluators were presented with two videos from different parameter configurations simultaneously, displayed side by side, as shown in Figure 4.17, and asked to answer

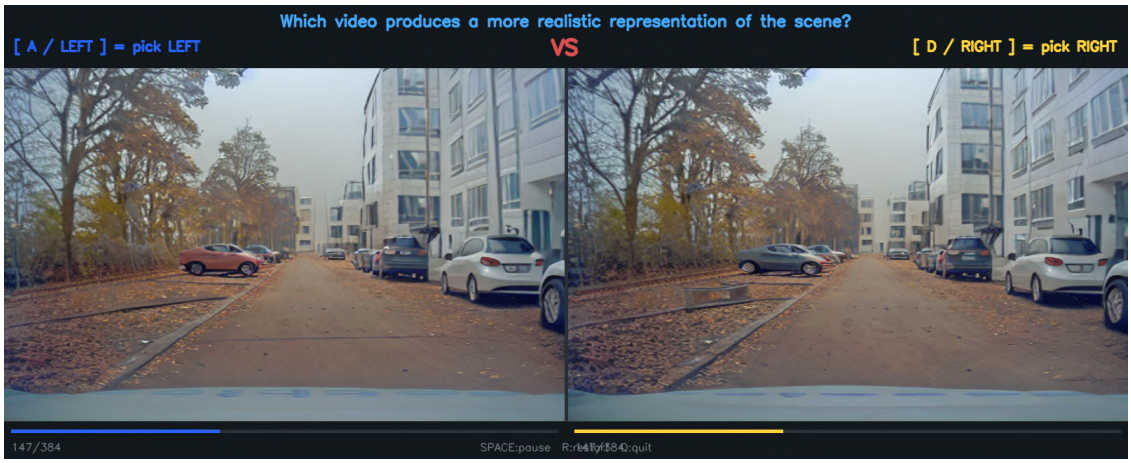


Figure 4.17: Screenshot of the human evaluation interface. Two generated video sequences are displayed side by side, and the evaluator selects the one that produces a more realistic representation of the scene.

the question: *which of the two videos produces a more realistic representation of the scene?* The left-right placement of each pair was randomized to avoid position bias. After the first comparison, which established the first and second positions in the ranking, each subsequent configuration was inserted using a binary search procedure [4]: the new video was compared against the configuration currently in the middle of the ranking, and based on the evaluator's choice, the search continued in the upper or lower half, where the evaluator was asked again to compare against the new middle element, repeating until the correct position was found. This approach requires $O(\log n)$ comparisons per configuration, resulting in a total of 42 pairwise comparisons and an average evaluation time of approximately 15 minutes per evaluator, keeping the evaluation manageable.

The final configuration is selected by the average rank across the three evaluators.

4.7.3 NeRF and 3DGS Parameter Selection

Unlike Stable Diffusion, which requires extensive parameter tuning, NeRF and 3DGS implementations benefit from well-established default training configurations provided by Nerfstudio [82], making the parameter selection significantly simpler. The only parameter we evaluate is the undersampling rate applied to the training frames. Since the driving recordings are captured

at 30 fps, consecutive frames are highly redundant, providing minimal additional viewpoint diversity for scene reconstruction. Training on all frames would significantly increase computational and memory costs without meaningful improvements in reconstruction quality. Furthermore, nearly identical adjacent frames add little geometric baseline, which is a key factor for accurate depth estimation in neural rendering methods [82].

Subsampling the training dataset reduces the level of redundancy while preserving an acceptable level of scene coverage and multi-view overlap. However, an aggressive subsemple rate may compromise view diversity. We therefore evaluate different undersampling rates to identify the configuration that best balances scene coverage, reconstruction quality, and computational efficiency. Specifically, we evaluate three subsampling rates: 1/2, 1/3, and 1/5 of the original frame rate. For each rate, we train a separate model using the training pipeline described in Section 3.5.1 on the validation dataset. We then use the trajectory replay module described in Section 3.6.1 to generate the first 384 frames for each configuration. To select the best undersampling rate, we conduct the same human evaluation study that we did for Stable Diffusion, comparing the three generated videos and the ground truth sequence, for both NeRF and 3DGS, resulting in four configurations in total. The final undersampling rate for each technique is selected as the one ranked highest by the evaluators.

4.7.4 Procedures

We use the training models component of our framework (Section 3.5.1) to train SD, NeRF, and 3DGS on the testing datasets introduced in Section 4.4.2. Specifically, we train each the techniques under the three different weather conditions in which we have taken our testing datasets, this is done to assess image quality across multiple datasets with visually different characteristics. After training, we obtain a total of nine models, one for each combination of technique and weather condition. For 3DGS and NeRF, we use the testing datasets undersampled at the optimal rate identified in the previous section, and for SD during inference we employed the best congifuration found after human evaluation. We then load the driving scenario in CARLA in Section 4.6.2, and use the trajectory replay component of our framework (Section 3.6.1) to replay the real-world trajectory in the simulated scenario, collecting a generated image at each position along the trajectory corresponding to the positions where the real-world images were captured.

At the end of this process, we obtain a set of generated images for each technique and for each weather condition. We then evaluate the quality of these images separately for each weather condition using the metrics presented in Section 4.7.1 and compare the results across the different techniques. Unlike the parameter selection phase, where only 384 frames were generated per configuration, here we generate images for the entire testing dataset, which consist for respectivelly the smaller and biggest to 1734 frames and 3895 frames, providing a sufficient sample size to compute also distribution-level metrics.

4.8 RQ3: System level Behavior

How does the System Under Test behave in system-level testing when fed with raw synthetic images or with generated images compared to real-world baseline behavior in the same driving scenario and in the same weather condition?

In this research question, we aim to evaluate the effectiveness of the different neural rendering techniques in terms of system-level testing of autonomous driving systems. To this end, we provide our system under test with raw synthetic images, images generated by Stable Diffusion, NeRF, and 3DGS, and compare the resulting driving behaviors against real-world baseline in the same driving scenario and weather condition. This evaluation is important as it goes beyond image quality assessment: as shown by Lambertenghi et al. [94], image quality metrics alone do not always correlate with correct driving behavior. By comparing system-level outputs, we can determine which technique most effectively reduces the reality gap in terms of how the system actually behaves, rather than relying on image quality metrics.

4.8.1 Metrics

Task Completion Metrics

Task completion metrics evaluate whether the SUT tested in simulation can successfully match the driving behavior of the real world baseline, in case of failure how much of the scenario is completed before a failure occurs, and what type of failures are encountered. These metrics are essential to understand which techniques produce images of sufficient quality for the SUT to be able to perform similarly to real world, receiving those images as input, providing a measure of the reality gap beyond image quality alone.

Fail Rate reports the number of runs that resulted in a failure out of the total number of attempts. A run is considered failed if the SUT cannot complete the driving scenario requiring human intervention in real world and a car crash or fell out of map in simulation. This metric provides a quick indication of which neural rendering technique best matches the real-world baseline in system-level testing. It is important to note that the absence of failures is not necessarily a positive result, since it may provide a false sense of confidence if the simulation testing does not find a failure occurred in the real world baseline.

Failure types are categorized as:

- **OR** (Off Road): the vehicle leaves the road unexpectedly during a straight segment.
- **CC** (Car Crash): the vehicle collides with another vehicle.
- **OS** (Oversteering): the vehicle steers too much in a curve causing a failure.
- **US** (Understeering): the vehicle does not steer enough in a curve causing a failure.

Failure types are important not only for assessing the criticality of the generated images, but also for evaluating whether the simulated failures are consistent with those observed in the real world. By analyzing when and how the SUT fails, we can identify under which conditions the generated images are sufficient for correct driving, under which conditions they are not, and, when a failure also occurs in the real world, whether the simulated failure is of the same type. This analysis helps reveal which visual properties of the generated images deviate from their real-world counterparts, providing insight into the specific limitations of each generation technique.

Completion Rate measures the percentage of the route completed before failure or successful termination:

$$\text{Completion Rate} = \frac{d_{\text{completed}}}{d_{\text{total}}} \times 100\% \tag{4.33}$$

where $d_{\text{completed}}$ is the distance traveled along the driving scenario and d_{total} is the total driving scenario length. We report the average, maximum, and minimum values across runs. These values are used to assess how closely the simulated behavior matches the real-world baseline in terms of how far

the SUT progresses before failure or successful completion. Large differences with respect to the real-world runs indicate that the corresponding image generation technique does not reproduce the real-world testing conditions faithfully enough for system-level evaluation.

Driving Quality Metrics

Driving quality metrics evaluate how closely the SUT behavior in simulation matches its behavior in the real-world baseline, in terms of both executed trajectory and steering smoothness. While task completion metrics indicate whether the SUT reaches a similar outcome to the one observed in the real world, driving quality metrics assess whether it behaves similarly while doing so. A technique may allow the SUT to complete the scenario, yet still produce trajectories or steering patterns that differ substantially from the real-world runs, revealing a remaining reality gap. We therefore evaluate two aspects of driving quality: trajectory similarity, which measures how closely the simulated trajectories match the real-world ones, and steering smoothness, which measures how closely the steering behavior in simulation resembles that observed in the real world.

Trajectory Similarity. Fréchet Distance [99] measures the similarity between the executed and reference trajectories as ordered sequences of points. We compute the discrete Fréchet distance between the SUT trajectory in simulation with the neural rendering techniques applied and each real-world trajectory, and report the minimum value, i.e., the distance to the closest real-world run. This accounts for the natural variability between real-world runs, avoiding penalizing a generated trajectory that follows one valid real-world path but not another:

$$d_F(P, Q) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \|P(\alpha(t)) - Q(\beta(t))\| \tag{4.34}$$

where P and Q are the executed and reference trajectories respectively, and α, β are monotone reparameterization. Lower values indicate closer trajectory following. This metric provides an overall measure of how closely the SUT replicates the real-world driving path when fed with generated images.

Corridor Violation Rate measures the percentage of trajectory points where the vehicle deviates beyond the lateral corridor defined by the real-world trajectories. The corridor is constructed as the area defined by of all real-world runs: at each point along the driving scenario, corridor boundaries

are set by the minimum and maximum lateral distance observed across the real-world trajectories. A point is considered outside the corridor when it is outside this area:

$$\text{Corridor Viol.} = \frac{1}{N}\sum_{i=1}^{N} \mathbb{1}[d_{\perp}(P_i, C) > 0] \times 100\% \tag{4.35}$$

where $d_{\perp}(P_i, C)$ is the lateral distance from trajectory point P_i to the nearest corridor boundary C, and N is the total number of trajectory points. Lower values indicate the SUT stayed close to real world runs.

Mean Excess is the average lateral deviation beyond the corridor boundary, computed only over the points that violate the corridor:

$$\text{Mean Excess} = \frac{\sum_{i=1}^{N} d_{\perp}(P_i, C) \cdot \mathbb{1}[d_{\perp}(P_i, C) > 0]}{\sum_{i=1}^{N} \mathbb{1}[d_{\perp}(P_i, C) > 0]} \tag{4.36}$$

Lower values indicate that when violations occur, they are minor. Two techniques may have the same corridor violation rate, but one may deviate by a few centimeters while the other deviates by several meters this metric distinguishes between the two.

Excess When Out is the average lateral deviation beyond the corridor boundary, computed over all trajectory points (assigning zero excess to points within the corridor):

$$\text{Excess When Out} = \frac{1}{N}\sum_{i=1}^{N} \max(0, d_{\perp}(P_i, C)) \tag{4.37}$$

This metric combines the frequency and severity of corridor violations into a single value, providing a compact summary of the overall lateral deviation from the real-world driving corridor. Lower is better.

Steering Smoothness. **Average Steering Jitter** measures the standard deviation of the steering rate over time, capturing how erratic the steering behavior is:

$$\text{Steering Vol.} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N-1}\left(\frac{\delta_{i+1} - \delta_i}{\Delta t} - \overline{r}\right)^2} \tag{4.38}$$

where δ_i is the steering angle at time step i, Δt is the time interval between consecutive steps, and $\overline{r}$ is the mean steering rate. We use the standard

deviation rather than the mean absolute rate because it would have penalize steady curves, where the steering rate is consistently high but smooth. The standard deviation instead captures how much the steering rate varies around its mean, distinguishing between stable turning and nervous oscillating behavior. Lower values indicate more stable steering.

Maximum Steering Jitter captures the worst-case steering oscillation:

$$\text{Max Jitter} = \max_{i \in \{1,\ldots,N-1\}} \left| \frac{\delta_{i+1} - \delta_i}{\Delta t} \right| \tag{4.39}$$

Lower values indicate the absence of sudden, large steering corrections. While the steering volatility may be acceptable overall, a single extreme correction could indicate a specific frame where the generated image was particularly unrealistic, causing a dangerous sudden reaction from the SUT.

4.8.2 Procedures

For this research question, we use the models trained in Section 4.7.4 and the system-level simulation component of our framework (Section 3.6.2) to let the SUT drive the vehicle in the simulated driving scenario. We test the SUT at system level under the three different weather conditions of the testing dataset, providing as input images generated by each neural rendering technique. Specifically, we perform three runs for each technique and for each weather condition. We also let the SUT drive with raw synthetic images only, without any neural rendering applied, to empirically demonstrate the existence of the reality gap. Run results are compared against the real-world driving behavior, which serves as the baseline for this evaluation using the metrics described in 4.8.1. For successful runs, we compute the driving quality metrics discarding unsuccessful runs from this evaluation, as incomplete trajectories would introduce bias in the comparison by mixing driving quality with task completion issues.

Chapter 5

Results

5.1 RQ1: Scenario Reconstruction Fidelity

5.1.1 Parked Vehicles Detection Accuracy

We evaluated both camera-based and LiDAR-based detection on the three weather conditions of the driving scenario. Table 5.1 shows the results for both approaches.

LiDAR-based detection significantly outperforms camera-based detection across all metrics and conditions. The combined F1-score is 91.20% for LiDAR compared to 76.34% for camera. Position error is almost four times lower with LiDAR (0.39m vs 1.72m mean), and orientation error is about five times lower (2.02° vs 9.92° mean).

Camera-based detection shows relatively high position errors, with a combined mean of 1.72m and maximum values approaching 4m. Among the three conditions, snowy presents the most challenging case, with the lowest F1-score (71.23%) and the highest mean position error (2.26m).

LiDAR-based detection maintains high precision and recall across all conditions. Notably, the snowy scenario achieves 100% recall, as LiDAR measures geometry directly and is not affected by changes in visual appearance. Position errors remain consistently below 0.5m mean across all conditions. However, the snowy scenario also presents the lowest precision (82.61%) with 8 false positives, likely due to snow-covered objects being incorrectly detected as vehicles.

Both methods exhibit high maximum orientation errors (up to 86° for camera and 28° for LiDAR), but the low mean values indicate that these are outliers affecting a small number of detections.

Overall, LiDAR is the more reliable detection method. However, camera-based detection still achieves acceptable results and can serve as an alternative when LiDAR data is not available. In both cases, the manual correction step provided by the framework is used to address remaining detection errors before the scenario is loaded in simulation.

Table 5.1: Parked vehicle detection results: Camera-based vs LiDAR-based.

	Sunny		**Cloudy**		**Snowy**		**Combined**	
Metric	Cam	LiDAR	Cam	LiDAR	Cam	LiDAR	Cam	LiDAR
Ground Truth	89	89	90	90	38	38	217	217
Detections	73	83	81	86	35	46	189	215
True Positives	64	**78**	65	**81**	26	**38**	155	**197**
False Positives	9	**5**	16	**5**	9	**8**	34	**18**
False Negatives	25	**11**	25	**9**	12	**0**	62	**20**
Detection								
Precision (%)	87.67	**93.98**	80.25	**94.19**	74.29	**82.61**	82.01	**91.63**
Recall (%)	71.91	**87.64**	72.22	**90.00**	68.42	**100.00**	71.43	**90.78**
F1-Score (%)	79.01	**90.70**	76.02	**92.05**	71.23	**90.48**	76.34	**91.20**
Position								
Pos. Error - Mean (m)	1.665	**0.385**	1.560	**0.442**	2.264	**0.309**	1.724	**0.393**
Pos. Error - Std (m)	0.725	**0.298**	1.021	**0.408**	0.914	**0.269**	0.904	**0.345**
Pos. Error - Max (m)	3.583	**1.600**	3.873	**2.433**	3.865	**0.894**	3.873	**2.433**
Orientation								
Or. Error - Mean (°)	9.73	**1.36**	9.59	**2.92**	10.80	**1.36**	9.92	**2.02**
Or. Error - Std (°)	19.03	**3.03**	16.47	**6.68**	21.01	**4.82**	18.31	**5.12**
Or. Error - Max (°)	76.33	**14.32**	70.56	**28.65**	86.35	**28.65**	86.35	**28.65**

5.1.2 Scenario Reconstruction Similarity

Table 5.2 report the results of the scenario reconstruction similarity. We can observe that background is the class with the highest IoU. This is expected since in our segmentation maps everything that is not road or vehicle is treated as background, making it the dominant class. Road IoU shows values close to 0.90 across all three scenarios, indicating that the CARLA map adequately represents the real-world road geometry. We observe slightly lower values for the snowy scenario, which may be caused by snow on the ground leading to misclassification of road pixels in the real-world segmentation maps.

Vehicle IoU is the lowest across all classes. This can be attributed to differences in vehicle dimensions between the real world and CARLA, since we do not control the exact size of the spawned vehicles in simulation. Additionally, small misalignments in parked vehicle positions can further reduce

Table 5.2: Semantic segmentation IoU results between CARLA and real-world segmentation maps.

Metric	Sunny	Cloudy	Snowy	Combined
BG IoU	0.931 ± 0.035	0.937 ± 0.024	0.929 ± 0.032	0.933 ± 0.031
Car IoU	0.581 ± 0.202	0.552 ± 0.198	0.509 ± 0.253	0.556 ± 0.214
Road IoU	0.915 ± 0.030	0.912 ± 0.030	0.872 ± 0.129	0.905 ± 0.066
mIoU	0.815 ± 0.072	0.802 ± 0.068	0.786 ± 0.118	0.805 ± 0.083

the overlap between real and simulated vehicle pixels.

5.1.3 Summary

The framework estimates parked vehicle coordinates accurately enough for scenario reconstruction, with LiDAR-based detection providing the most reliable results across all tested weather conditions. Camera-based detection is less accurate, but still usable as a fallback solution when LiDAR data is unavailable.

The framework also reconstructs the driving scenarios in simulation with a good degree of fidelity. The high similarity observed for the road and background classes shows that the overall scene layout is reproduced faithfully, while the lower similarity for vehicles indicates that vehicles positioning remains more challenging. These limitations are mainly caused by differences in vehicle dimensions between the real world and CARLA. Nevertheless, the results confirm that the framework can reconstruct the driving scenario with sufficiently high fidelity for our use, which enables us to isolate reconstruction errors from the subsequent research questions.

5.2 RQ2: Generated image quality

5.2.1 Parameters Selection

Stable Diffusion

Preliminary Parameters Selection In the preliminary parameters selection, we found that the configurations labeled as acceptable by human evaluators also ranked in the top three positions by overall metric score across all 100 tested configurations. Table 5.3 reports the average overall score

Table 5.3: Average metric scores for the top 3 configurations identified in the preliminary exploration compared to the remaining 97. All scores are standardized (z-score); higher is better.

Group	N	Overall	Single Image	Vehicle	Temporal
Top 3	3	**1.075 ± 0.051**	**0.893 ± 0.057**	**0.878 ± 0.102**	**1.451 ± 0.103**
Other	97	−0.033 ± 0.507	−0.028 ± 0.617	−0.027 ± 0.941	−0.045 ± 0.924

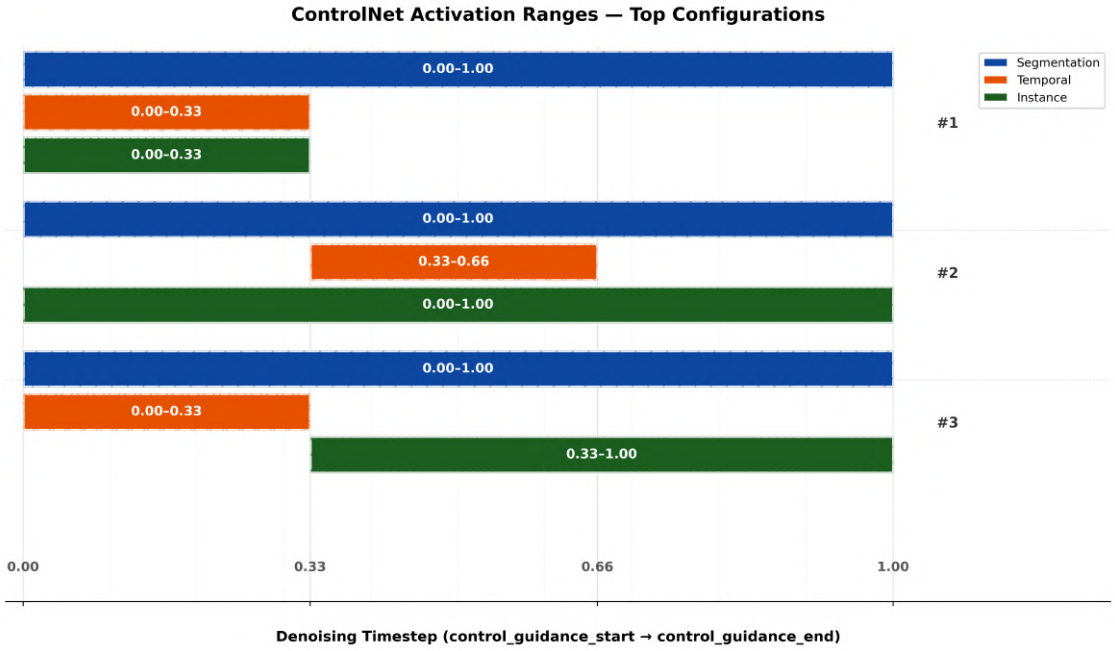


Figure 5.1: Activation ranges of the three ControlNets for the top 3 configurations from the preliminary parameter selection.

and per-category scores with their standard deviations for the top 3 configurations compared to the remaining 97. The top 3 configurations achieved substantially higher scores across all metric categories, indicating a strong agreement between human judgment and the automatic evaluation metrics in this stage of the evaluation.

The three best configurations all share a fixed seed and guess mode set to *TRUE*. Regarding the ControlNet activation schedules, Figure 5.1 provides insights on activation time of each ControlNet. Specifically it shows that the semantic segmentation ControlNet is active for the entire diffusion process in all three configurations. The temporal ControlNet is active only for the first 33% of the process in the first and third configurations, and from 33%

to 66% in the second configuration. The instance segmentation ControlNet shows the most variability across the three configurations, being active over different ranges of the denoising process. Figures 5.2a, 5.2b, and 5.2c present sequences of three generated frames from the three highest-ranked configurations. In contrast, Figures 5.3a, 5.3b, and 5.3c illustrate examples of the worst-performing configurations, respectively for overall image quality, vehicle representation, and temporal consistency. The presented examples show that an inappropriate parameter configuration can lead to artifacts and unrealistic visual appearance in the generated images, even when the same trained Stable Diffusion model is used.

(a) Configuration 1: Seg [0.0–1.0], Inst [0.0–0.33], Temp [0.0–0.33]

(b) Configuration 2: Seg [0.0–1.0], Inst [0.0–1.0], Temp [0.33–0.66]

(c) Configuration 3: Seg [0.0–1.0], Inst [0.33–1.0], Temp [0.0–0.33]

Figure 5.2: Examples of Stable Diffusion generated images with different ControlNet configurations from the coarse grid exploration.

(a) Worst single image quality: Seg [0.0–0.33], Inst [0.0–0.66], Temp [0.0–1.0]

(b) Worst vehicle detection: Seg [0.0–0.33], Inst [0.66–1.0], Temp [0.0–1.0]

(c) Worst temporal consistency: Seg [0.0–0.33], Inst [0.66–1.0], Temp [0.0–0.33]

Figure 5.3: Examples of worst performing Stable Diffusion configurations from the coarse grid exploration, ranked by different metric categories.

Focused Parameter Selection Given the results obtained in the previous section, we identify the following search space rules: I) Seed should be *fixed*.

II) Guess mode should be *TRUE*.
III) The Segmentation ControlNet was active for the entire denoising process in all three top configurations. We therefore consider that the Segmentation ControlNet should start near the beginning, and end near the end of the denoising process.
IV) The Temporal ControlNet was active for exactly one third of the denoising process in all three top configurations, in two different time windows: 0%–33% and 33%–66%. We therefore consider that the Temporal ControlNet should be active for approximately one third of the denoising process, within the first 66%.
V) The Instance ControlNet was active for 100%, 66%, and 33% of the denoising process respectively in the three top configurations, each with different starting points. We were not able to derive a specific rule for this ControlNet, as such we will allow random starting, ending, and duration parameters.

We randomly generated 50 configurations following these rules for the focused evaluation stage. Since the preliminary exploration discretized the control guidance values at fixed intervals, in this stage we refine the search by allowing each parameter to vary at each percentage point of the denoising timeline, while still respecting the overall rules derived from the top 3 configurations. Specifically, for the semantic segmentation ControlNet, we allow it to start after 0% and end before 100%, but always before 33% and after 66% respectively, with the constraint that it must remain active for more than two thirds of the diffusion process, matching the behavior observed in the preliminary evaluation. For the temporal ControlNet, we allow it to be active for more than 33% but less than 66% of the process, with a starting point ranging from 0% to 60%. For the instance segmentation ControlNet, since no consistent pattern was identified, we explore the full range of possible values. Additionally, we maintain the constraint from the preliminary exploration that at least one ControlNet must be active at every timestep of the diffusion process.

Among the focused 50 configurations we selected the best 15 configurations scored by metrics and performed the human study introduced in Section 4.7.2 to find the best possible configuration.

Table 5.4 reports the top 15 configurations selected for the focused parameter study, ordered by human evaluation score, with rank 1 corresponding to the configuration most preferred by the evaluators. For each configuration, the table lists the ControlNet guidance start and end values for the segmentation, instance, and temporal conditions, together with the corresponding

overall automated metric score. Figure 5.4 complements this table by visualizing the same set of configurations from the perspective of the automated evaluation. On the x-axis, the 15 configurations are arranged according to their metric-based ranking, while the red numbers above each group indicate the corresponding human ranking for that configuration. Each group contains three bars showing the normalized scores for single-image quality, vehicle consistency, and temporal consistency. Taken together, the table and the figure show that there is no clear correlation between human preference and automated quality metrics in the focused parameter selection task. Although the automated metrics were effective during the preliminary exploration stage for filtering out clearly poor configurations, they did not align well with human judgment among the top candidates. In particular, the metrics were useful for rejecting unacceptable generations, but less reliable for distinguishing subtle quality differences between visually similar results. For example, the configuration ranked first by human evaluators was placed only 8th by the overall automated metric score. This finding supports the need for human evaluation in the final selection stage when the differences between candidate configurations are small. The configuration ranked first by the evaluators, with ControlNet guidance scales Seg: [0.0–1.0], Inst: [0.0–0.6], Temp: [0.35–0.55], was therefore selected for all subsequent experiments.

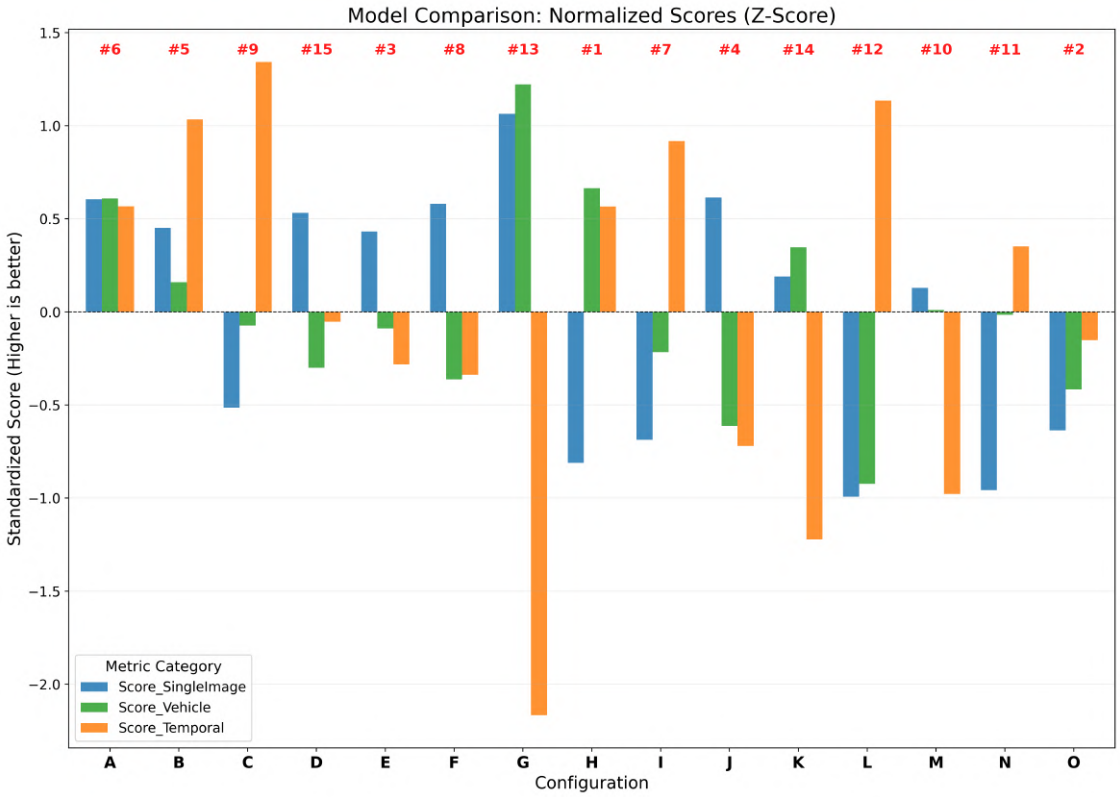


Figure 5.4: Top 15 configuration of ControlNets guidance scales ranked by overall metrics.

NeRF and 3DGS

For NeRF and 3DGS, the automated ranking and the human evaluation were consistent in identifying the undersampling rate of 1/2 as the preferred overall configuration. Unlike the Stable Diffusion parameter study, where the number of tested configurations was too large to report all metric values in detail, here only three configurations were evaluated for each generation model. For this reason, Tables 5.5 and 5.6 report the complete set of quantitative results, grouped into single-image quality, vehicle consistency, and temporal consistency metrics.

For NeRF (Table 5.5), the comparison between 1/2 and 1/3 is relatively close. The 1/2 setting achieves the best MSE (78.58), PSNR (29.18), CPL (4.1220), SegScore (50.16), and temporal CPL (0.2663), whereas 1/3 performs slightly better on SSIM (0.4574 vs. 0.4572), vehicle recall (0.3740 vs. 0.3660), vehicle precision (0.3560 vs. 0.3549), temporal SSIM (0.8487 vs.

Table 5.4: Top 15 configurations from the focused exploration, ranked by human. The selected configuration is highlighted in bold.

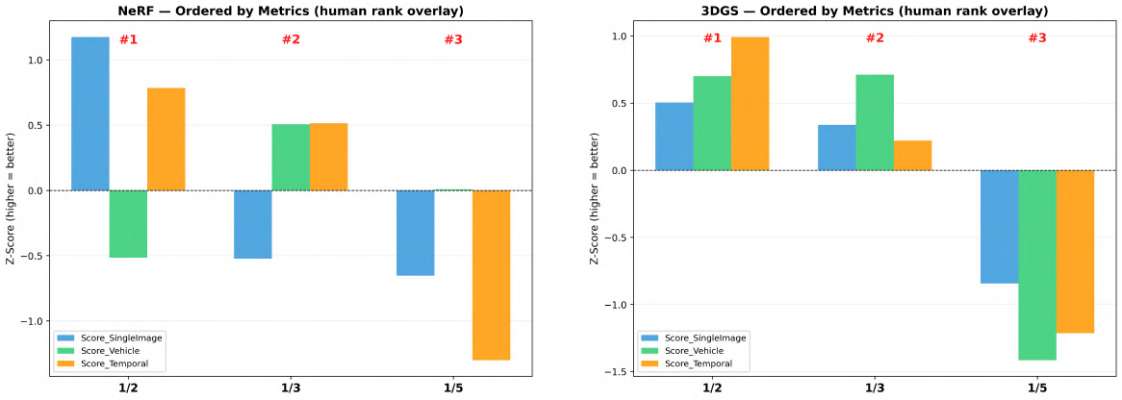
	Guidance Start			Guidance End			
Human Score	**Seg**	**Inst**	**Temp**	**Seg**	**Inst**	**Temp**	**Score Overall**
1	**0.0**	**0.0**	**0.35**	**1.0**	**0.6**	**0.55**	**-0.066**
2	0.0	0.45	0.3	1.0	1.0	0.55	-0.055
3	0.0	0.0	0.0	0.7	1.0	0.25	-0.168
4	0.0	0.4	0.0	1.0	0.8	0.25	-0.177
5	0.0	0.3	0.35	1.0	0.7	0.5	0.513
6	0.0	0.3	0.35	1.0	1.0	0.5	0.644
7	0.0	0.0	0.3	1.0	0.5	0.5	-0.094
8	0.0	0.0	0.0	0.75	1.0	0.25	-0.167
9	0.0	0.3	0.35	1.0	0.8	0.55	0.336
10	0.0	0.0	0.0	0.8	1.0	0.2	-0.398
11	0.0	0.0	0.3	1.0	1.0	0.6	0.045
12	0.0	0.2	0.3	1.0	0.8	0.55	-0.160
13	0.0	0.45	0.0	1.0	1.0	0.2	-0.050
14	0.0	0.0	0.0	0.85	1.0	0.2	-0.350
15	0.0	0.0	0.0	1.0	0.35	0.3	0.147

0.8483), temporal PSNR (29.88 vs. 29.79), and temporal MSE (117.64 vs. 119.55). The 1/5 configuration is generally weaker, although it achieves the highest vehicle average IoU (0.6237 compared to 0.6221 for both 1/2 and 1/3). Overall, the NeRF results show that 1/2 and 1/3 are competitive, with 1/2 being preferred by the final ranking despite 1/3 outperforming it on several individual metrics.

For 3DGS (Table 5.6), the advantage of 1/2 is clearer. It achieves the best MSE (70.17), PSNR (29.67), CPL (2.5023), vehicle recall (0.3652), temporal SSIM (0.8372), temporal PSNR (31.01), temporal MSE (121.42), and temporal CPL (0.1079). The 1/3 configuration only slightly improves over 1/2 in SegScore (49.89 vs. 50.03), vehicle precision (0.3563 vs. 0.3562), and vehicle average IoU (0.6245 vs. 0.6224), while 1/5 gives the highest SSIM (0.4610) but is substantially worse on several other metrics, including CPL (3.9879), SegScore (60.68), and temporal MSE (231.03). These results indicate that, for 3DGS, the 1/2 configuration provides the strongest and most balanced performance overall.

Therefore, unlike the focused Stable Diffusion parameter selection task,

where the top configurations were not well separated by the automated metrics, in this validation setting the quantitative evaluation is much more consistent with human judgment. In particular, Figure 5.5 shows that the overall metric-based ranking is aligned with the human evaluation for both NeRF and 3DGS, even though, in the NeRF case, some individual metrics still favor the 1/3 configuration.



(a) NeRF (b) 3DGS

Figure 5.5: Categories metrics scores ordered by Score_Overall with human ranking overlay for NeRF and 3DGS validation configurations across undersample rates.

Table 5.5: Quantitative evaluation of NeRF across undersample rates on the validation scenario.

	Undersample Rate		
Metric	**1/2**	**1/3**	**1/5**
Single Image			
MSE ↓	**78.58**	85.14	83.63
PSNR ↑	**29.18**	28.84	28.91
SSIM ↑	0.4572	**0.4574**	0.4535
CPL ↓	**4.1220**	4.3390	4.2139
SegScore ↓	**50.16**	50.19	50.21
Vehicle Consistency			
Veh. Recall ↑	0.3660	**0.3740**	0.3699
Veh. Precision ↑	0.3549	**0.3560**	0.3518
Veh. Avg IoU ↑	0.6221	0.6221	**0.6237**
Temporal Consistency			
Temp. SSIM ↑	0.8483	**0.8487**	0.8402
Temp. PSNR ↑	29.79	**29.88**	29.23
Temp. MSE ↓	119.55	**117.64**	134.65
Temp. CPL ↓	**0.2663**	0.2904	0.2984

Table 5.6: Quantitative evaluation of 3DGS across undersample rates on the validation scenario.

	Undersample Rate		
Metric	**1/2**	**1/3**	**1/5**
Single Image			
MSE ↓	**70.17**	70.39	73.18
PSNR ↑	**29.67**	29.66	29.51
SSIM ↑	0.4539	0.4536	**0.4610**
CPL ↓	**2.5023**	2.7824	3.9879
SegScore ↓	50.03	**49.89**	60.68
Vehicle Consistency			
Veh. Recall ↑	**0.3652**	0.3646	0.3260
Veh. Precision ↑	0.3562	**0.3563**	0.3233
Veh. Avg IoU ↑	0.6224	**0.6245**	0.5493
Temporal Consistency			
Temp. SSIM ↑	**0.8372**	0.8323	0.8324
Temp. PSNR ↑	**31.01**	30.28	28.73
Temp. MSE ↓	**121.42**	126.33	231.03
Temp. CPL ↓	**0.1079**	0.1192	0.4313

(a) Ground Truth Real World Image

(b) 3DGS

(c) NeRF

(d) SD

Figure 5.6: Comparison of the same frame across generation techniques. Ground truth from Real World, followed by 3DGS, NeRF, and Stable Diffusion outputs.

5.2.2 Generated Image Quality

Table 5.7: Qualitative evaluation of trajectory replay across weather conditions and generation methods.

	Cloudy			Snowy			Sunny		
Metric	3DGS	NeRF	SD	3DGS	NeRF	SD	3DGS	NeRF	SD
Single Image									
MSE ↓	**79.07**	88.58	97.62	**82.66**	90.55	99.41	**79.24**	87.30	97.06
PSNR ↑	**29.17**	28.69	28.24	**28.98**	28.57	28.16	**29.17**	28.74	28.27
SSIM ↑	0.3835	**0.3905**	0.3685	0.3692	**0.3765**	0.3327	0.4067	**0.4120**	0.3992
CPL ↓	**3.7616**	4.6740	6.4171	**4.9581**	5.6666	9.3890	**4.0396**	4.4642	6.1026
SegScore ↓	59.16	59.45	**48.29**	54.70	53.31	**42.45**	61.66	62.30	**46.80**
Vehicle Consistency									
Veh. Recall ↑	0.2628	0.2386	**0.3268**	**0.3123**	0.3113	0.2814	0.2926	0.2936	**0.3218**
Veh. Precision ↑	0.2801	0.2581	**0.3753**	0.3218	0.3319	**0.3578**	0.3003	0.3051	**0.3561**
Veh. Avg IoU ↑	0.4221	0.4054	**0.4636**	0.4004	**0.4006**	0.3773	0.4348	0.4308	**0.4602**
Distribution									
FID ↓	**31.18**	44.35	99.32	**42.05**	55.71	115.37	**31.78**	35.25	91.44
KID ↓	**0.0085**	0.0144	0.0707	**0.0140**	0.0167	0.0637	**0.0088**	0.0090	0.0516
IS ↑	**2.8071**	2.7369	2.1187	2.8088	**2.8384**	2.1629	**2.7176**	2.6836	2.1309
MMD-RBF ↓	**0.0126**	0.0195	0.0944	**0.0183**	0.0208	0.0843	**0.0127**	0.0133	0.0722
PRDC Prec. ↑	**0.0019**	0.0013	0.0000	**0.0091**	0.0000	0.0000	0.0000	0.0000	0.0000
PRDC Rec. ↑	0.0000	**0.0013**	0.0000	**0.0097**	0.0000	0.0000	0.0000	0.0000	0.0000
PRDC Dens. ↑	1.3358	**1.3364**	1.3334	1.3406	1.3377	**1.3421**	1.3583	**1.3615**	1.3493
PRDC Cov. ↑	**0.0013**	0.0010	0.0000	**0.0051**	0.0000	0.0000	0.0000	0.0000	0.0000
Temporal Consistency									
Temp. SSIM ↑	0.7662	0.7791	**0.9546**	0.6906	0.7185	**0.9579**	0.8107	0.8196	**0.9643**
Temp. PSNR ↑	26.83	27.50	**31.97**	25.69	26.33	**32.87**	28.97	29.05	**33.48**
Temp. MSE ↓	255.18	204.79	**59.77**	291.26	217.26	**49.04**	169.21	154.24	**43.22**
Temp. CPL ↓	**0.2200**	0.2946	0.2767	0.5378	0.5857	**0.5172**	**0.1971**	0.2271	0.2207

The Table 5.7 presents the results of Neural Rendering Techniques considered in our evaluation across all the presented metrics. Results are obtained by comparing generated image using trajectory replay 3.6.1 and their baseline real-world counterparts.

Single Image Quality. From the results reported in Table 5.7, we can see that in single image metrics, 3DGS consistently achieved the best results across all weather conditions in MSE (e.g., 79.07 vs. 88.58 for NeRF and 97.62 for SD in cloudy), PSNR (e.g., 29.17 vs. 28.69 and 28.24), and CPL (e.g., 3.76 vs. 4.67 and 6.42), while NeRF obtained the best results in terms of SSIM (e.g., 0.3905 vs. 0.3835 for 3DGS) and SD achieved the best SegScore (e.g., 48.29 vs. 59.16 for 3DGS and 59.45 for NeRF), while instead it performed considerably worse in each of the other metrics. This ranking reflects the fundamental difference in how these techniques generate images: 3DGS and NeRF synthesize novel views by rendering a reconstructed

3D representation of the scene from a new viewpoint. Stable Diffusion, on the other hand, generates images through a learned denoising process conditioned on structural inputs, which introduces a higher degree of variation from the real-world appearance.

Distribution Metrics. The distribution-level metrics reinforce this observation, confirming that 3DGS and NeRF produce image distributions that are closer to the real-world distribution than those generated by Stable Diffusion. In particular, 3DGS achieved the best FID (31.18 in cloudy, compared to 44.35 for NeRF and 99.32 for SD), KID (0.0085 vs. 0.0144 and 0.0707), and MMD-RBF (0.0126 vs. 0.0195 and 0.0944) scores, outperforming NeRF, which in turn performed better than Stable Diffusion.

Vehicle Consistency. Interestingly, the vehicle consistency metrics show the opposite trend, with Stable Diffusion achieving better results than 3DGS and NeRF in terms of vehicle positioning. For example, in the cloudy condition, SD obtained a vehicle precision of 0.3753 and an average IoU of 0.4636, compared to 0.2801 and 0.4221 for 3DGS, and 0.2581 and 0.4054 for NeRF. This is due to alignment errors introduced during trajectory replay, as illustrated in Figure 5.7: although 3DGS and NeRF produce images that are closer to the real world in terms of visual quality, small errors in the trajectory alignment can place the virtual camera at slightly different positions than the original real-world viewpoints. This causes vehicles to appear at different locations than in the ground truth images, reducing the vehicle detection matching scores. Stable Diffusion does not suffer from this issue because it is conditioned on the semantic segmentation maps from the simulator, where vehicles are placed by our framework at positions matching the real world. As a result, vehicles in the generated images appear at the same locations as in the ground truth, regardless of camera alignment errors.

Temporal Consistency. Perhaps the most counterintuitive result is that Stable Diffusion achieves better temporal consistency scores than 3DGS and NeRF, despite producing visually less consistent sequences, as shown in Figure 5.8. For instance, in the cloudy condition, SD achieved a temporal SSIM of 0.9546 and a temporal MSE of 59.77, compared to 0.7662 and 255.18 for 3DGS, and 0.7791 and 204.79 for NeRF. When inspecting the generated videos, 3DGS and NeRF maintain a coherent scene across frames, while Stable Diffusion can produce noticeable changes in objects and background

Figure 5.7: Frames generated from the same position by the three techniques compared against the ground truth. Left: ground truth. Center: 3DGS and NeRF, which exhibit perspective misalignment due to trajectory replay errors. Right: Stable Diffusion, whose perspective more closely resembles the real-world viewpoint thanks to its conditioning on semantic segmentation maps.

between consecutive frames. However, the temporal metrics compare pairs of consecutive frames at the pixel level: in Stable Diffusion, although the scene content may change, a large portion of pixels between consecutive frames remains identical due to the conditioning on similar structural inputs, as visible in the heatmaps in Figure 5.9. In contrast, 3DGS and NeRF, despite maintaining a globally consistent scene, introduce small pixel-level variations between consecutive frames due to the rendering process, which the temporal metrics penalize more heavily. This highlights a limitation of pixel-based temporal consistency metrics, which do not fully capture the perceptual coherence of a sequence.

5.2.3 Summary

3DGS generates the most similar images compared to their real-world counterparts, consistently achieving the best results in single image quality metrics (MSE, PSNR, CPL) and distribution metrics (FID, KID, MMD-RBF) across all weather conditions. NeRF follows closely, obtaining the best SSIM scores and ranking second in most other metrics. Stable Diffusion produces the least similar images in terms of visual fidelity, due to the inherent variability of its generative denoising process. However, Stable Diffusion achieves better vehicle consistency scores thanks to its conditioning on semantic segmentation maps, which avoids the perspective misalignment issues that affect 3DGS and NeRF during trajectory replay. Similarly, Stable Diffusion obtains higher temporal consistency scores at the pixel level, although visual inspection reveals that 3DGS and NeRF produce perceptually more coherent sequences across frames, highlighting a limitation of the pixel-based metrics

(a) 3DGS: coherent scene appearance with small pixel-level variations.

(b) Stable Diffusion: noticeable changes in objects and background despite higher pixel-level temporal consistency scores.

(c) NeRF: coherent scene appearance similar to 3DGS.

Figure 5.8: Temporal consistency comparison: sequences of frames generated by the three techniques across 1/3 of second.

proposed to evaluate temporal consistency. These results highlight that image quality metrics alone do not fully capture the differences between these techniques: while 3DGS and NeRF are superior in visual fidelity, Stable

(a) 3DGS heatmap of different pixels in consecutive generated frames.

(b) Stable Diffusion heatmap of different pixels in consecutive generated frames.

Figure 5.9: Heatmaps of pixel-level differences between generated frames and the corresponding ground truth for 3DGS and Stable Diffusion.

Diffusion benefits from its conditioning mechanism in terms of structural alignment.

5.3 RQ3: System level Behavior

Table 5.8 reports the system-level testing results, including the fail rate (number of runs in which the driving model failed to complete the route), the completion rate (average, maximum, and minimum percentage of the route completed), and the failure type classification (out of road, car crash, oversteer, or understeer) for each technique and weather condition. Figure 5.10 shows the corresponding driving trajectories produced by the driving model in each domain across sunny, cloudy, and snowy conditions, where the real-world trajectory serves as the baseline for comparison.

From the success/failure metrics reported in and the trajectory plots in Figure 5.10, we first observe that in raw simulation runs without neural rendering applied, the SUT never completed the driving scenario, showing a completion rate consistently below 40% across all conditions. This confirms the existence of the reality gap and motivates the use of proposed techniques to mitigate it.

Among the tested techniques, 3DGS achieved the best results. The SUT,

when fed with 3DGS-generated images, successfully completed all runs in both sunny and cloudy conditions, matching its behavior in the real world. From Figure 5.10(b), we also observe that in snowy conditions, two out of three runs failed near the same location where a real-world failure occurred. However, in one run the SUT exceeded the real-world completion rate, stopping shortly after due to 3DGS being unable to render views from positions never seen during training. This highlights a risk, as it could provide overconfidence in the SUT's capabilities, and underlines the difficulty of perfectly replicating real-world physical conditions in simulation.

NeRF is the second best performing technique in terms of completion rate. With NeRF images as input, the SUT successfully completed all runs in the cloudy condition, matching the real-world baseline. However, in the sunny condition the SUT never reached the end of the scenario as it did in the real world and with 3DGS employed, failing at a point where no failures were observed in the real-world baseline. From Figure 5.10(c), in the snowy condition the SUT surpassed two of the real-world runs in terms of completion rate, but never reached the maximum completion observed in the baseline, presenting failures in a specific location that however was never observed in the real world.

Stable Diffusion is the worst performing technique. The SUT with Stable Diffusion completed the full scenario only twice, both in cloudy conditions, and produced the least deterministic results, with large variation between maximum and minimum completion rates across runs. It also exhibited the most varied failure types across scenarios, including out-of-road, collisions, oversteering, and understeering. This may be attributed to the fundamental difference in approach: while NeRF and 3DGS are novel view synthesis techniques that reconstruct a 3D representation of the scene Stable Diffusion is a generative model that synthesizes images from learned distributions, where small variations in the conditioning inputs can produce notably different outputs.

Notably, in the snowy condition, Stable Diffusion achieved a lower completion rate than raw simulation without any neural rendering applied, suggesting that the generated images actively degraded driving performance rather than improving it.

However with this exception, the use of neural rendering techniques consistently provided a measurable improvement in the SUT's driving performance compared to raw synthetic images, confirming their effectiveness in reducing the sim-to-real gap.

Table 5.9 reports driving quality metrics for successful runs. Since cloudy

Table 5.8: Task completion results across weather conditions (all runs).

Metric	Domain	Sunny	Cloudy	Snowy
Fail Rate	Real	0/3	0/3	3/3
	3DGS	0/3	0/3	3/3
	NeRF	3/3	0/3	3/3
	SD	3/3	1/3	3/3
	Sim	3/3	3/3	3/3
Completion Rate (%) avg–max–min	Real	100 - 100 - 100	100 - 100 - 100	47 - 70 - 34
	3DGS	100 - 100 - 100	100 - 100 - 100	70 - 72 - 70
	NeRF	71 - 72 - 71	100 - 100 - 100	58 - 58 - 58
	SD	38 - 39 - 36	71 - 100 - 13	16 - 16 - 15
	Sim	30 - 36 - 22	21 - 23 - 20	25 - 34 - 13
Failure Type* OR–CC–OS–US	Real	–	–	0 - 0 - 0 - 3
	3DGS	–	–	0 - 1 - 0 - 2
	NeRF	0 - 0 - 3 - 0	–	0 - 0 - 0 - 3
	SD	0 - 0 - 0 - 3	0 - 1 - 0 - 0	2 - 1 - 0 - 0
	Sim	0 - 1 - 0 - 2	2 - 1 - 0 - 0	2 - 1 - 0 - 0

*OR = Out of Road, CC = Car Crash, OS = Oversteer, US = Understeer.

is the only condition where the SUT completed the scenario at least once for each technique, we only compare the techniques in this condition.

NeRF reported the lowest Fréchet distance (2.13 m), with 3DGS closely behind (2.22 m), while SD performed notably worse (3.83 m). NeRF also achieved the best corridor violation rate (63.2%), staying inside the corridor of the real world trajectoreies for more than 3DGS (67.3%) and SD (70.8%). However the mean excess distance shows that 3DGS produced the smallest deviations when outside the corridor (0.330 m), compared to NeRF (0.425 m) and SD (0.544 m), indicating that while 3DGS exits the corridor more frequently then NeRF, it remains closer to the boundary. The excess when out metric further confirms this pattern, showing the average distance from the corridor boundary specifically at the points where the trajectory is outside: 3DGS (0.491 m), NeRF (0.673 m), and SD (0.767 m). It is worth noting that despite all techniques being outside the corridor for a significant portion of the trajectory, they still completed the driving scenario successfully. This suggests that there are more valid driving trajectories than those defined by the real-world runs alone.

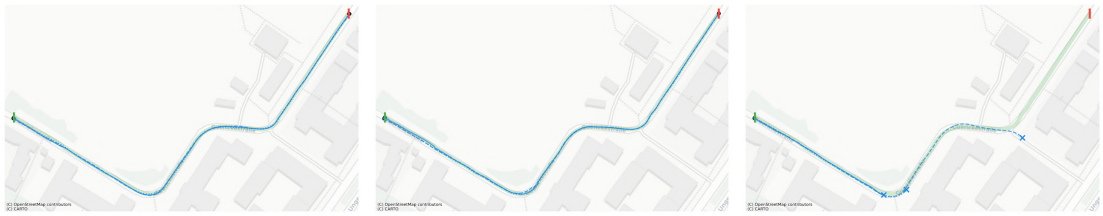
Comparing 3DGS across sunny and cloudy conditions, we observe a clear degradation in sunny (Fréchet 3.57 m vs 2.22 m, corridor violation 78.7% vs 67.3%), which may indicate that the SUT struggles more under sunny conditions, potentially due to stronger shadows and lighting variations in

Table 5.9: Drive quality metrics across weather conditions (successful runs only).

Metric	Domain	Sunny	Cloudy	Snowy
Trajectory Similarity				
	Real	–	–	–
	3DGS	3.57	2.22	–
Frechet Dist. (m)	NeRF	–	2.13	–
	SD	–	3.83	–
	Sim	–	–	–
	Real	–	–	–
	3DGS	78.7	67.3	–
Corridor Viol. (%)	NeRF	–	63.2	–
	SD	–	70.8	–
	Sim	–	–	–
	Real	–	–	–
	3DGS	0.608	0.330	–
Mean Excess (m)	NeRF	–	0.425	–
	SD	–	0.544	–
	Sim	–	–	–
	Real	–	–	–
	3DGS	0.769	0.491	–
Excess When Out (m)	NeRF	–	0.673	–
	SD	–	0.767	–
	Sim	–	–	–
Steering Smoothness				
	Real	1.127	1.167	–
	3DGS	1.666	0.867	–
Avg Jitter (rad/s)	NeRF	–	0.880	–
	SD	–	1.780	–
	Sim	–	–	–
	Real	9.818	6.452	–
	3DGS	13.762	4.983	–
Max Jitter (rad/s)	NeRF	–	5.881	–
	SD	–	12.751	–
	Sim	–	–	–

the reconstructed images.

Regarding steering smoothness, both 3DGS and NeRF produced lower average jitter than the real-world baseline in cloudy conditions (0.867 and 0.880 rad/s respectively, vs 1.167 rad/s). SD, exhibited significantly higher average jitter (1.780 rad/s) with a peak of 12.75 rad/s, nearly double the real-world maximum of 6.45 rad/s.

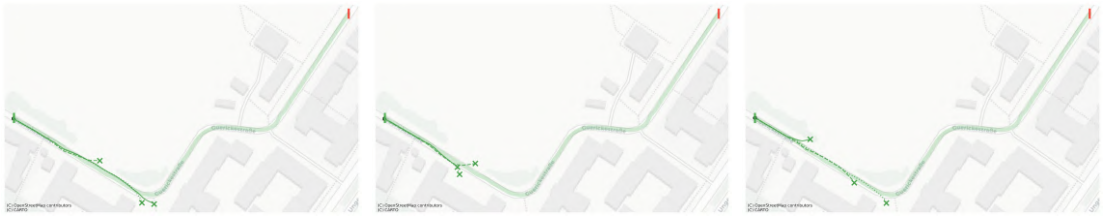


(a) Real-world driving trajectories

(b) 3DGS driving trajectories

(c) NeRF driving trajectories

(d) Stable Diffusion driving trajectories



(e) Raw simulation driving trajectories

Figure 5.10: System-level driving trajectories across weather conditions (left: sunny, center: cloudy, right: snowy).

5.3.1 Summary

The system-level results show that the ranking of neural rendering techniques in terms of driving behavior is consistent with the image quality metrics results from RQ2: 3DGS best replicates the real-world driving behavior, followed by NeRF, while Stable Diffusion and raw simulation produce significantly worse results. In sunny and cloudy conditions, 3DGS is the only technique that matches the real-world outcome, completing the route successfully in both cases. NeRF completes the route in cloudy but fails in sunny due to oversteer, while Stable Diffusion fails in sunny and partially completes the route in cloudy. Raw simulation consistently fails across all conditions, confirming the severity of the sim-to-real perception gap when no neural rendering technique is applied. In terms of driving quality, 3DGS and NeRF produce trajectories that closely follow the real-world baseline, with Fréchet distances of 2.22 m and 2.13 m respectively in cloudy, and steering smoothness comparable to or better than the real-world runs. Stable Diffusion, despite completing some runs, exhibits larger trajectory deviations (3.83 m) and significantly higher steering jitter, indicating less stable driving behavior.

Chapter 6

Discussion

In RQ1, we evaluated how accurately our framework reconstructed the driving scenario in simulation. We first compared the LiDAR-based 3.3.2 and camera-based 3.3.1 parked-vehicle detection approaches, and the results showed that the LiDAR-based approach performed better across all evaluated metrics (Table 5.1). In particular, it achieved better results in vehicle detection, positional accuracy, and orientation estimation, with an average positional error of **0.393 m** compared to **1.724 m** for the camera-based approach. These results showed that LiDAR sensing was more effective for parked-vehicle detection and, consequently, for reconstructing the simulated scenarios. We then compared the semantic maps of the generated scenarios against their real-world counterparts and observed overall good performance in terms of IoU across object classes reaching a mIoU over classe and scenario of 0.805(Table 5.2).

In RQ2, we compared the image quality metrics of the images produced by each generation technique. Through an initial preliminary parameter selection, followed by a more focused parameter search guided by human evaluation, we found that the considered metrics were useful for ranking images when quality differences and artifacts were pronounced, but were less effective when comparing similar images. Based on both quantitative metrics and human scores, we selected the best parameter configurations for SD, NeRF, and 3DGS. Our experiments showed that, for both 3DGS and NeRF, stronger undersampling led to worse generation quality. For SD, we inferred that generation quality improved when guess mode was enabled and the segmentation ControlNets remained active most of the time, as they provided important scene understanding cues.

Among the evaluated techniques (Table 5.7), 3DGS achieved the best

overall results in terms of single-image quality, scoring best on MSE, PSNR, and CPL. For example, it reached MSE values as low as **79.07**, PSNR values up to **29.17**, and CPL values down to **3.7616**, while NeRF obtained the highest SSIM values, up to **0.4120**. At the distribution level, 3DGS also performed better than the other techniques across most of the metrics, for instance achieving FID values as low as **31.18** and KID values down to **0.0085**.

We also observed that, according to the vehicle consistency metrics, Stable Diffusion performed best overall. We attributed this result to the fact that, despite their higher image quality, NeRF and 3DGS were more difficult to align with the real-world trajectories due to misalignment errors produced by COLMAP. As a result, they often produced incorrect perspectives, with vehicles appearing in positions inconsistent with their real-world counterparts. In contrast, Stable Diffusion, conditioned on synthetic semantic maps, generated vehicle placements that were more coherent with the real-world scenes. This trend was reflected, for example, in vehicle precision, where Stable Diffusion reached values up to **0.3753**, higher than both 3DGS and NeRF in the same setting.

Somewhat surprisingly, the temporal consistency metrics also favored Stable Diffusion in most cases, with the exception of temporal CPL, where 3DGS performed better. This result was counterintuitive, since Stable Diffusion frequently introduced visually noticeable inconsistencies across frames, such as vehicles changing color or shape over time. This suggested that these metrics were more sensitive to low-level pixel changes than to perceptually meaningful temporal coherence. Indeed, by visually inspecting consecutive frames, we inferred that 3DGS and NeRF were penalized by small pixel-level color variations caused by motion, even though they appeared more temporally coherent overall. For instance, Stable Diffusion achieved temporal SSIM values up to **0.9643** and temporal PSNR up to **33.48**, despite its evident visual inconsistencies.

The main conclusion from this research question was therefore that 3DGS produced the highest visual quality images, followed by NeRF, with Stable Diffusion ranking last in this respect. However, Stable Diffusion generated images that were more semantically consistent with the environment than our implementations of the neural rendering techniques, while the temporal metrics considered in this work did not appear suitable for assessing true frame-to-frame coherence.

In RQ3, we used the real-world behavior of our SUT as a baseline to measure the sim-to-real gap in terms of driving behavior, rather than relying

only on image quality metrics. We first confirmed the existence of a reality gap by testing the model in simulation using only raw synthetic images, under which the SUT never reached the end of the scenario in all the three tested weather conditions. We then evaluated the SUT behavior using images generated by the different techniques and found that 3DGS produced behavior that most closely matched the real-world baseline. We also observed one case in which the SUT slightly outperformed its real-world counterpart when driven using 3DGS-rendered images, highlighting another limitation of simulation-based testing: the possibility of generating overconfident results. In terms of completion rate, Stable Diffusion performed worst, with the SUT able to complete only two runs when provided with images generated by this technique (Table 5.8).

Looking at the driving quality metrics (Table 5.9), 3DGS was the only technique that produced successful runs across multiple weather conditions, completing scenarios in both sunny and cloudy settings, whereas NeRF and Stable Diffusion only completed runs under cloudy conditions. No technique succeeded in snowy scenarios. Under cloudy conditions, where all three techniques could be compared directly, NeRF and 3DGS produced trajectories closest to the real-world baseline, with Fréchet distances of 2.13 m and 2.22 m, respectively, while Stable Diffusion deviated more substantially, reaching 3.83 m. A similar pattern emerged for corridor violations and mean excess deviation, where 3DGS and NeRF consistently outperformed Stable Diffusion. Interestingly, both 3DGS and NeRF produced smoother steering profiles than the real-world baseline, with average jitter values of 0.867 rad/s and 0.880 rad/s, compared to 1.167 rad/s in reality, whereas Stable Diffusion showed significantly more jittery behavior at 1.780 rad/s. This suggested that the images generated by neural rendering techniques may have lacked some of the visual complexity present in real-world scenes, resulting in overly smooth driving behavior and potentially overconfident simulation outcomes.

Overall, we observed a sim-to-real gap when comparing the results obtained in simulation with those measured in the real world. At the same time, the ranking suggested by the image quality metrics was broadly consistent with the behavioral results from RQ3 (Tables 5.8 and 5.9)Specifically, 3DGS, which achieved the best overall image quality, was also the technique under which the SUT exhibited driving behavior most similar to that observed in the real world. Likewise, NeRF, which scored higher than Stable Diffusion in terms of image quality, also led to SUT behavior that more faithfully reproduced the real-world baseline. This suggested that image quality

metrics, while not sufficient on their own, could still serve as a meaningful proxy for predicting how well a rendering technique supported realistic system-level behavior in simulation-based testing.

Other aspects worth discussing were training time and inference performance. Among the evaluated techniques, 3DGS again provided the best overall results. On our sunny scenario, comprising 3,895 frames, training each model split with 3DGS required a total of 45 minutes. In contrast, training NeRF on the same scenariorequired almost three hours, while fine-tuning Stable Diffusion together with its ControlNets required a total of five hours

A similar trend was observed at inference time. 3DGS achieved by far the best performance, reaching a generation speed of nearly 100 FPS. In comparison, both NeRF and Stable Diffusion required approximately two seconds per frame. These results further highlighted the practical advantage of 3DGS, which not only produced the best visual quality overall, but also offered substantially lower training and inference costs.

Despite its weaker performance in terms of visual quality and behavioral fidelity, Stable Diffusion could still be useful for ADS testing because it offered a level of flexibility that NeRF and 3DGS did not natively provide. In particular, diffusion-based methods made it easier to generate alternative scene configurations and to modify relevant elements of the environment, for example by adding, removing, or repositioning vehicles. By contrast, NeRF and 3DGS were fundamentally tied to the geometry and appearance of the recorded scene and were therefore better suited for faithful reconstruction than for controlled scenario modification. For this reason, Stable Diffusion remained a promising option when the goal was not only to reproduce a real-world scenario as accurately as possible, but also to create diverse test conditions for probing the robustness of the system under test.

6.1 Threats to validity

Driving model limitations. In this study, we limited the driving model to predicting steering commands only, while throttle and braking were controlled by a human driver. Although allowing the model to control both steering and speed would have enabled a more complete analysis of driving behavior, this choice was made to ensure safety during real-world testing. To obtain consistent and trustworthy results in simulation, we fixed the vehicle speed at 10 km/h, which was close to the mean speed observed in the training

dataset.

Dynamic actors. We evaluated an interesting driving scenario characterized by the absence of parking lanes and by a wide variety of parked vehicle positions, including vehicles parked partially over drivable lanes. However, we considered only scenarios without dynamic actors. Including moving vehicles, pedestrians, or other road users would have required substantially more training and testing before the self-driving model could be safely deployed in real-world conditions.

Driving scenario. We conducted our evaluation on a single driving scenario. Although this limits the generalizability of the results, the choice was made because collecting data and training all the neural rendering techniques on additional scenarios would have required significantly more time and effort. Moreover, testing the SUT on a new scenario would have required additional data collection for both training and evaluation, and would likely have demanded many more weeks of experimentation, since just for collecting training and testing data on a single scenario demanded days of experimentation in different seasons. To mitigate the effects of this limitation, we varied the testing conditions collecting our testing dataset in different days, with different parked cars configurations and in different weather conditions.

Single driving model. The behavioral findings are dependent on the specific NVIDIA PilotNet-based driving model used in this study and may differ when considering other models. Nevertheless, since our results are evaluated relative to a real-world reference behavior, the findings regarding sim-to-real gap mitigation are expected to remain valid.

6.2 Future work

A direction that should be explored in future work is the adoption of hybrid approaches combining Stable Diffusion and 3DGS to modify the generated scenario and make the framework more suitable for ADS testing and failure identification. As discussed in this thesis, 3DGS alone does not support the addition, removal, or repositioning of vehicles within the scene. Stable Diffusion, or other generative techniques, could help address this limitation, either by modifying the training set used for Gaussian Splatting or by applying edits directly at inference time. This would also make it possible to introduce dynamic actors into the scenario, which would be necessary for testing ADS in more realistic and failure-prone conditions.

It would also be interesting to investigate dedicated editing techniques for

3DGS and NeRF reconstructions, such as Instruct-GS2GS [112] and Instruct-NeRF2NeRF [113]. These methods could make it possible to artificially vary weather conditions and other scene properties in the generated environment, enabling the driving model to be tested under different environmental conditions.

Chapter 7

Conclusion

Autonomous driving systems combine hardware and software components that enable a vehicle to perceive its environment and make driving decisions without human intervention [1]. Because these systems operate in safety-critical environments, failures can lead to severe real-world consequences, highlighting the need for rigorous validation before deployment [26, 32, 33, 34]. Real-world testing provides the most reliable evaluation since the system operates directly within its deployment domain [26]. However, it is costly, difficult to reproduce, and potentially unsafe due to environmental variability such as weather, traffic conditions, and lighting [26].

Alternative testing methodologies have therefore been explored. Model-level testing using prerecorded driving datasets enables reproducible evaluation with realistic sensor recordings [35, 37, 40, 38, 39], but it cannot capture the closed-loop interaction between the driving system and its environment because predicted actions do not influence future observations [35]. Simulation-based testing instead allows closed-loop evaluation in controllable and repeatable virtual environments, and modern simulators such as CARLA, SVL Simulator, and AirSim provide realistic physics and sensor simulation widely used in autonomous driving research [55, 48, 49, 51]. However, simulated environments inevitably differ from the real world in visual appearance, physical modeling, and environmental dynamics, leading to the *sim-to-real gap*, where systems that perform well in simulation may behave differently in real conditions [45, 25]. A key component of this discrepancy is the perception gap caused by differences between synthetic and real sensor observations [46]. Because many autonomous driving systems rely on learning-based perception models, even small shifts in the input distribution can significantly affect system behavior [53].

To mitigate this problem, recent research has explored the use of generative models and neural rendering techniques capable of transforming simulated observations into more realistic ones [65, 80]. These approaches aim to reduce the visual discrepancy between simulated and real environments so that perception systems operate under conditions closer to those encountered during real-world deployment [80]. However, most prior studies evaluate these techniques primarily using image similarity metrics such as SSIM or FID [96, 98], while recent work suggests that improvements in visual metrics do not necessarily translate into improved driving behavior [94]. Furthermore, existing approaches are often evaluated on static datasets rather than within dynamic simulation environments where the system interacts with the environment in a closed loop [75, 77, 79, 81].

To address these limitations, this thesis proposed a framework that reconstructs real-world driving scenarios inside a simulator and integrates neural rendering techniques to generate realistic visual inputs for simulation-based testing. The framework enables a direct comparison of different techniques not only in terms of image quality but also with respect to the system-level behavior of an autonomous driving model.

To achieve this goal, we first trained an end-to-end camera-based driving model based on the NVIDIA PilotNet architecture [21] using real-world driving recordings collected with a full-size vehicle. The trained model was then deployed on the same vehicle and evaluated on a public urban road, producing a real-world baseline for comparison. Starting from these recordings, the proposed framework reconstructs the corresponding driving scenario inside the CARLA simulator [48]. The pipeline extracts sensor data from ROS recordings [87], including camera frames, LiDAR point clouds, and vehicle trajectory information, and processes them to reconstruct the scenario geometry and environmental configuration.

A key step in this process is the detection and localization of parked vehicles present in the recorded scenario. To achieve this, the framework supports both camera-based and LiDAR-based detection pipelines, relying on deep learning techniques such as FCOS3D [83] for monocular 3D detection and PointPillars [86] for point-cloud-based object detection. The reconstructed road topology is obtained from OpenStreetMap data [44] and converted into the OpenDRIVE format [61], allowing the scenario to be recreated in the simulator with the same trajectory and vehicle configuration observed in the real-world recordings.

Once the scenario is reconstructed, neural rendering models are trained using the real-world data extracted from the recordings. The framework

supports three different approaches: diffusion-based image generation using Stable Diffusion with ControlNet [62, 71], neural radiance fields for novel view synthesis [59], and 3D Gaussian Splatting for real-time neural rendering [60]. These models are trained using real camera images together with their estimated camera poses obtained through structure-from-motion methods such as COLMAP [74]. During simulation, the trained models generate images corresponding to the current vehicle viewpoint, replacing or transforming the raw synthetic images produced by the simulator.

The evaluation of the framework was guided by three research questions. The first investigated the fidelity of the reconstructed scenarios, focusing on the accuracy of parked vehicle detection and the overall consistency between the reconstructed environment and the real-world recordings. The results showed that the LiDAR-based detection pipeline significantly outperformed the camera-based approach, reducing the average localization error from 1.724 m to 0.393 m, while the reconstructed environments also showed strong semantic consistency with the real-world scenes.

The second research question analyzed the visual quality of the generated images produced by the neural rendering techniques. The results showed that 3D Gaussian Splatting produced the best overall reconstruction quality, while NeRF achieved strong structural similarity and Stable Diffusion provided higher temporal stability and object consistency, highlighting different strengths among the evaluated techniques.

The third research question evaluated the impact of the proposed techniques on the system-level behavior of the driving model. The experiments confirmed the presence of a sim-to-real gap when the driving model operated directly on raw synthetic simulation images, as the system failed to complete the scenario under any evaluated weather condition.

Under sunny conditions, the real-world baseline completed the scenario in all three runs. In contrast, raw simulation failed in all runs, reaching an average completion rate of 30%. When neural rendering techniques were applied, 3D Gaussian Splatting (3DGS) matched the real-world outcome by completing all runs. NeRF and Stable Diffusion both failed in all runs but improved the completion progress compared to raw simulation, reaching 71% and 38% average completion rates, respectively.

Under cloudy conditions, the real-world baseline again completed all runs, while raw simulation failed in all runs with only 21% average completion. With neural rendering, 3DGS and NeRF completed all runs, matching the real-world outcome, while Stable Diffusion completed two out of three runs.

In this condition, NeRF and 3DGS produced trajectories closest to the real-world baseline, with Fréchet distances of 2.13 m and 2.22 m, respectively, compared to 3.83 m for Stable Diffusion. Driving smoothness showed a similar trend, with 3DGS and NeRF achieving steering jitter values of 0.867 rad/s and 0.880 rad/s, compared to 1.780 rad/s for Stable Diffusion and 1.167 rad/s in the real-world baseline.

Under snowy conditions, the real-world baseline failed in all runs after an average completion of 47%. Raw simulation also failed in all runs but significantly earlier (25% completion), indicating a different failure behavior. When neural rendering techniques were applied, 3DGS also failed in all runs but reached a higher average completion rate of 70%, with failures occurring near the real-world location in two runs. In one run, however, the vehicle progressed further before failing, suggesting that the model may exhibit overconfidence, continuing beyond the point where the real-world system failed. NeRF and Stable Diffusion failed earlier, reaching 58% and 16% average completion, respectively.

Overall, 3D Gaussian Splatting most closely reproduces the real-world driving behavior, matching success outcomes in favorable conditions and producing the most realistic failure patterns in challenging ones. NeRF provides partial improvements, while Stable Diffusion produces the least consistent system-level behavior.

From a practical perspective, the experiments also highlighted important trade-offs between realism and computational efficiency. Training the models on the evaluated scenario required approximately 45 minutes for 3D Gaussian Splatting, compared to nearly 3 hours for NeRF and about 5 hours for Stable Diffusion. During inference, 3DGS achieved rendering speeds close to 100 FPS, whereas both NeRF and Stable Diffusion required approximately two seconds to generate a single frame. This makes 3D Gaussian Splatting particularly suitable for integration into large-scale simulation pipelines for autonomous driving testing [60].

Overall, this thesis demonstrated that real-world driving data can be transformed into executable simulation scenarios and used to analyze the sim-to-real gap in a closed-loop testing environment. By combining real-world recordings, scenario reconstruction, neural rendering, and behavioral evaluation, the proposed framework enables a more realistic and behavior-oriented comparison of neural rendering techniques. In this sense, the presented approach represents a step toward safer, more reliable, and more realistic validation methodologies for autonomous driving systems.

Bibliography

[1] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A Survey of Autonomous Driving: Common Practices and Emerging Technologies," *IEEE Access*, vol. 8, pp. 58443–58469, 2020. http://dx.doi.org/10.1109/ACCESS.2020.2983149

[2] J. Janai, F. Güney, A. Behl, and A. Geiger, "Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art," *arXiv preprint arXiv:1704.05519*, 2021. https://arxiv.org/abs/1704.05519

[3] Y. Li and J. Ibanez-Guzman, "Lidar for Autonomous Driving: The Principles, Challenges, and Trends for Automotive Lidar and Perception Systems," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 50–61, 2020. http://dx.doi.org/10.1109/MSP.2020.2973615

[4] D. E. Knuth, "The Art of Computer Programming, Volume 3: Sorting and Searching," 2nd ed. Reading, MA, USA: Addison-Wesley, 1997.

[5] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A Survey of Deep Learning Techniques for Autonomous Driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2019. http://dx.doi.org/10.1002/rob.21918

[6] K. Hussain, C. Moreira, J. Pereira, S. Jardim, and J. Jorge, "A Comprehensive Literature Review on Modular Approaches to Autonomous Driving: Deep Learning for Road and Racing Scenarios," *Smart Cities*, vol. 8, no. 3, p. 79, 2025. https://www.mdpi.com/2624-6511/8/3/79

[7] A. Gupta, A. Anpalagan, L. Guan, and A. S. Khwaja, "Deep Learning for Object Detection and Scene Perception in Self-Driving Cars: Survey, Challenges, and Open Issues," *Array*, vol. 10, p. 100057, 2021. https://doi.org/10.1016/j.array.2021.100057

[8] E. Arnold, O. Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby, and A. Mouzakitis, "A Survey on 3D Object Detection Methods for Autonomous Driving Applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 10, pp. 3782–3795, 2019. https://doi.org/10.

1109/TITS.2019.2892405

[9] S. Zheng, J. Wang, C. Rizos, W. Ding, and A. El-Mowafy, "Simultaneous Localization and Mapping (SLAM) for Autonomous Driving: Concept and Analysis," *Remote Sensing*, vol. 15, no. 4, p. 1156, 2023. https://www.mdpi.com/2072-4292/15/4/1156

[10] D. S. González, J. Pérez, V. Milanés, and F. Nashashibi, "A Review of Motion Planning Techniques for Automated Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, pp. 1135–1145, 2016. https://api.semanticscholar.org/CorpusID:11971675

[11] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles," *arXiv preprint arXiv:1604.07446*, 2016. https://arxiv.org/abs/1604.07446

[12] S. Tang, Z. Zhang, Y. Zhang, J. Zhou, Y. Guo, S. Liu, S. Guo, Y.-F. Li, L. Ma, Y. Xue, and Y. Liu, "A Survey on Automated Driving System Testing: Landscapes and Trends," *arXiv preprint arXiv:2206.05961*, 2023. https://arxiv.org/abs/2206.05961

[13] Fortiss, F. Home. *Fortiss*. (2026), https://www.fortiss.org/en/

[14] W.-H. Chen, J.-C. Wu, Y. Davydov, W.-C. Yeh, and Y.-C. Lin, "Impact of Perception Errors in Vision-Based Detection and Tracking Pipelines on Pedestrian Trajectory Prediction in Autonomous Driving Systems," *Sensors*, vol. 24, no. 15, p. 5066, 2024. https://www.mdpi.com/1424-8220/24/15/5066

[15] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing Systems for Autonomous Driving: State of the Art and Challenges," *IEEE Internet of Things Journal*, vol. 8, pp. 6469–6486, 2020. https://api.semanticscholar.org/CorpusID:222066853

[16] Baidu Apollo Team, "Apollo: Open Source Autonomous Driving," https://github.com/ApolloAuto/apollo, 2017.

[17] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems," *Proc. ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 287–296, 2018. https://api.semanticscholar.org/CorpusID:13792468

[18] H.-Y. Jung, D.-H. Paek, and S.-H. Kong, "Open-Source Autonomous Driving Software Platforms: Comparison of Autoware and Apollo," *arXiv preprint arXiv:2501.18942*, 2025. https://arxiv.org/abs/2501.18942

[19] V. M. Raju, V. Gupta, and S. Lomate, "Performance of Open Autonomous Vehicle Platforms: Autoware and Apollo," *Proc. IEEE 5th International Conference for Convergence in Technology (I2CT)*, pp. 1–5, 2019. https://api.semanticscholar.org/CorpusID:212703007

[20] S. Ochs, J. Doll, D. Grimm, T. Fleck, M. Heinrich, S. Orf, A. Schotschneider, H. Gremmelmaier, R. Polley, S. Pavlitska, M. Zipfl, H. Schneider, F. Mütsch, D. Bogdoll, F. Kuhnt, P. Schörner, M. R. Zofka, and J. M. Zöllner, "One Stack to Rule Them All: To Drive Automated Vehicles, and Reach for the 4th Level," *arXiv preprint arXiv:2404.02645*, 2024. https://arxiv.org/abs/2404.02645

[21] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to End Learning for Self-Driving Cars," *arXiv preprint arXiv:1604.07316*, 2016. https://arxiv.org/abs/1604.07316

[22] A. Tampuu, T. Matiisen, M. Semikin, D. Fishman, and N. Muhammad, "A Survey of End-to-End Driving: Architectures and Training Methods," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 4, pp. 1364–1384, 2022. http://dx.doi.org/10.1109/TNNLS.2020.3043505

[23] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-End Driving via Conditional Imitation Learning," *arXiv preprint arXiv:1710.02410*, 2018. https://arxiv.org/abs/1710.02410

[24] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, "End-to-End Autonomous Driving: Challenges and Frontiers," *arXiv preprint arXiv:2306.16927*, 2024. https://arxiv.org/abs/2306.16927

[25] A. Stocco, B. Pulfer, and P. Tonella, "Mind the Gap! A Study on the Transferability of Virtual Versus Physical-World Testing of Autonomous Driving Systems," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1928–1940, 2023. http://dx.doi.org/10.1109/TSE.2022.3202311

[26] P. Koopman and M. D. Wagner, "Autonomous Vehicle Safety: An Interdisciplinary Challenge," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, pp. 90–96, 2017. https://api.semanticscholar.org/CorpusID:7631831

[27] H. Guo, J. Li, N. Saravanan, J. Wishart, et al., "Developing an Automated Vehicle Research Platform by Integrating Autoware with the DataSpeed Drive-By-Wire System," *SAE Technical Paper 2024-01-1981*, 2024. https://doi.org/10.4271/2024-01-1981

[28] T. Kessler, J. Bernhard, M. Buechel, K. Esterle, P. Hart, D. Malovetz,

M. Truong Le, F. Diehl, T. Brunner, and A. Knoll, "Bridging the Gap Between Open Source Software and Vehicle Hardware for Autonomous Driving," *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, pp. 1612–1619, 2019. https://doi.org/10.1109/IVS.2019.8814229

[29] M. Testouri, G. Elghazaly, and R. Frank, "RoboCar: A Rapidly Deployable Open-Source Platform for Autonomous Driving Research," *arXiv preprint arXiv:2405.03572*, 2024. https://arxiv.org/abs/2405.03572

[30] L. Chen, T. Tang, Z. Cai, Y. Li, P. Wu, H. Li, J. Shi, J. Yan, and Y. Qiao, "Level 2 Autonomous Driving on a Single Device: Diving into the Devils of Openpilot," *arXiv preprint arXiv:2206.08176*, 2022. https://arxiv.org/abs/2206.08176

[31] S. Feng, H. Sun, X. Yan, H. Zhu, Z. Zou, S. Shen, and H. X. Liu, "Dense Reinforcement Learning for Safety Validation of Autonomous Vehicles," *Nature*, vol. 615, pp. 620–627, 2023. https://doi.org/10.1038/s41586-023-05732-2

[32] J. Stempel, "Tesla Sued Over Model S Crash That Killed Three in New Jersey," *Reuters*, June 2025. https://www.reuters.com/legal/litigation/tesla-sued-over-new-jersey-crash-model-s-that-killed-three-2025-06-23/

[33] D. Jamali, "Tesla Hit with $243 Million in Damages After Jury Finds Its Autopilot Feature Contributed to Fatal Crash," *NBC News*, Aug. 2025. https://www.nbcnews.com/news/us-news/tesla-autopilot-crash-trial-verdict-partly-liable-rcna222344

[34] The Guardian, "US Regulators Investigate After Waymo Self-Driving Car Struck Child Near School," Jan. 2026. https://www.theguardian.com/technology/2026/jan/29/us-regulators-investigate-waymo-struck-child

[35] F. U. Haq, D. Shin, S. Nejati, and L. C. Briand, "Can Offline Testing of Deep Neural Networks Replace Their Online Testing?: A Case Study of Automated Driving Systems," *Empirical Software Engineering*, vol. 26, no. 5, 2021. http://dx.doi.org/10.1007/s10664-021-09982-4

[36] Z. Wang and A. C. Bovik, "Mean squared error: Love it or leave it? A new look at signal fidelity measures," *IEEE Signal Processing Magazine*, vol. 26, no. 1, pp. 98–117, 2009.

[37] Y. Kang, H. Yin, and C. Berger, "Test Your Self-Driving Algorithm: An Overview of Publicly Available Driving Datasets and Virtual Testing Environments," *IEEE Transactions on Intelligent Vehicles*, vol. 4, pp. 171–185, 2019. https://api.semanticscholar.org/CorpusID:88469717

[38] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3213–3223, 2016.

[39] A. Geiger, P. Lenz, R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3354–3361, 2012.

[40] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, S. Zhao, S. Cheng, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in Perception for Autonomous Driving: Waymo Open Dataset," *arXiv preprint arXiv:1912.04838*, 2020. https://arxiv.org/abs/1912.04838

[41] H. Schafer, E. Santana, A. Haden, and R. Biasini, "A Commute in Data: The comma2k19 Dataset," *arXiv preprint arXiv:1812.05752*, 2018. https://arxiv.org/abs/1812.05752

[42] A. Stocco, B. Pulfer, and P. Tonella, "Model vs System Level Testing of Autonomous Driving Systems: A Replication and Extension Study," *Empirical Software Engineering*, vol. 28, no. 3, 2023. https://doi.org/10.1007/s10664-023-10306-x

[43] Velodyne, V. Digital Lidar sensors for Automation, Drones Robotics: Ouster. *Ouster*. (2026), https://ouster.com/

[44] M. Haklay, P. Weber, "OpenStreetMap: User-Generated Street Maps," in *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

[45] E. Salvato, G. Fenu, E. Medvet, and F. A. Pellegrino, "Crossing the Reality Gap: A Survey on Sim-to-Real Transferability of Robot Controllers in Reinforcement Learning," *IEEE Access*, vol. 9, pp. 153171–153187, 2021. https://doi.org/10.1109/ACCESS.2021.3126658
latex

[46] S. C. Lambertenghi, M. Flores Valdez, and A. Stocco, "A Multi-Modality Evaluation of the Reality Gap in Autonomous Driving Systems," *arXiv preprint arXiv:2509.22379*, 2025. https://arxiv.org/abs/2509.22379

[47] Providentia, P. Providentia. *Fortiss*. (2026), https://www.fortiss.org/en/research/projects/detail/providentia

[48] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," *arXiv preprint arXiv:1711.03938*, 2017.

[49] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lember, M. Moužannar, S. Pang, D. Cobber, O. Colber, and others, "LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving," *Proc. IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–6, 2020. https://arxiv.org/abs/2005.03778

[50] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-Art Natural Language Processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (EMNLP)*, pp. 38–45, 2020.

[51] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," *Field and Service Robotics*, pp. 621–635, 2018. https://arxiv.org/abs/1705.05065

[52] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger, "TransFuser: Imitation with Transformer-Based Sensor Fusion for Autonomous Driving," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 11, pp. 12878–12895, 2022. https://arxiv.org/abs/2205.15997

[53] J. Quiñonero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence, "Dataset Shift in Machine Learning," *MIT Press*, 2009. https://api.semanticscholar.org/CorpusID:61294087

[54] M. Schwonberg, J. Niemeijer, J.-A. Termöhlen, J. P. Schäfer, N. M. Schmidt, H. Gottschalk, and T. Fingscheidt, "Survey on Unsupervised Domain Adaptation for Semantic Segmentation for Visual Perception in Automated Driving," *arXiv preprint arXiv:2304.11928*, 2023. https://arxiv.org/abs/2304.11928

[55] X. Hu, S. Li, T. Huang, B. Tang, R. Huai, and L. Chen, "How Simulation Helps Autonomous Driving: A Survey of Sim2real, Digital Twins, and Parallel Intelligence," *IEEE Transactions on Intelligent Vehicles*, vol. 9, no. 1, pp. 593–612, 2023. https://arxiv.org/abs/2305.01263

[56] M. E. Atik, "Comparative Assessment of Neural Radiance Fields and 3D Gaussian Splatting for Point Cloud Generation from UAV Imagery," *Sensors*, vol. 25, no. 10, art. 2995, 2025. https://www.mdpi.com/1424-8220/25/10/2995

[57] H. Zhu, Z. Zhang, J. Zhao, H. Duan, D. Yao, X. Xiao, and J. Yuan, "Scene Reconstruction Techniques for Autonomous Driving: A Review of 3D Gaussian Splatting," *Artificial Intelligence Review*, vol. 58, art. 30,

2024. https://doi.org/10.1007/s10462-024-10955-4

[58] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[59] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.

[60] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3D Gaussian Splatting for Real-Time Radiance Field Rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, 2023.

[61] ASAM e.V., "OpenDRIVE – Open Dynamic Road Information for Vehicle Environment," https://www.asam.net/standards/detail/opendrive/.

[62] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-Resolution Image Synthesis with Latent Diffusion Models," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10684–10695, 2022. https://arxiv.org/abs/2112.10752

[63] J. Ho, A. Jain, and P. Abbeel, "Denoising Diffusion Probabilistic Models," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020.

[64] K. Winter, A. Vivekanandan, R. Polley, Y. Shen, C. Schlauch, M.-K. Bouzidi, B. Derajic, N. Grabowsky, A. Mariani, D. Rochau, G. Lucente, H. Yadav, F. Mualla, A. Molin, S. Bernhard, C. Wirth, Ö. Ş. Taş, N. Klein, F. B. Flohr, and H. Gottschalk, "Generative AI for Autonomous Driving: A Review," *arXiv preprint arXiv:2505.15863*, 2025. https://arxiv.org/abs/2505.15863

[65] Y. Wang, S. Xing, C. Can, R. Li, H. Hua, K. Tian, Z. Mo, X. Gao, et al., "Generative AI for Autonomous Driving: Frontiers and Opportunities," *arXiv preprint arXiv:2505.08854*, 2025. https://arxiv.org/abs/2505.08854

[66] M. Tancik, V. Casser, X. Yan, S. Pradhan, B. Mildenhall, P. P. Srinivasan, J. T. Barron, H. Kretzschmar, "Block-NeRF: Scalable Large Scene Neural View Synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8248–8258, 2022.

[67] H. Turki, D. Ramanan, M. Satyanarayanan, "Mega-NeRF: Scalable Construction of Large-Scale NeRFs for Virtual Fly-Throughs," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12922–12931, 2022.

[68] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *arXiv preprint arXiv:1312.6114*, 2022.

[69] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *arXiv preprint arXiv:1505.04597*, 2015. https://arxiv.org/abs/1505.04597

[70] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning Transferable Visual Models From Natural Language Supervision," *arXiv preprint arXiv:2103.00020*, 2021. https://arxiv.org/abs/2103.00020

[71] L. Zhang, A. Rao, and M. Agrawala, "Adding Conditional Control to Text-to-Image Diffusion Models," *arXiv preprint arXiv:2302.05543*, 2023. https://arxiv.org/abs/2302.05543

[72] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323, pp. 533–536, 1986. https://api.semanticscholar.org/CorpusID:205001834

[73] National Imagery and Mapping Agency, "Department of Defense World Geodetic System 1984, Its Definition and Relationships with Local Geodetic Systems," NIMA Technical Report TR8350.2, Third Edition, 2000.

[74] J. L. Schönberger and J.-M. Frahm, "Structure-from-Motion Revisited," *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[75] J. Pahk, J. Shim, M. Baek, Y. Lim, and G. Choi, "Effects of Sim2Real Image Translation via DCLGAN on Lane Keeping Assist System in CARLA Simulator," *IEEE Access*, vol. 11, pp. 33915–33927, 2023. https://doi.org/10.1109/ACCESS.2023.3262991

[76] L. Baresi, D. Y. Xian Hu, A. Stocco, and P. Tonella, "Efficient Domain Augmentation for Autonomous Driving Testing Using Diffusion Models," *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 398–410, 2025. https://arxiv.org/abs/2409.13661

[77] Y. Zhou, M. Simon, Z. Peng, S. Mo, H. Zhu, M. Guo, and B. Zhou, "SimGen: Simulator-Conditioned Driving Scene Generation," *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. https://arxiv.org/abs/2406.09386

[78] NVIDIA et al., "Cosmos World Foundation Model Platform for Physical AI," *arXiv preprint arXiv:2501.03575*, 2025. https://arxiv.org/abs/2501.03575

[79] Z. Yang, Y. Chen, J. Wang, S. Manivasagam, W.-C. Ma, A. J. Yang,

and R. Urtasun, "UniSim: A Neural Closed-Loop Sensor Simulator," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1389–1399, 2023. https://arxiv.org/abs/2308.01898

[80] A. Tonderski, C. Lindström, G. Hess, W. Ljungbergh, L. Svensson, and C. Petersson, "NeuRAD: Neural Rendering for Autonomous Driving," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14895–14904, 2024. https://arxiv.org/abs/2311.15260

[81] Y. Yan, H. Lin, C. Zhou, W. Wang, H. Sun, K. Zhan, X. Lang, X. Zhou, and S. Peng, "Street Gaussians for Modeling Dynamic Urban Scenes," *Proceedings of the European Conference on Computer Vision (ECCV)*, 2024. https://arxiv.org/abs/2401.01339

[82] M. Tancik, E. Weber, E. Ng, R. Li, B. Yi, T. Wang, A. Kristoffersen, J. Austin, K. Salahi, A. Ahuja, D. Mcallister, J. Kerr, and A. Kanazawa, "Nerfstudio: A Modular Framework for Neural Radiance Field Development," *ACM SIGGRAPH Conference Proceedings*, 2023. http://dx.doi.org/10.1145/3588432.3591516

[83] T. Wang, X. Zhu, J. Pang, and D. Lin, "FCOS3D: Fully Convolutional One-Stage Monocular 3D Object Detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pp. 913–922, 2021.

[84] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, *et al.*, "nuScenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11621–11631, 2020.

[85] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," https://github.com/ultralytics/ultralytics, 2023.

[86] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, O. Beijbom, "PointPillars: Fast Encoders for Object Detection from Point Clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12697–12705, 2019.

[87] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[88] J. P. Snyder, "Map Projections—A Working Manual," *U.S. Geological Survey Professional Paper 1395*, U.S. Government Printing Office, Washington, DC, 1987.

[89] W. R. Hamilton, "On a new species of imaginary quantities connected

with a theory of quaternions," in *Proceedings of the Royal Irish Academy*, vol. 2, pp. 424–434, 1844.

[90] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2022.

[91] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, P. Luo, "SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers," in *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 12077–12090, 2021.

[92] C. Rowles, "TemporalNet: Temporal Consistency for Stable Diffusion," https://huggingface.co/CiaraRowles/TemporalNet, 2023.

[93] M. Buechel, M. Schellmann, H. Rosier, T. Kessler, and A. Knoll, "Fortuna: Presenting the 5G-Connected Automated Vehicle Prototype of the Project PROVIDENTIA," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[94] S. C. Lambertenghi and A. Stocco, "Assessing Quality Metrics for Neural Reality Gap Input Mitigation in Autonomous Driving Testing," *arXiv preprint arXiv:2404.18577*, 2024.

[95] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," 2016.

[96] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[97] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," *CoRR*, vol. abs/1603.08155, 2016.

[98] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local Nash equilibrium," 2018.

[99] M. Fréchet, "Sur la distance de deux lois de probabilité," *Annales de l'ISUP*, vol. VI, no. 3, pp. 183–198, 1957.

[100] J. Yu, X. Xu, F. Gao, S. Shi, M. Wang, D. Tao, and Q. Huang, "Towards realistic face photo-sketch synthesis via composition-aided GANs," 2020.

[101] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[102] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *Journal of Machine Learning Research*, vol. 13, pp. 723–773, 2012.

[103] M. F. Naeem, S. J. Oh, Y. Uh, Y. Choi, and J. Yoo, "Reliable Fidelity and Diversity Metrics for Generative Models," in *Proc. International Conference on Machine Learning (ICML)*, 2020.

[104] iMAR Navigation GmbH, "Tasks in Navigation, Localization, Guidance, Control, Surveying, Integration Engineering," iMAR Navigation, St. Ingbert, Germany. [Online]. Available: https://imar-navigation.de/en/component/zoo/item/tasks-in-navigation-localization-guidance-control-surveying-integration-e

[105] T. Cover and P. Hart, "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[106] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8," https://github.com/ultralytics/ultralytics, 2023.

[107] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common Objects in Context," *arXiv preprint arXiv:1405.0312*, 2015.

[108] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Y. Zeevi, "The Farthest Point Strategy for Progressive Image Sampling," *IEEE Transactions on Image Processing*, vol. 6, no. 9, pp. 1305–1315, 1997.

[109] M. J. Chong and D. Forsyth, "Effectively Unbiased FID and Inception Score and Where to Find Them," *arXiv preprint arXiv:1911.07023*, 2020.

[110] S. Jayasumana, S. Ramalingam, A. Veit, D. Glasner, A. Chakrabarti, and S. Kumar, "Rethinking FID: Towards a Better Evaluation Metric for Image Generation," *arXiv preprint arXiv:2401.09603*, 2024.

[111] Y. Zhao, Y. Wang, and others, "Exploring Generative AI for Sim2Real in Driving Data Synthesis," *arXiv preprint arXiv:2404.09111*, 2024.

[112] C. Vachha and A. Haque, "Instruct-GS2GS: Editing 3D Gaussian Splats with Instructions," 2024. https://instruct-gs2gs.github.io/

[113] A. Haque, M. Tancik, A. Efros, A. Holynski, and A. Kanazawa, "Instruct-NeRF2NeRF: Editing 3D Scenes with Instructions," *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.