EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Mathematisch-Naturwissenschaftliche Fakultät

Lernbasierte Computer Vision

Masterarbeit

Tackling CARLA Leaderboard 2.0 with End-to-End Imitation Learning

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik
Lernbasierte Computer Vision
Julian Zimmerlin, `julian.zimmerlin@uni-tuebingen.de`, 2024

Bearbeitungszeitraum: 15.03.2024-16.09.2024

Betreuer/Gutachter: Prof. Dr. Andreas Geiger, Universität Tübingen
Zweitgutachter: Prof. Dr. Georg Martius, Universität Tübingen

Selbstständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Masterarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Julian Zimmerlin (Matrikelnummer 6009977), September 26, 2024

Abstract

We tackle the CARLA Leaderboard 2.0, a novel benchmark for autonomous vehicles that involves solving 38 complex traffic scenarios in simulation, such as navigating in intersections, avoiding road obstacles, and high-speed highway driving. Using end-to-end imitation learning, we train a model that directly maps sensor inputs to driving signals by imitating a privileged expert system. Building on an existing architecture, we refine key aspects of the machine learning pipeline, involving data collection, model training and evaluation, to adapt to the new benchmark. We highlight the impact of expert driving style and frame importance, two understudied aspects of imitation learning, on downstream model performance. We also uncover a design flaw in the leaderboard's evaluation metrics, which unintentionally encourages premature termination of evaluation routes, and propose a solution for future challenges. By applying our insights, we develop the top-performing open-source model for CARLA Leaderboard 2.0, ranking second overall in the 2024 CARLA Autonomous Driving Challenge.

Acknowledgments

I want to thank Kashyap Chitta for providing guidance and useful advice throughout this project. I want to thank my teammate in the challenge Jens Beißwenger, for his work on the expert driver and many insightful discussions. Further, I want to thank Katrin Renz and Berhard Jaeger for the insights and assistance provided and Andreas Geiger for giving me the opportunity to write this thesis.

Contents

1 Introduction

According to the World Health Organization, road accidents result in approximately 1.19 million deaths worldwide each year and are the leading cause of death for children and young adults aged 5–29 [Org23]. Autonomous vehicles have the potential to significantly reduce traffic-related fatalities by eliminating human error, a major contributor to these incidents. Additionally, self-driving cars could improve traffic efficiency by minimizing congestion and idle time, lowering the environmental impact of transportation. They also promise increased accessibility for elderly and disabled individuals as well as an improvement in overall quality of life by allowing commuters to use travel time for work or leisure.

Despite these promising benefits, widespread deployment of autonomous vehicles depends on their ability to operate safely and reliably across diverse weather conditions and traffic scenarios. Addressing this challenge, the CARLA Leaderboard 2.0 provides a novel simulation-based evaluation framework, designed to test autonomous systems with a variety of complex tasks. These tasks include swerving into opposite traffic to avoid obstacles, safely navigating at highways speeds, merging into slow and fast traffic, negotiating at intersections, and reacting to dynamic actors like pedestrians and cyclists.

This thesis consitutes a first attempt to tackle the CARLA Leaderboard 2.0 evaluation benchmark using end-to-end imitation learning (IL), a learning paradigm that has seen success in many fields, such as robotics and video games. In this approach, a machine learning model learns to imitate behavioral demonstrations provided by humans or an expert system. What makes our task particularly challenging is that the agent often needs to make split-second decisions in the new scenarios provided by Leaderboard 2.0. This means that the crucial frames necessary to solve the new scenarios form only a tiny fraction of the overall dataset. Thus, even when using an expert system to collect a large training dataset in an automated manner, we still encounter a low-data regime in many crucial situations.

The main contributions of our work are the following:

- We rethink all stages of the machine learning pipeline – data collection, training, and evaluation – on the new CARLA Leaderboard 2.0 benchmark, distilling the minimal adaptations required at each stage to optimize the performance of an existing state-of-the-art architecture, TransFuser++ [JCG23].
- We highlight the impact of two understudied aspects, expert driving style

and frame importance, on the performance of imitation learning models in autonomous driving.

- We develop a competitive open-source model that ranks first in the MAP track and second in the SENSORS track of the 2024 CARLA Autonomous Driving Challenge. We also include ablation studies and qualitative analyses of failure modes to offer a solid foundation for future research on CARLA Leaderboard 2.0.
- We theoretically demonstrate how the performance metrics used in the official leaderboard inadvertently encourage participants to terminate evaluation routes prematurely to achieve competitive driving scores, and propose changes to the metrics that can solve this problem for future challenges.

The content of this thesis is structured as follows. In Chapter 2, we provide an overview over related research on imitation learning with a focus on autonomous driving. In Chapter 3, we explain the main components of our project from a technical perspective, including the CARLA Leaderboard 2.0 and our end-to-end imitation learning approach. In Chapter 4, we provide details about the evaluation benchmarks and datasets we create as part of this project. In Chapter 5, we present our experimental results, including the official leaderboard results and a range of internal ablations. Finally, in Chapter 6, we discuss the implications of our findings, providing insights into the design of performance metrics and our model's failure modes.

2 Related Work

This chapter presents an overview of relevant research, starting with an introduction to end-to-end self-driving and its advantages over traditional modular architectures. We discuss imitation learning as a scalable and effective method for training agents to navigate complex environments by imitating an expert. Finally, we situate IL within the context of autonomous driving, covering datasets, trends in output representations for driving and the common issue of covariate shift.

2.1 End-to-End Self-Driving

Most self-driving solutions in use today consist of modular pipelines. As detailed by [BGC+19], these pipelines typically include a range of subsystems that work together to drive the vehicle. Early modules are responsible for perceiving the environment using sensors like cameras or LiDAR, producing explicit intermediate representations that may include road layouts, information about nearby dynamic actors, the state of traffic lights, and more. Additional modules forecast the future states of the ego vehicle and other dynamic actors such as pedestrians and vehicles. Based on the outputs of these modules, planning modules determine the future path of the ego vehicle, for example in the form of steering commands or waypoints (lateral control). The pipeline also predicts braking and acceleration signals (longitudinal control), which consider the behavior of other agents and the overall traffic situation. Commonly, such modular pipelines use sophisticated rule-based algorithms to translate environmental perception into driving signals [AK21, XMC+21].

While these systems offer strong explainability through examination of the intermediate output representations, they come with several drawbacks. A major challenge is handling uncertainty - uncertainty information from upstream modules is either discarded or needs to be explicitly managed by downstream modules, complicating their design [MGK+17]. Additionally, error propagation is a significant issue [YLCT20] - mispredictions in early modules can lead to catastrophic erros in downstream modules, as was the case in one of the first known fatal accidents involving an autonomous vehicle in 2016, where the Tesla autopilot failed to detect a white truck against the background of a brightly lit sky [TPJR18]. Finally, for rule-based modules, it is difficult to account for all possible edge cases in real-world driving situations. Without careful design, these problems can negatively affect the overall reliability and performance of the autonomous vehicle.

In recent years, an alternative paradigm has gained increasing attention in the self-driving community. End-to-end self-driving [CWC+24, TMS+22] involves training neural networks that take sensor information as inputs and directly produce driving outputs without relying on explicit intermediate representations. This approach can simplify the architecture of self-driving models and reduce the number of failure points within the system. In addition, it allows learning from larger datasets [CBL+20, GKM+20], as data mapping sensory inputs to driving output can be produced simply by driving (either by a human or by an expert system in a simulator), whereas previously, it may have been necessary to collect expensive human annotations for the intermediate representations. However, training an end-to-end model that robustly generalizes to a wide variety of complex driving scenarios in different environments and weather conditions remains a significant challenge and active area of research [CWC+24]. Due to the absence of hard-coded safety measures in end-to-end systems, it is especially crucial to enable such models to safely manage rare "long-tail" situations, which may not be part of the training data. With our end-to-end approach to solving the CARLA Leaderboard 2.0, we aim to contribute to the development of end-to-end autonomous vehicles that are reliable and safe for widespread use.

2.2 Imitation Learning

Imitation learning (IL) is a learning paradigm where an agent (usually a deep neural network) learns to operate in an environment by mimicking the behavior of an expert. This expert, which may be either a human or an automated system (expert) using privileged information not accessible to the IL agent, provides demonstrations of the desired actions. The IL agent then learns to map its sensory inputs to the expert's actions as accurately as possible. This approach sets it apart from the field of *reinforcement learning* (RL), where agents learn through trial and error and optimize a numerical reward function that defines the goal of the task, without relying on explicit demonstrations. In many complex tasks, including autonomous driving, where creating an appropriate reward function is difficult and errors during learning could be costly or dangerous, imitation learning presents itself as an appealing alternative to reward-based RL [MYDM22].

The most straightforward form of imitation learning is *behavior cloning* (BC) [BS95], where the agent is trained in a supervised fashion on a large, fixed dataset of expert demonstrations. In this approach, the inputs are sensory data and the outputs are the actions taken by the expert. This is the strategy that we use in this work, and its main strength lies in its simplicity: BC works completely *offline*, i.e., it does not require rolling out the learned policy or interacting with the environment (simulator) during training.

Still, the behavior cloning setting differs from classical supervised learning in several ways. In classical supervised learning, it is assumed that samples are independent and identically distributed (i.i.d.) during both training and testing. In contrast, BC involves training samples that are highly correlated, as they are sequential frames from the expert's trajectory. Additionally, during inference, prediction errors affect the agent's future states, potentially pushing it out of the training distribution and leading to increasingly inaccurate predictions. This problem, known as *covariate shift*, can thus result in accumulating errors and is a frequent cause of failure for imitation learning models [CUS+21, KWB+21].

Several other IL methods have been developed, including *inverse reinforcement learning* (IRL) [AD21], which aims to infer the underlying reward function that the expert is implicitly optimizing and then uses this inferred reward function to guide learning in a reinforcement learning framework. The learned reward function can help the model generalize across a wider range of states learned, mitigating covariate shift. *Generative adversarial imitation learning* (GAIL) [HE16], on the other hand, combines adversarial training with imitation learning, using a discriminator to differentiate between expert and agent-generated behavior. By continually improving the agent's policy through adversarial training, GAIL promises improved resilience to variations in state distributions compared to classical behavioral cloning. Despite the potential advantages offered by IRL and GAIL, their training processes are considerably more complex and can be difficult to stabilize, particularly with large models like ours. Therefore, we opt for behavior cloning, which allows us to leverage the benefits of a simpler, highly parallelizable training approach at a large model and dataset scale.

2.3 Imitation Learning in Autonomous Driving

In recent years, the combination of end-to-end models with imitation learning has become a dominant paradigm in autonomous driving research. This popularity mainly stems from the scalability and effectiveness of such models, particularly in leveraging large-scale driving datasets that have recently become available.

Datasets used for IL in autonomous driving are typically generated either by collecting data from real-world vehicles equipped with various sensors or within simulators. Some of the most widely used real-world datasets include the Waymo Open Dataset [SKD+20], the Audi Autonomous Driving Dataset [GKM+20], nuScenes [CBL+20], nuPlan [CKT+22], and Argoverse 2 [WQA+21]. Simulated datasets, such as those created using CARLA [DRC+17], offer an alternative approach. This approach inevitably comes with the drawback of introducing domain shift compared to the real world, as simulators cannot fully capture the complexity of real-world driving. Additionally, researchers often generate their own datasets, which can hinder comparability across studies. Even when using the same evaluation benchmarks, differences in results may stem from variations in datasets rather than methods, creating a potential confounding factor. Despite these drawbacks, simulated datasets are significantly easier and more cost-effective to generate compared to real-world datasets, enabling the creation of larger-scale datasets. Additionally, simulators provide researchers with full control over key parameters such as sensor configurations, expert driving behavior, routes, traffic scenario distributions, weather conditions, and other variables of interest – factors that cannot be manipulated when working with fixed real-world datasets.

IL methods have also achieved notable success in CARLA, with many top-ranking submissions to the CARLA Leaderboard 1.0 (see Section 3.1.1) utilizing end-to-end models trained with this approach [CPJ$^+$22, JCG23, SWC$^+$22, CK22]. A key trend that has emerged in recent years is the change in output representations for IL models. Initially, most researches used direct control outputs in the form of steering and acceleration values. This may, however, produce inconsistent control values in sucessive frames and requires models to specialize to the precise vehicle dynamics, reducing generalizability [CWC$^+$24]. Consequently, the community transitioned to predicting waypoints – future positions of the vehicle at fixed time intervals (e.g., every 0.5 seconds). This representation combines lateral (path) and longitudinal (acceleration) control in a single target but requires additional mechanisms, such as PID controllers, to convert the predictions into actionable control signals for the vehicle.

Recently, [JCG23] introduced a disentangled representation, which separates lateral and longitudinal control into two labels: Lateral control is represented by checkpoints, which are future vehicle positions spaced equally by *distance* (e.g., 1 meter apart), while longitudinal control is expressed through one-hot encoded vectors representing target speed classes. This representation has the benefits of providing path supervision even when the vehicle is stationary (where waypoints would collapse into a single point), and improving interpretability by making it easier to diagnose whether a model failure is due to path or speed prediction errors. This representation is adopted in this work.

To address covariate shift in the domain of autonomous driving, there have been attempts to augment behavior cloning with on-policy data in the training loop. In particular, DAgger [RGB11] is a method that involves querying the expert for corrections from failure states encountered during the model's policy rollout, and then continuing to train the model on this additional data. This method has been successfully applied to improve model robustness in states that are usually not reached by the expert policy [ZC17, PBOB$^+$20]. However, to maintain simplicity in the training process, this work focuses on BC alone, addressing covariate shift by applying data augmentation techniques to the sensory inputs, which also help the model learn corrective behaviors, as will be discussed in subsequent chapters.

3 Background & Methods

This chapter outlines the methods used in our approach and provides the necessary background. We start by introducing the CARLA simulator for autonomous driving research and the CARLA Leaderboard 2.0, covering the different types of scenarios that agents must navigate and the metrics used to evaluate their performance. Additionally, the chapter explains how the privileged expert driver we use to collect our datasets operates. It also details the architecture and training routine employed for our IL agent, highlighting the adjustments made to meet the challenges of Leaderboard 2.0.

3.1 CARLA Leaderboard 2.0

This section introduces the CARLA simulator for autonomous driving research and describes the CARLA Leaderboard 2.0 challenge in which we partake.

3.1.1 CARLA Simulator

CARLA [DRC+17] is an open-source simulator for autonomous driving research, developed with the support of the Computer Vision Centre in Barcelona, Spain, and the Embodied AI Foundation in California. Built on the Unreal Engine [Gam], CARLA offers detailed graphics and a range of free digital assets, including urban layouts, buildings, and various vehicle models, allowing developers to build varied and realistic custom environments. It features numerous weather settings, combining elements such as rain, cloudy weather, clear skies, sunset, and night. The simulator also supports flexible sensor configurations and provides complete control over static and dynamic actors, tools for map generation, and more.

Furthermore, CARLA includes a Python API, which enables researchers to test both privileged agents (expert drivers) and sensor-based agents. It also provides automatic tools for benchmarking performance, such as collision detection, measurements of traveled distance, and detection of traffic rule violations. Additionally, CARLA can render supplementary data modalities, such as semantic segmentation or depth maps, which are important for developing machine learning-based autonomous driving systems.

Leaderboard

The CARLA Autonomous Driving Leaderboard [Lea] is an open competition for participants from both academia and industry, with the objective of evaluating the driving performance of autonomous agents in realistic traffic conditions. The leaderboard features two tracks: SENSORS and MAP. In the SENSORS track, agents are allowed to use up to 8 RGB cameras, 2 LIDAR sensors, 4 RADARs, 1 GNSS sensor, 1 Inertial Measurement Unit (IMU), and a speedometer. The MAP track includes all the sensors from the SENSORS track, and an additional high-definition (HD) map in OpenDRIVE format. This map provides the road network, including lanes and turns, to the agent as a network of dense waypoints. Thus, even though our model is developed for the SENSORS track without usage of the HD map, it can also be submitted to the MAP track. In addition to the sensor data, the agent receives route information through *target points* (spaced up to 200 meters apart) indicating which turns to take and roads to follow.

In the previous version of the leaderboard (CARLA Leaderboard 1.0), participants were tasked with driving secret test routes of 1-2 kilometers, facing six traffic scenarios. The competition has received significant interest from the autonomous driving community, with 36 public entries on the SENSORS track of the leaderboard to date, and several technical advancements pushing leaderboard scores. Notable innovations include advanced sensor fusion techniques (TransFuser [CPJ+22], InterFuser [SWC+22]), a strong expert driver trained via reinforcement learning (TCP [WJC+22]) and a novel training method that involves predicting trajectories of other (non-ego) vehicles (LAV [CK22]). As a consequence, scores have dramatically improved since the opening of the leaderboard in 2020, rising from under 10DS to nearly 80DS in 2023 (with the maximum being 100DS). As scores approach saturation in this benchmark, the CARLA team has introduced CARLA Leaderboard 2.0, which poses a new and much more difficult challenge.

The CARLA release for Leaderboard 2.0 introduces three new large maps with a size of 10 square kilometers each. Two of these maps, Town12 and Town13, serve as a training and validation pair and are publicly available, while the third map, Town14, is a secret test town. These maps encompass a variety of environments, such as urban city centers, suburban areas, and rural country roads. On these new maps, the driving task is particularly challenging due to the diverse environments that require agents to adapt to different traffic patterns, the need to navigate safely on highways with speed limits of up to 120 km/h, and the need to handle 38 new, complex scenarios described in Section 3.1.2. Unlike Leaderboard 1.0, the test routes in Leaderboard 2.0 are significantly longer, averaging about 10 kilometers, requiring agents to solve many more scenarios in sequence. As a consequence, leaderboard scores have dropped back into the single digits, and it remains to be seen whether this benchmark will see the same rapid improvement as Leaderboard 1.0 did in the coming years.

3.1.2 Scenario Types

In CARLA Leaderboard 2.0, agents need to solve 38 challenging scenarios, which are classified into 6 categories by the organizers [Lea]:

- **Control loss (1 scenario):** The ego vehicle must recover from a loss of control caused by bad road conditions.
- **Traffic negotiation (10 scenarios):** These scenarios take place on intersections and involve negotiating with other dynamic actors, for example by performing unprotected turns through oncoming traffic, negotiating unsignalized intersections, yielding to bicycles, or avoiding collisions with emergency vehicles running red lights. Some of these scenario types come with different variants (right turn or left turn, signalized or unsignalized junction), for a total number of 10 scenarios.
- **Highway (8 scenarios):** Scenarios related to driving on highways. Examples include merging into moving traffic from an on-ramp, avoiding collisions with vehicles cutting into the ego's lane from an adjacent lane of static traffic, and crossing a lane of moving traffic to exit the highway.
- **Obstacle avoidance (10 scenarios):** The ego vehicle must navigate around various obstacles, including stationary or slow-moving hazards, vehicles opening doors, or oncoming vehicles invading its lane. These scenarios pose unique challenges since they require leaving the ego's original lane, which requires precise timing and accurate judgement of distance to other vehicles and their velocities in order to find adequate gaps in adjacent lanes.
- **Braking and lane changing (8 scenarios):** Includes scenarios that require the ego vehicle to perform sudden braking maneuvers, for example due to sudden deceleration of the leading vehicle or a pedestrian emerging from behind a parked vehicle. Also includes scenarios where an obstacle (e.g., a pedetrian) appears while the ego is performing another maneuver such as a turn.
- **Parking Exit (1 scenario):** The ego vehicle must safely enter the flow of traffic from a parking bay.

The assignment of specific scenario names to these groups is shown in Figure 6.3.

3.1.3 Evaluation Metrics

In CARLA Leaderboard 2.0, the three primary metrics used to evaluate agents' performance are *driving score* (DS), *route completion* (RC) and *infraction score* (IS).

- **Driving score (DS)** is the most important metric, since it is used to determine the ranking of different entries to the leaderboard. It ranges from 0 to 100 and is defined the product of the other two main metrics: $DS = RC \times IS$.

- **Route completion (RC)** measures the portion of the route completed by the agent on a scale of 0 to 100. The score only includes the distance covered while the vehicle remains on the road; any distance traveled off the road is excluded from the calculation.

- The **infraction score (IS)** ranges from 0 to 1 and measures any violations of traffic rules in a multiplicative manner:

$$IS = 0.5^{\#Ped} * 0.6^{\#Veh} * 0.65^{\#Stat} * 0.7^{\#Red} * 0.7^{\#Yie} * 0.7^{\#Sce} * 0.8^{\#Sto}$$

  The individual infraction types, ordered from most to least severe, are:

  - *Ped:* Collision with a pedestrian, penalty factor 0.5.
  - *Veh:* Collision with a vehicle, penalty factor 0.6.
  - *Stat:* Collision with static layout (e.g., guardrails), penalty factor 0.65.
  - *Red:* Running a red light, penalty factor 0.7.
  - *Yie:* Failure to yield to an emergency vehicle, penalty factor 0.7
  - *Sce:* Scenarios that can block the ego vehicle indefinitely have a timeout of four minutes, after which the ego vehicle will be released to continue the route. However, in this case a penalty factor of 0.7 will be applied.
  - *MinSp:* The ego vehicle is expected to keep a minimum speed calculated from the speed of surrounding traffic. A failure to maintain this speed will lead to a penalty factor between 1 and 0.7. To simplify the presentation, this infraction type is not included in the formula above. Furthermore, we exclude this infraction type from IS calculation in evaluations on the *Town13 Short* benchmark, since incurring this infraction is often unavoidable on short routes.
  - *Sto:* Running a stop sign, penalty 0.8.
  - *Blo:* If an agent doesn't take any actions for 180 simulation seconds, for instance because it is blocked by another vehicle, the simulation will end. This will not produce a penalty in the infraction score (but will of course prevent the ego from completing the route).

  It is important to note that infraction score (IS) is calculated simply using the *absolute number of occurences* (which we denote with #) of the respective infraction as exponents in the formula above, whereas when we report statistics on infraction types in our result tables, we report the average number of occurences *per kilometer driven*. This normalization facilitates comparisons of infraction probabilities across benchmarks with different route lengths.

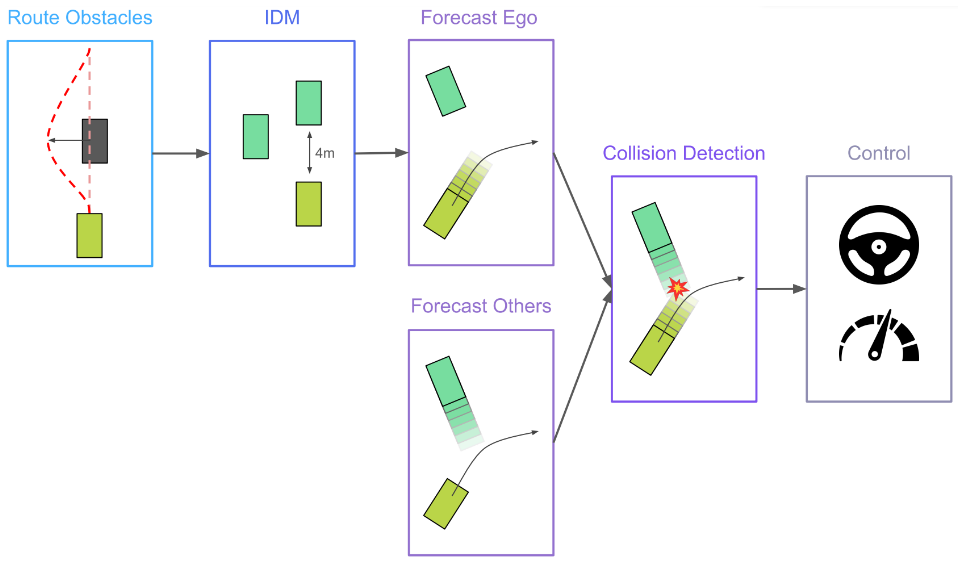


Figure 3.1: **PDM-Lite** [Bei24]. This open-source, rule-based planner is capable of solving all 38 scenarios in the CARLA Leaderboard 2.0.

3.2 PDM-lite Expert Driver

To collect a dataset for training our imitation learning agent, we employ a modified version of PDM-lite [Bei24] as a privileged expert driver, the components of which are illustrated in Figure 3.1. PDM-lite is the first published rule-based system capable of solving all 38 scenarios of CARLA Leaderboard 2.0, achieving higher driving scores than previous expert systems and near-perfect route completion on short routes. Its open-source nature enables us to adjust expert behavior based on observations about downstream model behavior. We summarize the operating principle of this expert below, and refer the reader to the technical report for a more comprehensive description.

- **Creating a dense path:** As a first step, the expert plans a dense sequence of spatially equidistant points (spaced 10cm apart) with the A* algorithm, based on the sparse target points (spaced up to 200m apart) provided by the simulator. For the obstacle avoidance scenarios that require deviating from this path, a short section of the route where the scenario occurs is laterally shifted to an adjacent lane.

At each timestep the expert then performs the following steps to generate control outputs using its privileged knowledge about the state of the world and other dynamic actors:

- **Proposing a target speed:** The expert identifies potential "leading actors" (which may also be static objects) such as vehicles, pedestrians, red lights or stop signs in the ego vehicle's future path. For each identified actor, a target speed proposal is calculated using the Intelligent Driver Model (IDM) [THH00], a mathematical model that determines an appropriate speed for the ego vehicle based on the leading actor's speed, distance, and other parameters. The minimum of these proposals is selected as the final target speed proposal. In free traffic conditions, where no leading actor is present, the expert defaults to using 72% of the speed limit as the target speed proposal, which is empirically sufficient to avoid incurring *MinSpeed* infractions.

- **Collision checks with dynamic actors:** Since IDM only ensures a safe distance to the leading vehicle, but does not account for potential collisions with other actors that may enter the ego's path, the expert performs additional collision checks. The movements of other vehicles are forecasted using an action repeat assumption (i.e., they are expected to repeat their current steering, throttle and brake values in future time steps) combined with the kinematic bicycle model [PAdNdLF17], while pedestrians are assumed to continue at a constant speed. If a potential collision with any vehicle is predicted, the expert sets the target speed to 0 to avoid an accident.
- **Conversion to control commands:** After determining the target speed, the expert chooses control commands for lateral and longitudinal control using the target speed and the planned path. For longitudinal control, a simple linear model estimates the required throttle and brake values that adjust the vehicle's speed towards the target speed. For lateral control, a PID controller is used to minimize the difference between the vehicle's heading angle and the angle to a checkpoint located between 2.4 meters and 10.5 meters ahead on the vehicle's planned path, depending on the current speed.

3.2.1 Expert Style Adjustments

Despite the strong performance of PDM-lite in CARLA Leaderboard 2.0 scenarios demonstrated in [Bei24], it was necessary to make adjustments to train an imitation learning model with this expert:

Collision Check Bug

The original PDM-lite implementation contained a bug in its bounding box intersection tests for detecting collisions with dynamic actors. The expert's target speed would only be set to zero every second frame when a collision was predicted. In the first frame where a collision was detected, the target speed would correctly be reduced to zero. However, in the following frame, the ego vehicle's forecasted position would assume a target speed of zero, causing no collision to be detected. The target speed would then revert to its original value, resulting in alternating speed settings between zero and the original target speed.

While braking every second frame was sufficient to bring the ego vehicle to a halt, these fluctuating values were used as labels for target speed prediction in the IL loss. This led the model to average these frame-level variations, producing a moderate target speed rather than executing hard braking. Fixing this issue to consistently produce zero speed labels when braking due to a predicted collision was critical for downstream model performance, even though it did not affect the expert's own scores.

Expert style

After resolving the collision check bug, video analysis of models trained on the expert data revealed that the trained models still failed to learn strong braking reflexes when approaching objects and road targets such as pedestrians, red lights, or stop signs. The root cause of this issue was the expert's driving style, which involved early braking and a slow approach towards obstacles, rather than maintaining a moderate speed and braking only when necessary shortly before the obstacle. While this behavior did not result in infractions during expert evaluations, it led to numerous crashes and infractions in the downstream IL models.

To address this, adjustments were made to the Intelligent Driver Model (IDM) parameters governing the expert's braking behavior. The goal was to make the expert brake more abruptly and closer to objects that trigger the braking response. The specific IDM parameter changes are listed in the table below:

Parameter	**Previous Value**	**Updated Value**
Safe time headway T		
Pedestrians, traffic lights, stop signs	$0.5s$	$0.1s$
Road obstacles	$0.25s$	$0.1s$
Leading vehicles, bicycles	$0.25s$	$0.25s$
Safe distance s_0		
Pedestrians, bicycles, leading vehicles	$4m$	$4m$
Road obstacles	$0.1m$	$2m$
Stop signs	$2m$	$2m$
Traffic lights	$5m$	$6m$
Comfortable deceleration b		
if ego speed > $6.02m/s$	$20m/s^2$	$3.72m/s^2$
if ego speed < $6.02m/s$	$20m/s^2$	$8.7m/s^2$
Maximum acceleration a	$11m/s^2$	$24m/s^2$
Runge-Kutta integration t_bound	$1.0s$	$0.05s$

Table 3.1: **IDM parameter changes** to adjust expert braking behavior.

These adjustments resulted in more reactive and decisive braking behavior, improving the downstream IL model's performance, as discussed in Section 5.2.

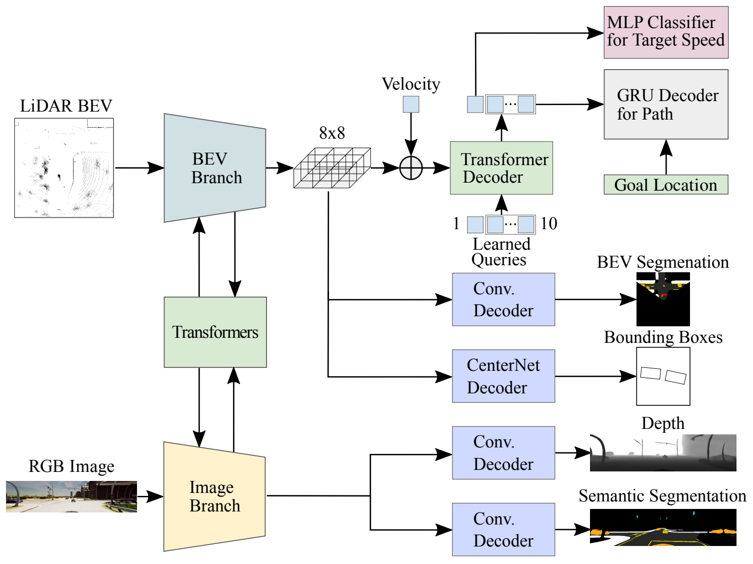


Figure 3.2: **TF++** [JCG23]. This end-to-end imitation learning approach is the best publicly available baseline for CARLA.

3.3 TransFuser++ Architecture

The architecture we use for our IL agent is based on TransFuser++ [JCG23], one of the leading open-source architectures with proven success in previous self-driving challenges in CARLA. In general, our goal in this project was to make only minimal adjustments to this architecture and its training routine, allowing us to assess what can be achieved with an existing state-of-the-art model and to identify any architectural changes necessitated by the new evaluation setting and scenarios. Here, we first describe the original TF++ implementation, and then our adjustments for the new leaderboard in the following subsections.

TF++ is a single frame model, i.e., it only takes sensor information from the current timestep as input, without any information about the past. As shown in Figure 3.2, the model inputs are a single RGB front camera image, a rasterized 360° birds-eye-view image from a LiDAR scanner mounted above the vehicle center, the current speed and a target point provided by the route ("goal location" in the figure). The outputs are a class label for one of the target speed classes (cf. Section 3.3.1 for handling of continuous target speeds) and lateral control predictions in the form of "checkpoints", i.e., points along the future path of the vehicle spaced 1 meter apart, with the first checkpoint located at a distance of about 2.5 meters in front of the vehicle center.

TF++ processes RGB and birds-eye-view LiDAR data in two separate branches, which consist of convolutional neural networks that are connected with transformer modules at multiple resolutions. The output of the BEV branch is an 8×8 feature grid, which is flattened and used as input tokens to a transformer decoder, along with the current velocity of the ego vehicle. There are 11 learned queries, which the decoder uses aggregate relevant information for target speed prediction (1 token) and checkpoint prediction (10 tokens). The target speed token is passed to a 2-layer MLP to produce the final target speed prediction. The checkpoint transformer tokens are passed one-by-one to a GRU decoder, which is initialized with the next 2D target point provided in the route, producing ten checkpoint predictions. Finally, the target speed and checkpoint predictions are used to produce steering, brake and throttle commands using a PID controller for lateral control and a simple linear model for longitudinal control, similar to the expert (cf. Section 3.2).

Auxiliary training tasks

In addition to the outputs relevant for driving, there are four additional prediction heads (depth, semantic segmentation, BEV bounding boxes, BEV semantic segmentation). Prediction of these additional data modalities serves to guide and regularize the model's perception training. From the 8×8 feature grid, a convolutional decoder predicts a birds-eye-view image, which covers a cone in front of the ego vehicle that corresponds to the area visible in the front camera, with classes such as road, lane marking, pedestrian, vehicle, red light, stop sign, and more. A CenterNet decoder [DBX+19] predicts up to 10 bounding boxes of nearby vehicles and pedestrians (including the boxes' position, extent and rotation). From the output of the image branch, convolutional decoders predict a depth view and a semantic segmentation mask for the front camera image. Note that the output of the convolutional image encoder is used exclusively for these auxiliary tasks; thus, any information from the camera image relevant to driving has to be fused with the BEV branch through the transformer modules.

Perception backbone architecture

For the perception branches (RGB and LiDAR), we use the ResNet-34 architecture [HZRS16] in our standard ("Base") model configuration. This widely used architecture consists of a 34-layer convolutional neural network and uses residual connections to improve gradient flow between layers, which allows training of deeper networks by mitigating the vanishing gradient problem. Additionally, we experiment with the larger RegNetY_032 architecture [RKG+20], which uses linearly parameterized block widths and squeeze-and-excitation blocks to adaptively reweight feature response across channels and focus on more informative features. This is the architecture we use for our leaderboard models. A performance comparison of both architectures can be found in Section 5.6. Depending on whether ResNet-34 or RegNetY_032 is used as perception architecture, our models have a total of around 64 million and 121 million parameters, respectively.

3.3.1 Adjustments for Leaderboard 2.0

Here, we describe the minimal architectural changes necessitated by the new evaluation setting and scenarios.

- **Extra speed classes.** While TF++ originally used only three target speed classes at 2m/s, 5m/s, and 8m/s (28.8km/h), in Leaderboard 2.0, the agent is required to drive at much higher speeds in interurban and highway scenarios (failure to maintain a certain minimum speed would incur an infraction). Therefore, we use 8 target speed classes at [0.0, 4.0, 8.0, 10, 13.89, 16, 17.78, 20] m/s (up to 72km/h). These classes were selected by analyzing the distribution of target speeds chosen by the expert in our dataset.

- **Two-hot labels.** PDM-Lite operates with a continuous range of target speed values, enabling it to follow leading vehicles at any speed. To solve target speed regression with TransFuser's classification module, we employ *two-hot labels* [FOV$^+$24]. This method converts a continuous value into a two-hot representation by interpolating between one-hot labels of the two nearest classes. This means that at most two (adjacent) entries of the two-hot vector will be non-zero. For instance, with our 8 speed classes ($[0.0, 4.0, 8.0, 10, 13.89, 16,$ 17.78, 20] m/s), a target speed of 3.0m/s is represented as $[0.25, 0.75, 0, 0, 0, 0, 0, 0]$.

- **Increased RGB height.** We increase the resolution of the input RGB image from 1024×256 to 1024×384, adding 128 pixels to the top of the image. This adjustment is necessary because, in some intersections in the new Leaderboard 2.0 towns, traffic lights are positioned much closer to the stop line, making them invisible at the original image height. This comes at the cost of increased GPU memory consumption due to the increased number of activations in the convolutional RGB perception branch.

- **Dynamic lookahead controller** For stable lateral control at the high speeds required by Leaderboard 2.0, it is advantageous to adjust the distance of the point selected to follow along the ego vehicle's predicted path based on the current speed. TF++ predicts a set of 10 checkpoints, each spaced 1m apart, with the first checkpoint located 2.5 meters from the vehicle center. The distance of the checkpoint to which the lateral controller minimizes the angle is determined by the formula $d = (0.097v + 0.692)$, where v is the ego's speed in km/h. We round down to the nearest available predicted checkpoint. This scaling ensures that at low speeds, the controller selects a closer point, facilitating tight turns, while at high speeds, it selects a distant point, resulting in more stable steering.

3.4 Training

In this section, we provide details about our distributed training routine, including the losses we use for the different prediction task, the optimization algorithm and its parameters.

Data parallelism

To efficiently train our self-driving model with a large dataset, we employ distributed training on four Nvidia A100 GPUs, using a Slurm compute cluster provided by the University of Tübingen. Parallel training is implemented using PyTorch's DistributedDataParallel (DDP) functionalities, which enable distribution of the model's parameters and gradients across multiple GPUs, each handling a subset of the data, while ensuring synchronized updates to the model parameters across all devices. With this setup, a typical training run takes about 2 days on four A100 GPUs.

Loss functions

We use the same loss functions as the original TransFuser++ implementation [JCG23], which are cross entropy loss for target speed classification (using two-hot labels, see Section 3.3.1) and mean absolute error for checkpoint predictions. The loss for the bounding boxes is a composite loss that including position, rotation, and the bounding box class. The other auxiliary losses are mean absolute error for depth, and mean cross entropy loss for semantics and BEV semantics, where the mean is taken across pixels (limited to a cone in front of the vehicle, which represents the area visible in the camera, in the case of BEV semantics).

Optimization

Our training routine uses the AdamW optimizer [LH19] and a batch size of 16 per GPU, 64 in total. We train for 31 epochs with a cosine annealing schedule for the learning rate [LH17], motivated by empirically observed performance improvements over a constant learning rate in early experiments. This technique aims to overcome local minima in the loss by repeatedly resetting the learning rate and annealing it to 0, resulting in bigger gradient updates after a reset and smaller updates close to the minima in the learning rate schedule.

The initial learning rate of $3 \cdot 10^{-4}$ is reduced to zero periodically with initial period length $T_0 = 1$ (i.e., the first minimum is reached after the first epoch) and $T_{mult} = 2$. T_{mult} is the multiplier applied to the period length after each reset, so that the cosinusoidal annealing will take place over twice the number of epochs in the following period. With these parameters, there are minima in the learning rate schedule after $1, 3, 7, 15, 31, \ldots$ epochs, in each of which the model parameters are saved to storage.

Single-stage vs. two-stage training

Our default training routine consists of a single training stage, including both driving-related losses (target speed, checkpoints) and auxiliary perception losses (depth, semantic segmentation, BEV bounding boxes, BEV semantic segmentation). However, prior research [CPJ+22, CK22] has shown that a staged approach, where the model is first trained exclusively on perception losses, followed by training on both driving-related and perception losses, can lead to performance improvements. Separating perception and control tasks in early epochs may prevent the model from being overwhelmed by conflicting objectives early on, enabling it to build better perception capabilites that provide a stronger basis for learning the driving task later on.

We employ two-stage training when training models geared towards leaderboard submission and provide an ablation study on its effect in Section 5.6. Specifically, we train the model for 31 epochs using only perception losses, and subsequently train for 31 more epochs with a reset of the learning rate schedule, i.e., the period length of the cosine annealing schedule is reset to one at the start of the second training stage.

Ensembling

In most of our experiments, we repeat the training process three times with different random seeds for parameter initialization and random processes in data loading, and report mean performance metrics and standard deviations over the evaluations of the three individual models. Furthermore, we experiment with using all models as an ensemble, and apply this technique in our leaderboard submissions as well. Ensembling is a widely used technique in machine learning and involves combining predictions from multiple models to improve overall performance and robustness. In our case, we simply average the predicted checkpoints and target speed prediction (after softmax) across the three models, aiming to take advantage of complementary strategies learned by the models. We study the effect of ensembling in Section 5.6.

3.4.1 Frame Importance: Target Speed Weights

One aspect of the training process to which we give particular attention is *frame importance*. We study it from two perspectives, the first being the usage of target speed weights in the classification loss. The task of target speed classification is inherently imbalanced, as the frequency of instances across speed classes varies significantly. Figure 3.3 illustrates the distribution of frames per target speed class in our dataset.

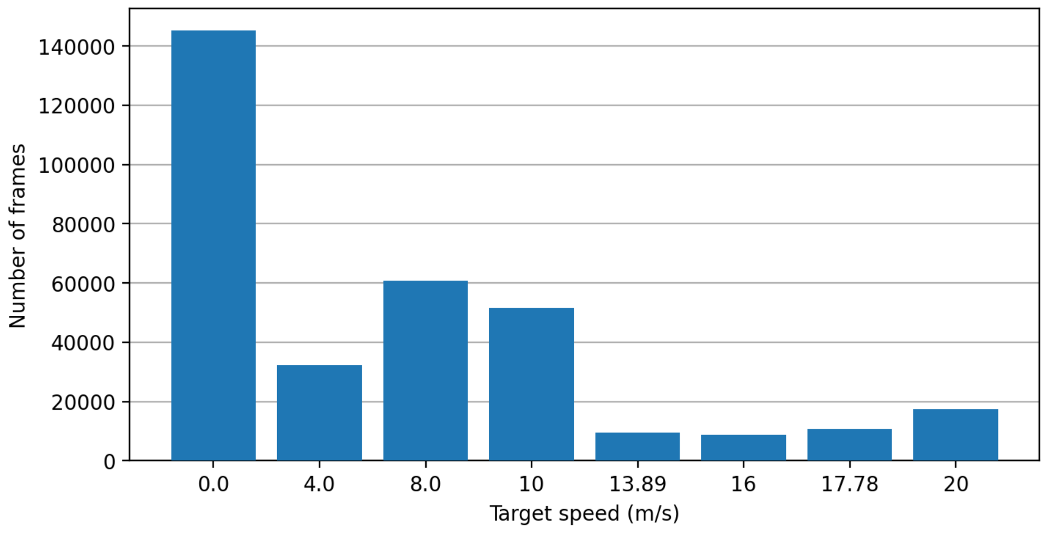


Figure 3.3: **Target speed class distribution** in our dataset after converting continuous expert target speeds to discrete two-hot labels (cf. Section 3.3.1).

speed The imbalance stems from the nature of the driving task and the varying number of routes across different road types (see Section 4.1 for more details on the routes we use for data collection). Situations where the agent is stationary (e.g., waiting at traffic lights) constitute a large portion of the dataset. As shown in the figure, the target speed class 0 dominates the dataset, while higher-speed classes, corresponding to interurban or highway driving, contain substantially fewer frames.

Class imbalance is a common challenge in classification tasks, often leading to biases towards the majority class and poor performance on minority classes. A standard solution is to assign higher weights to the underrepresented classes in the classification loss during training. This approach is employed in TransFuser++'s published code [JCG23], where class weights are calculated using the formula:

$$w_c = \frac{N}{C * N_c},$$

where N is the total number of frames, C is the number of classes, and N_c is the number of frames in class C. These weights are then used to scale the cross-entropy loss for each frame:

$$\text{CELoss}_n = -w_{c_n} \log \frac{\exp(x_{n,c_n})}{\sum_{c'=1}^{C} \exp(x_{n,c'})},$$

where c_n is the true class label for frame n, and $x_{n,c}$ is the model prediction (softmax probability) that the model assigns to class c for frame n. By scaling the class weights inversely with class frequency, this method ensures that the total weight, summed over all frames in each class, is equal for all classes. For our dataset, with $N = 337\text{k}$,

$C = 8$ and the class counts depicted in Figure 3.3, we obtain the class weights [0.29, 1.30, 0.69, 0.81, 4.43, 4.76, 3.90, 2.41] for our target speed classes [0.0, 4.0, 8.0, 10, 13.89, 16, 17.78, 20] m/s.

However, when we applied this method, we noticed issues during testing, especially in situations where the model needed to brake quickly, like when approaching stop signs or pedestrians. The model didn't brake properly in these crucial situations, which indicated a potential problem: This approach does not account for intra-class variance and treats all frames within the same class equally. To explore this further, we conducted an ablation study by setting uniform weights ($w_c = 1$) for all classes. The results of this experiment are presented and discussed in Section 5.3.

3.4.2 Frame Importance: Data Filtering

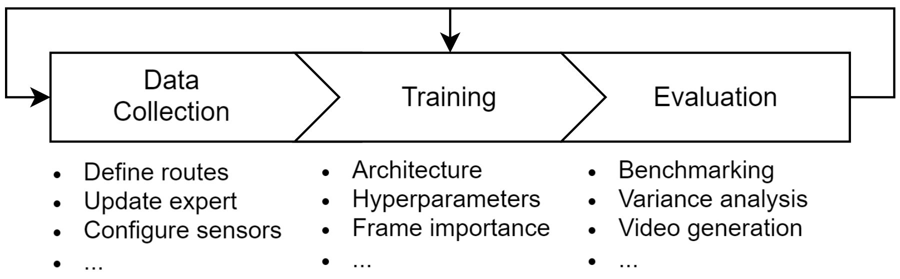
Autonomous driving datasets typically contain many redundant frames from either waiting times or routine driving, while critical decisions must be made in a fraction of a second based on very few key frames. Despite research showing that pruning large scale datasets can improve training efficiency and thus reduce resource costs, or even improve performance in some machine learning tasks [SGS+23, YXP+22], the challenge of effectively guiding the learning process to focus on the most relevant parts of the dataset has not been thoroughly studied in the field of autonomous driving.

In this work, we propose a simple heuristic and evaluate its ability to detect relevant frames for model training, measuring the impact on overall performance. Our central hypothesis is that *important frames are those where the agent reacts to environmental stimuli by adjusting its future path or target speed*. To implement this, our method selects frames where the model's target labels change significantly compared to earlier frames. More precisely, we retain all frames where the target speed changes by more than 0.1 m/s, or the angle to any of the ten path checkpoints shifts by more than 0.5° relative to the previous frame. This heuristic-based method captures approximately 40% of all frames, focusing on those that introduce meaningful variations during driving.

Furthermore, it is still important for the model to be exposed to portions of the data considered redundant by the heuristic. For example, when waiting for a gap in oncoming traffic to overtake an obstacle in the ego agent's lane, the target labels may not change. While such data can be redundant due to extended waiting times, the model still needs to learn to judge gaps accurately. To prevent the model from producing undefined results in these situations, we randomly select and retain 14% of the remaining frames, discarding the rest. With this filtering strategy, we obtain datasets containing 162k frames when excluding Town13 data (48% of the the full dataset, see Section 4.1 for more details), and 259k frames (49%) with Town13 data. We test and discuss the performance of models trained on this filtered dataset in Sections 5.1 and 5.4.

4 Benchmarks and Datasets

Developing a competitive model for the CARLA Leaderboard 2.0 requires addressing all stages of the machine learning pipeline. As depicted in the following diagram, our workflow in this project was structured into three primary phases: data collection, model training, and evaluation. This process was inherently iterative, as errors identified through qualitative video analysis often prompted adjustments in earlier phases, such as refining expert behavior or modifying the weights used in loss functions during training.



For reasons described in more detail in the following sections, the sets of routes provided by the CARLA Leaderboard team were not directly suitable for use as training or validation sets in this project. Consequently, as part of this iterative process, we generated custom datasets and evaluation benchmarks, which we will describe in greater detail in the following.

4.1 Training Set

Our dataset is divided into two parts, which contain data from the following towns:

- **Town12: 1885 routes, 198k frames.** The CARLA team provides a set of 90 training routes on Town12 with a total length of 780.6km. However, directly collecting data on these routes is suboptimal for two reasons. Firstly, such a dataset would contain too much data of routine driving in between scenarios, where nothing interesting happens. This basic driving behavior can be learned relatively easily by TransFuser++ and does not require extensive amounts of training data. Secondly, the distribution of scenarios is highly imbalanced on the official training routes (see Figure 4.1), which may lead to the model neglecting less common scenarios during training.

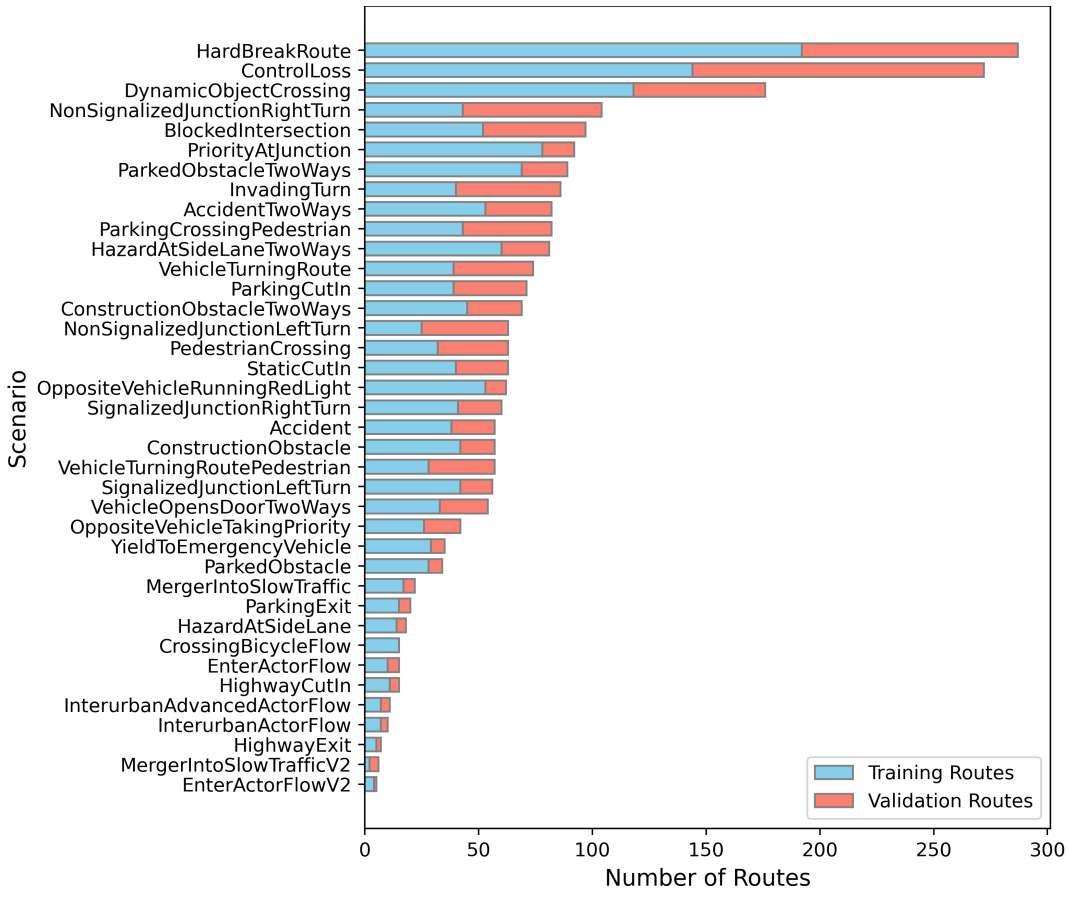


Figure 4.1: **Scenario distribution** in the official training routes (Town12) and validation routes (Town13).

Therefore, we create a set of training routes as follows: We split the official routes into shorter segments containing one scenario each, including a short distance of normal driving (roughly 50-100m on average, the exact amount varies depending on the scenario) before and after the scenario (we find this to be sufficient data to learn normal non-scenario driving behavior). After removing duplicates, we sample from these routes with replacement to obtain a set of 50 short routes per scenario. As shown in Figure 4.1, some scenarios only occur a handful of times in the official routes, meaning they will be used multiple times for data collection (with different weather and traffic initialization).

- **Towns 01-05, Town10HD: 1828 routes, 139k frames.** To increase diversity, we additionally collect data on old towns which were provided by the CARLA team for previous self-driving challenges. On these towns, we use the route files from [JCG23], which are generally similar in length to our Town12 routes and contain only those six scenario types which were already present in the previous leaderboard 1.0, with additional routes for lane changes on highways. The six scenario types are: ControlLoss (171 routes), DynamicObjectCrossing (171 routes), VehicleTurningRoute (520 routes), SignalizedJunctionLeftTurn (248 routes), OppositeVehicleRunningRedLight (179 routes), SignalizedJunctionRightTurn (175 routes).

When training models for leaderboard submission (as opposed to internal evaluations and ablations), we include an additional third part in the training set:

- **Town13: 1722 routes, 194k frames.** This town contains 20 validation routes for the 2024 CARLA challenge with a total length of 247.6km. Despite the shorter total length compared to the training set, the validation routes are more densely populated with scenarios (see Figure 4.1), which helps increase the number of scenario instances in the training set. Analogously to the Town12 data, we split the long routes into shorter segments and upsample to 50 routes per scenarios.

On these routes, we collect data with the PDM-lite expert explained in Section 3.2 at a frequency of 4Hz. Our dataset includes all the input and output data modalities needed to train TransFuser++ (see Section 3.3). In total, our training dataset contains roughly 337k frames in the internal setting, and 531k frames when including Town13 data for leaderboard models. Similar to [JCG23], we apply data augmentation in the form of rotations and translations, randomly shifting all data modalities by up to 1 meter to the left or right, and rotating them by up to 5° around the yaw axis of the ego. When loading the dataset into memory for training, these augmented frames are used with a probability of 50% (otherwise, the unaltered frames are loaded).

4.2 *Town13 Short* Benchmark

The CARLA team provides a set of 20 official validation routes on Town13 which on average are 12.39 km long and contain 93 scenarios. While performance on these routes is an important benchmark to evaluate generalization performance in a setting similar to the official leaderboard, it is not suitable for rapid iteration and allows only limited insights into individual scenario performance. When evaluating many scenarios in succession, the evaluation is biased towards scenarios that appear early in the routes, since improvements on later scenarios will not have any impact if the vehicle does not reach them in the first place. Thus, we again split the routes into segments containing one scenario each and sample up to 15 routes per scenario type without replacement to create the *Town13 short* benchmark. There are 38 scenario types, but in some cases, fewer (or no) routes are available, which gives a total of 400 routes from 36 scenarios in this benchmark (see also Figure 6.3 for instance counts for each scenario type).

The following explanation aims to clarify how driving scores should be interpreted on this benchmark, which includes only a single scenario per route, as this differs significantly from scores on benchmarks with longer routes, such as those from the official leaderboard or validation routes. To this end, consider a model that consistently collides with another vehicle when attempting a particular scenario, but manages to continue and finish the route. Because a single vehicle collision results in an infraction score of 0.6, the mean driving score in this case would be 60. Since this performance would typically be deemed a failure by human standards, an average driving score of around 60 or lower indicates a highly problematic scenario, while a score between 60 and 80 still reflects significant unreliability.

In contrast, current state-of-the-art models achieve driving scores of <10DS on official long routes, which is mainly due to the sequential evaluation of scenarios: If a model produces a vehicle collision in only 20% of cases when attempting scenarios and no other infractions (much better than the example above), that would result in roughly 19 expected collisions when evaluating 93 scenarios in sequence (as on the validation routes), producing a driving score of $100 \times 0.6^{19} = 0.006$ when assuming full route completion. We further discuss the implications of the chosen metrics on long routes in Section 6.1.

4.2.1 Estimating Evaluation Variance

The validation set can be viewed as a sample from the population of all possible scenarios on Town13. Ensuring that the validation sample is representative of the entire population is important to prevent overfitting. In addition, evaluating routes in CARLA involves a lot of inherent variance due to factors like different traffic initializations, weather conditions, and sensor noise. Estimating the evaluation variance is necessary to determine whether improvements in model performance between iterations are genuinely significant or simply due to random chance. To this end, we bootstrap the evaluation variance for different sample sizes on a per-scenario basis and decide on a tradeoff between variance and computational cost.

We evaluate a trained model ("Base" configuration) on all instances of four hand-picked scenarios present in the official validation routes. The numbers of instances for these four scenarios are as follows: InvadingTurn: 45, ConstructionObstacleTwoWays: 23, NonSignalizedJunctionRightTurn: 60, ControlLoss: 127. In bootstrapping terminology, the resulting driving scores represent the initial sample from the population of all possible evaluations of model performance on the respective scenario type on Town13. For $K = 100000$ repetitions, we resample a set of N driving scores for each scenario with replacement, and calculate mean driving score across the N chosen routes per scenario. The histograms of the resulting means are plotted in Figure 4.1, and standard deviations are shown in Table 4.1.

As expected, evaluation variance decreases with increasing sample size, though the standard deviations differ notably between scenarios. A common rule of thumb used to interpret standard deviations is the 68–95–99.7 rule, which states that in a

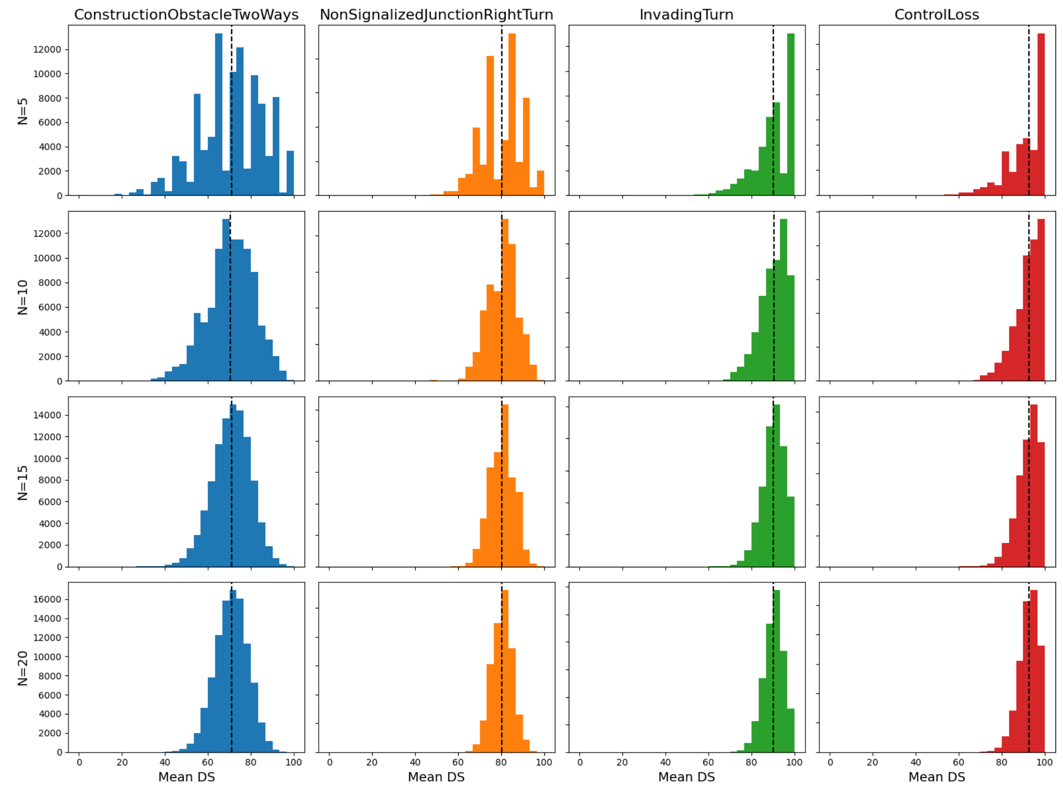


Figure 4.2: **Evaluation variance** on four hand-picked scenarios for varying number of routes per scenario (N).

Sample size	COTW (Mean DS: 71.0)	NSJRT (Mean DS: 80.0)	IT (Mean DS: 90.0)	CL (Mean DS: 92.6)
N = 5	15.28	9.5	9.2	9.17
N = 10	10.78	6.72	6.52	6.5
N = 15	8.82	5.49	5.27	5.28
N = 20	7.58	4.76	4.60	4.58

Table 4.1: **Bootstrapped evaluation standard deviation** for four scenarios (ConstructionObstacleTwoWays, NonSignalizedJunctionRightTurn, InvadingTurn, ControlLoss) when averaging driving scores over varying number of routes per scenario *N*.

Gaussian distribution, roughly 68%, 95% and 99.7% of random samples fall within one, two and three standard deviations from the mean, respectively. The distribution of driving scores in our case roughly resembles a Gaussian distribution for $N \geq 10$, if the mean is not too close to the edges of the support (100 DS). For $N = 15$, we can reasonably assume that performance improvements on these "unsolved" scenarios are unlikely to be due to chance if they exceed 10–15 DS, depending on the scenario type. Trading off evaluation variance and computational resources required for individual evaluations, we choose $N = 15$ for our evaluation set. A typical evaluation then takes around 4 hours when parallelizing evaluations across 20 GPUs, depending on GPU availablity in our compute cluster.

It's important to note that experiments with different datasets or model architectures introduce additional variance from the training process itself, stemming from factors like weight initialization and random processes during data loading. Thus, while individual scenario results were frequently considered during the iterative process to identify weaknesses in our self-driving models, this thesis primarily focuses on aggregate metrics across all scenarios in the evaluation set, since these are subject to much less variance. When repeating training and evaluation with identical parameters (varying only the random number generator seeds), the driving scores generally fluctuate by no more than 2–3 DS when averaged over all 400 routes in the Town13 short evaluation set.

5 Experiments

In this chapter, we present the official leaderboard results and qualitatively assess the impact of the key modifications made to adapt TransFuser++ to CARLA Leaderboard 2.0 and its new scenarios, evaluating each change in isolation. Additionally, we conduct experiments to evaluate the influence of varying aspects of the training dataset, such as filtering out less significant frames and adjusting the range of the LiDAR sensor. We also present ablation studies on various techniques that have proven successful in previous works on self-driving in CARLA [CPJ+22, CK22], including ensembling and perception-only pre-training. In short, we aim to provide an overview of how both our proposed modifications and previously successful techniques perform on the scenarios present in the new CARLA leaderboard.

5.1 Official Leaderboard

Model	**DS** $\uparrow$	**RC** $\uparrow$	**IS** $\uparrow$
LRM [RGdS+24]	1.2	9.6	0.318
Kyber-E2E [ZER+24]	3.5	8.5	0.50
CarLLaVA	6.9	18.1	0.42
TF++ (no filtering)	5.2	11.3	0.48
TF++ (w/ filtering)	5.6	11.8	0.47

Model	**Ped** $\downarrow$	**Veh** $\downarrow$	**Stat** $\downarrow$	**Red** $\downarrow$	**Sto** $\downarrow$	**Sce** $\downarrow$	**Blo** $\downarrow$
LRM [RGdS+24]	0.249	1.643	0.249	0.249	0.398	1.195	0.598
Kyber-E2E [ZER+24]	0.682	1.079	0.625	0.284	0.114	0.114	0.000
CarLLaVA [RCM+24]	0.053	1.173	0.053	0.000	0.107	0.133	0.453
TF++ (no filtering)	0.000	1.240	0.043	0.043	0.128	0.385	0.727
TF++ (w/ filtering)	0.000	1.263	0.082	0.041	0.122	0.204	0.693

Table 5.1: **Official results on CARLA Leaderboard 2.0.** Secret test routes (Town 14). TF++ (Ours) outperforms prior modular pipelines [RGdS+24, ZER+24], and places 2nd overall. Upper table contains main metrics, lower table contains infraction rates per kilometer for different infraction types.

In this section, we present our results on the official leaderboard of the CARLA Autonomous Driving Challenge, part of the Autonomous Grand Challenge 2024 hosted at the CVPR 2024 Workshop. Before the 2024 challenge, the best result on this benchmark was a driving score of 1.23 achieved by LRM [RGdS+24]. Our best submitted model outperforms this previous state of the art by 355% with a driving score of 5.6. With this result, we achieve first place on the MAP track of the challenge and second place in the SENSORS track, out of a total number of 40 participant teams and 280 submissions, according to the organizers.

Our models listed as TF++ in Table 5.1 are enhanced versions of our ´´Base" model as described in Chapter 3. The improvements compared to the "Base" model are:

- Additional training data collected on the validation town Town13, cf. Section 4.1.
- Bigger architecture of the convolutional perception modules in TransFuser++, cf. Section 3.3.
- Two-stage training: Includes pre-training stage with perception losses only, cf. Section 3.4.
- Ensemble model consisting of three training seeds, cf. Section 3.4.
- Early Stopping: Due to the design of the performance metrics, we include a mechanism that makes our agent stops after 1.5km (even if it could go further). This has a significant effect on DS, RC and IS - We discuss the motivation for and effect of applying this technique in more detail in Section 6.1.
- Data filtering: The model marked as "TF++ (w/ filtering)" is trained on a filtered version of the dataset focusing on the most interesting frames as explained in Section 3.4.2.

Ablation studies on the effect of some of these changes are provided in Section 5.6.

5.1.1 Comparison to Other Leaderboard Entries

Among the models listed in Table 5.1, CarLLaVA and TF++ (ours) are the only end-to-end models. LRM [RGdS+24] uses a modular design with separate modules for perception, risk assessment and navigation, including a hand-crafted finite-state machine for decision making. Kyber-E2E [ZER+24] is another modular architecture, consisting of modules for sensing, perception, tracking and prediction, as well as planning and control. In contrast to the other entries listed in the table, it uses the ground truth birds-eye-view HD map as model input, and is thus only eligible for the MAP track of the 2024 challenge (see Section 3.1).

Our model is the best openly available model in the challenge with published code, dataset and models. Major differences of the overall highest scoring closed-source entry, CarLLaVA [RCM+24], to our method, include the removal of LiDAR data as model input, dataset scale, and the usage of a large vision transformer for image perception. With roughly 2.9 million frames, their training dataset is roughly 5.5× larger than ours (531k frames). Additionally, their transformer vision encoder (LLaVA-NeXT [LLL+24]) is pre-trained on "internet scale data", whereas the convolutional neural network we use in our RGB perception branch is pre-trained on ImageNet, and the LiDAR branch is trained from scratch. As highlighted in their technical report, this pre-training is crucial for CarLLaVA's performance. In terms of the number of parameters, their models are between 2.9× and 10× larger than ours (350 million to 1.2 billion compared to 120 million). Other differences include the use of two target points as input, which we found to be not useful in our experiments (cf. Section 5.6), as well as the use of a "semi-entangled output representation" which uses the same equidistant checkpoint predictions we use for lateral control, but uses waypoints as in the original TransFuser [CPJ+22] implementation for longitudinal control (instead of a target speed classification network). Furthermore, CarLLaVA uses multi-layer perceptrons to predict waypoints and checkpoints from the intermediary transformer features, wherease we employ a GRU to predict checkpoints and a MLP to predict target speed classes.

5.2 Expert Style

Exp. Style	**DS↑**	**RC ↑**	**IS ↑**
PDM-l	78.9 ± 1.1	96.4 ± 0.6	0.80 ± 0.01
Base	84.8 ± 0.5	98.6 ± 0.3	0.86 ± 0.00

Exp. Style	**Ped ↓**	**Veh ↓**	**Stat ↓**	**Red ↓**	**Sto ↓**	**Sce ↓**	**Blo ↓**
PDM-l	0.31	2.16	0.54	0.06	0.21	0.35	0.20
Base	0.20	1.69	0.17	0.13	0.04	0.20	0.09

Table 5.2: **Expert style.** Results on Town13 short, reported over 3 training seeds. Original PDM-lite expert style (PDM-l) vs. modified behavior (Base). Includes confounding factor of reduced expert target speeds.

Initially, we used PDM-lite as the expert driver for data collection, expecting good results since PDM-lite achieves high scores compared to other privileged driving algorithms in CARLA [Bei24]. However, early video analysis of the trained model revealed poor braking performance, prompting a change in IDM parameters, as detailed in Section 3.2.1. Generally, this change made the expert brake more abruptly and at shorter distances from objects triggering the braking response, such as stop signs, obstacles, or pedestrians, making the expert's behavior easier for an imitation learning agent to learn.

Figure 5.1 illustrates this change when approaching pedestrians. By default, the expert slows down when it predicts that a pedestrian will enter the driving path, even when the pedestrian is often obstructed by an obstacle like a parked vehicle.

This made it difficult for the model to learn, as there was no clear signal in the IL agent's camera or LiDAR input data. With our adjusted IDM parameters, the expert brakes more sharply, stopping about 4 meters before the now visible pedestrian. The improvement likely arises from the adjusted behavior providing a clear braking signal for the model to learn from (a pedestrian directly in front of the ego vehicle), whereas the default behavior required the model to generalize across various situations where a pedestrian might appear further ahead.

While we used a pedestrian scenario to illustrate the effect, this change also impacts various other scenarios and types of infractions, as shown in Table 5.2. Overall, the change in expert behavior resulted in a significant performance boost of 5.9DS on average. Although there is a potential confounding factor – the reduction of the expert's target speeds from 80% to 72% of the speed limit when there is no traffic (which was changed in the same dataset iteration) – we estimate that the impact of this change is minor in comparison. Importantly, the updated dataset led to a 35% reduction in pedestrian collisions, a 22% decrease in vehicle collisions, and a 69% drop in collisions with static objects. Notably, these improvements occurred without significantly affecting the expert's own driving performance. Overall, this result highlights the importance of considering expert driving style when developing a competitive imitation learning agent for the CARLA Leaderboard.

5.3 Speed Weights

Speed W.	**DS↑**	**RC ↑**	**IS ↑**
✓	81.7 ± 0.9	97.8 ± 0.1	0.83 ± 0.01
✗ (Base)	84.8 ± 0.5	98.6 ± 0.3	0.86 ± 0.00

Speed W.	**Ped ↓**	**Veh ↓**	**Stat ↓**	**Red ↓**	**Sto ↓**	**Sce ↓**	**Blo ↓**
✓	0.18	1.89	0.27	0.24	0.12	0.23	0.13
✗ (Base)	0.20	1.69	0.17	0.13	0.04	0.20	0.09

Table 5.3: **Speed weights.** Results on Town13 short, reported over 3 training seeds. Weights: [0.29, 1.30, 0.69, 0.81, 4.43, 4.76, 3.90, 2.41] for the speeds [0.0, 4.0, 8.0, 10, 13.89, 16, 17.78, 20] m/s.

Our first experiment related to frame importance focused on applying class weights in the target speed classification loss as described in Section 3.4.1. In the original TransFuser++ implementation, more frequent classes are assigned lower weights, while less frequent classes are given higher weights. This approach aims to prevent the model from becoming biased toward common classes.

As shown in Table 5.3, removing these class weights and instead using uniform weights ($w_c = 1$) across all classes significantly improved the model's performance,

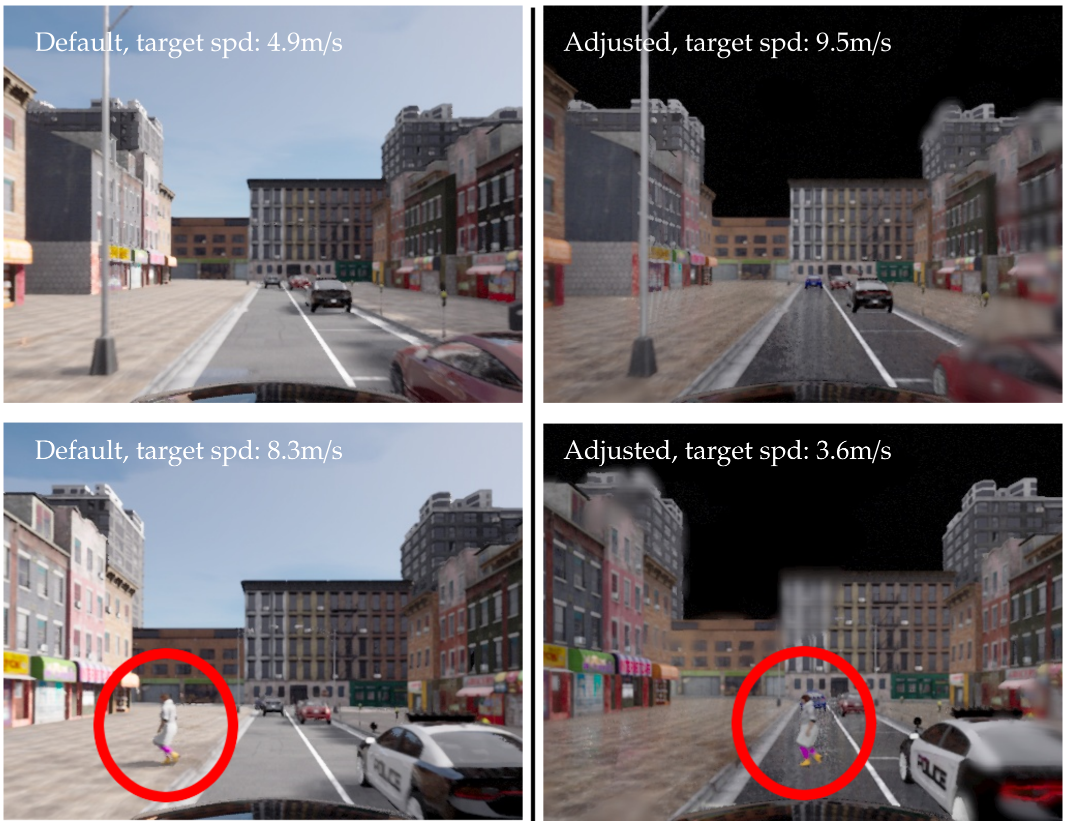

Default, target spd: 4.9m/s
Adjusted, target spd: 9.5m/s
Default, target spd: 8.3m/s
Adjusted, target spd: 3.6m/s

Figure 5.1: **Expert style compared on the same route.** The default PDM-Lite brakes early (left), when the pedestrian is hardly visible in the image, while the adjusted expert brakes later (right).

resulting in an average driving score increase of 3.1DS. This finding suggests that class frequencies are not a reliable measure for frame importance in our context.

For example, class 0, which corresponds to braking, is the most frequent class in our dataset (cf. Figure 3.3). While class 0 includes many redundant frames, such as waiting at red lights, it also contains critical events, like stopping for pedestrians or at stop signs, which are essential for safe driving. When class weights are applied, the loss associated with class 0 is reduced due to its high frequency, making it easier for the model to ignore brief, but important, braking phases in crucial situations.

By removing the class weights, we ensure that the model assigns appropriate importance to these critical braking frames. In general, applying class weights based on frequency is not a viable strategy when classes are inhomogeneous and contain both important and unimportant frames.

5.4 Data Filtering

Setting	**DS↑**	**RC ↑**	**IS ↑**
Base	84.8 ± 0.5	98.6 ± 0.3	0.86 ± 0.00
Filt	83.7 ± 1.6	98.5 ± 0.1	0.85 ± 0.01

Setting	**Ped ↓**	**Veh ↓**	**Stat ↓**	**Red ↓**	**Sto ↓**	**Sce ↓**	**Blo ↓**
Base	0.20	1.69	0.17	0.13	0.04	0.20	0.09
Filt	0.12	1.94	0.16	0.11	0.05	0.23	0.08

Table 5.4: **Dataset filtering.** Results on Town13 short, reported over 3 training seeds. Full dataset (Base) vs. filtered dataset containing half the number of frames (Filt).

As an alternative approach to measuring importance of dataset frames, we propose the use of heuristics that estimate whether a frame changes the model's target labels compared to previous frames. We filter the dataset based on the heuristic introduced in Section 3.4.2 and compare the performance of models trained on this filtered data to those trained on the full dataset to detect changes in driving performance. To maintain the overall number of gradient updates during training, we compensate for the reduced dataset size by training the model for twice the number of epochs.

In Table 5.4, we present the results of this filtering strategy compared to training on the full dataset. While the average driving score is slightly better with the full dataset, the difference of 1.1 DS falls within the typical range of variance observed across three training seeds and evaluation runs. Furthermore, the maximum performance achieved by an individual model across seeds is marginally higher when trained on the filtered dataset (85.5 DS vs. 85.3 DS). In addition, on the official leaderboard

(Table 5.1), our score of 5.2DS on the SENSORS track was obtained using a model trained without any filtering, whereas the score on the MAP track improved to 5.6 DS when the filtering strategy was applied. Again, this difference may be due to evaluation variance. In summary, despite reducing the dataset size by more than 50%, the proposed heuristic maintains comparable performance to models trained on the full dataset, effectively eliminating redundant frames while preserving essential information for the model's learning process.

5.5 LiDAR Range

Range	DS↑	RC ↑	IS ↑
32m (Base)	84.8 ± 0.5	98.6 ± 0.3	0.86 ± 0.00
64m	86.2 ± 0.1	98.1 ± 0.2	0.87 ± 0.00

Range	Ped ↓	Veh ↓	Stat ↓	Red ↓	Sto ↓	Sce ↓	Blo ↓
32m (Base)	0.20	1.69	0.17	0.13	0.04	0.20	0.09
64m	0.12	1.37	0.20	0.16	0.06	0.15	0.13

Table 5.5: **LiDAR range.** Results on Town13 short, reported over 3 training seeds.

The new obstacle avoidance scenarios in Leaderboard 2.0 require the ego vehicle to wait for a gap in the opposite lane that is large enough for it to safely accelerate, pass the obstacle in the opposite lane and return to its original lane. These scenarios occur in both urban and rural areas, and thus require accounting for vehicles far ahead of the ego vehicle. Due to observed failures of the "Base" model in these scenarios, where it often misjudged the distance of oncoming traffic, we extended the LiDAR range in front of the ego vehicle from 32 meters to 64 meters, while keeping the side and rear LiDAR ranges unchanged.

This adjustment improved the model's performance by enabling more accurate detection of distant objects, leading to a 19% reduction in vehicle collisions. Consequently, the average driving score increased by 1.4 points. Unlike in the case of data filtering, we are confident that this difference is significant because of the reduced standard deviations oberved across trials; all three seeds achieved a driving score of at least 86 DS, a score that was not reached with any seed with the standard LiDAR range. This finding is consistent with the recent study by [ZER+24], which observed a notable rise in vehicle collisions when the LiDAR range was limited to 32 meters.

Although this experiment showed promising results, we were unable to finalize and integrate these improvements in time for the challenge. Consequently, the extended LiDAR range was not used in the competition.

5.6 Additional Experiments

Setting	**DS ↑**	**RC ↑**	**IS ↑**
Base	84.8 ± 0.5	98.6 ± 0.3	0.86 ± 0.00
Big	84.5 ± 1.6	97.9 ± 0.7	0.86 ± 0.01
Pre	86.5 ± 0.9	98.4 ± 0.6	0.87 ± 0.01
Ens	86.2	98.3	0.87
GT Vel	82.8 ± 0.4	97.9 ± 0.7	0.84 ± 0.00
2TPs	82.2	95.6	0.85
NoAug	82.4 ± 1.7	98.7 ± 0.2	0.83 ± 0.02
NoOT	83.4 ± 1.0	97.9 ± 0.6	0.84 ± 0.01
Expert	*99*	*100*	*0.99*

Setting	**Ped ↓**	**Veh ↓**	**Stat ↓**	**Red ↓**	**Sto ↓**	**Sce ↓**	**Blo ↓**
Base	0.20	1.69	0.17	0.13	0.04	0.20	0.09
Big	0.07	1.73	0.29	0.07	0.05	0.27	0.14
Pre	0.14	1.40	0.13	0.16	0.03	0.17	0.12
Ens	0.16	1.55	0.16	0.16	0.04	0.22	0.09
GT Vel	0.04	1.94	0.20	0.13	0.06	0.38	0.08
2TPs	0.17	1.75	0.44	0.12	0.08	0.20	0.29
NoAug	0.11	2.13	0.47	0.23	0.03	0.17	0.07
NoOT	0.21	1.90	0.22	0.20	0.03	0.19	0.12
Expert	*0.01*	*0.10*	*0.01*	*0.00*	*0.00*	*0.00*	*0.00*

Table 5.6: **Results on Town13 short.** Std over 3 training seeds where available. Training on Towns 01-05, 10, 12 (337k frames).

Table 5.6 provides an overview over some additional experiments and ablation studies. One group of experiments relates to scaling up the model, which has proven effective in Leaderboard 1.0 [CPJ+22, JCG23].

- *Big*: Bigger architecture of the convolutional perception modules in TransFuser++, cf. Section 3.3. This change does not seem to have an effect on performance in this experiment, as the score differences are smaller than the standard deviations.

- *Pre*: Indicates pre-training with perception losses only, cf. Section 3.4. We observe a performance increase of 1.7DS on average, indicating that the reduced complexity of perception-only pre-training helps the model build a better foundation for the downstream driving task.

- *Ens*: Ensemble model consisting of three training seeds, cf. Section 3.4. We observe a performance increase of 1.4DS compared to the average performance of the individual models. It also beats the maximum performance of an individual Base model (85.3DS), indicating that the individual models learned complementary features and decision-making strategies that, when combined, enhance overall performance.

The *GT Vel* (ground truth velocity) experiment refers to appending the true 2D velocity vectors of dynamic actors, such as pedestrians, cyclists and vehicles to the BEV LiDAR model input. Thus, this experiment increases the number of BEV input channels from 1 to 3 and uses privileged inputs not available on the leaderboard. Velocity vectors are only appended to pixels with existing LiDAR hits, in order to avoid the confounding factor of exposing ground truth object shapes to the model. It aims to determine whether this information could reduce the most common type of infraction in all of our models: vehicle collisions.

Unexpectedly, this experiment produced a negative result: While it did produce the lowest pedestrian collision rate of any experiment at 0.04 collisions per kilometer, vehicle collisions increased by about 15%, resulting in an overall performance drop of 2DS on average. This suggests two important points:

- Adding velocity information is ineffective at reducing vehicle collisions - the issue may lie in a failure to forecast the future behavior of other actors, or to connect other vehicles' current and future behavior to the ego's own predicted path to infer appropriate driving signals in certain crucial moments.
- Even if the additional information is not helpful to avoid collisions, the model should theoretically be able to learn to ignore it. However, the performance drop indicates that the convolutional perception stack has problems learning to disentangle different dimensions of the LiDAR input to ignore irrelevant parts, picking up spurious correlations instead. This problem is known as causal confusion and has been observed in various studies on imitation learning, such as [PSL+24, WLQ+21].

The *2TPs* (two target points) experiment represents another attempt at reducing vehicle collisions (performed with a single training seed). In this setup, we provided the model with the next two target points on the route (as a concatenated 4D vector) instead of just one, based on the hypothesis that this might enhance the model's ability to anticipate and respond to lane changes by detecting them earlier. However, this modification resulted in a decline in performance. Additionally, we experimented with introducing the target points earlier in the architecture as an extra MLP-encoded token for the transformer decoder, but this approach also failed to yield performance improvements (results for this experiment are not shown in the table).

Finally, we provide two ablation studies related to dataset diversity.

- *NoAug*: Ablation on rotation and translation data augmentations, cf. Section 4.1. Without augmentation, we observed an increase in collisions with both vehicles and layout elements, resulting in a drop 2.4DS on average mainly due to an increased number of collisions with vehicles and static objects. This underscores the importance of augmentation in providing resistance to out-of-distribution (OOD) compounding errors: In almost all situations, our expert driver stays very close to the center of the lane. Without augmentation, if the model deviates slightly from the center during inference, it may produce highly inaccurate outputs due to having left its training distribution. Adding shifts and rotations during data collection teaches the model how to respond when it has deviated from the center of the lane. As a result, the model learns the corrective behavior of steering back towards the center, reducing the likelihood of collisions.

- *NoOT*: Ablation on using additional training data collected on old Leaderboard 1.0 towns, cf. Section 4.1. In this experiment, we correct for the reduced number of frames (198k instead of 337k frames) by oversampling the remaining data (Town12 only) in each epoch. Thus, each epoch still contains the same number of batches, which ensures a constant number of total gradient updates. This results in a drop in average performance of 1.4DS with a standard deviation of 1.0 over the three training seeds. While it is not clear if this drop is significant, we conclude that the increased dataset diversity obtained by collecting data on the old LB1 towns probably leads to a small increase in performance.

6 Discussion

In this chapter, we delve into the implications of our findings. We begin by focusing on how the current leaderboard evaluation metrics can lead to counterintuitive results, exploring mathematical strategies to optimize leaderboard scores, and proposing changes to the metrics to encourage more realistic driving behaviors. Next, we provide a qualitative analysis of our model's performance, highlighting strengths and identifying areas for improvement. Lastly, we offer a range of promising directions for future research inspired by our analysis and related work.

6.1 On Leaderboard Evaluation Metrics

Here, we discuss an issue introduced by the evaluation metrics used on the official leaderboard, which can sometimes yield counterintuitive results - models that might be deemed superior based on qualitative human assessment may receive lower scores on the leaderboard. We also demonstrate a way to exploit this fact to maximize leaderboard scores. As stated in 3.1.3, the main driving score is calculated as the product of route completion and *unnormalized* infraction score. This leads to a tradeoff where it can be advantageous to stop an agent preemptively, reducing RC and increasing IS to improve the overall driving score: In simple terms, if a model accumulates enough infractions on a route segment such that the resulting infraction score diminishes the driving score more than the distance traveled would increase it, then it may be preferable for the model to stop driving and wait for the simulator to time out rather than continue along the route. In the following, we formulate this concept mathematically and determine the optimal distance an agent should travel to maximize the driving score.

6.1.1 Optimizing Driving Score Mathematically

For this analysis, we approximate driving score as a function of the fraction $x \in [0,1]$ of the route that the agent completes (see Section 6.1.2 for details):

$$DS(x) = RC(x) \times IS(x) \approx 100xI^{xL},\ x \in [0,1],$$

$$DS(x) \approx 100xI^{xL},\ x \in [0,1],$$

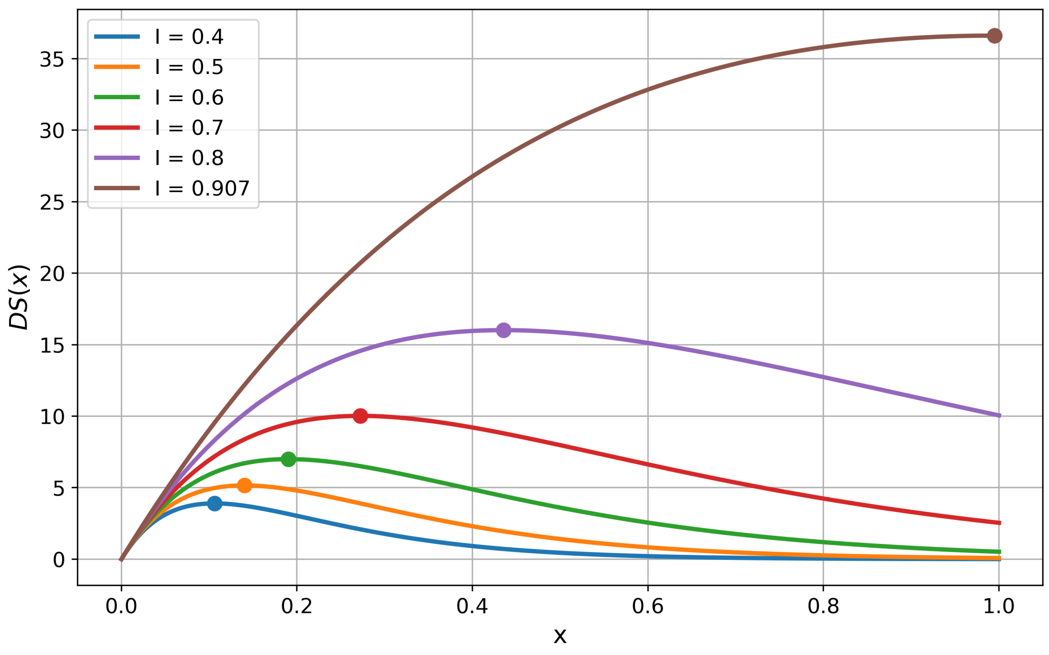


Figure 6.1: **Approximate driving score** as a function of route completion fraction x for different infraction coefficients I. $DS(x)$ has a global maximum at $x < 1$ if $I < 0.907$, which creates an incentive to stop early.

where L is the route length, and we define

$$I = 0.5^{\frac{\#Ped}{d}} * 0.6^{\frac{\#Veh}{d}} * 0.65^{\frac{\#Stat}{d}} * 0.7^{\frac{\#Red}{d}} * 0.7^{\frac{\#Yie}{d}} * 0.7^{\frac{\#Sce}{d}} * 0.8^{\frac{\#Sto}{d}}$$

as the *infraction coefficient*, which represents the average penalty factor that the agent accumulates in each kilometer driven. Note that in contrast to the formula for IS (see Section 3.1.3), the exponents here are normalized by the distance traveled d. We exclude infractions that do not have constant penalty factors, which are *MinSpeed* infractions and route deviations, as they have a negligible impact. Maximizing this function, we obtain the solution

$$x_{max} = -(L \cdot \log I)^{-1},$$

with a theoretically maximal driving score of $DS(x_{max}) = -\frac{100}{L \cdot e \cdot \log I}$. Figure 6.1 shows this function, along with the corresponding maxima, for $L = 10.295$ (the mean length in kilometers of the official test routes) and different values of I.

Mathematically, a model profits from early stopping if $x_{max} < 1$, i.e., $I < 0.907$. The figure illustrates this nicely: If $I < 0.907$, expected driving score is maximized at $x_{max} < 1$, with driving scores dropping off significantly at higher route completion fractions x. Furthermore, this threshold is far from being reached by any of the current top entries on the leaderboard, meaning that at the current state of progress, all participants must use a variant of early stopping in order to be competitive.

6.1.2 Infraction Score Approximation

Here, we explain how we approximate driving score as a function of the route completion fraction x. The formula introduced in Section 6.1.1 is:

$$DS(x) = RC(x) \times IS(x) \approx 100xI^{xL},\ x \in [0,1].$$

The first part, $RC(x) = 100x$, is exact (this is simply the definition of route completion). The second part, $IS(x) \approx I^{xL}$, is an approximation. For all infraction types (we use *Ped* as an example here), we approximate the number of infractions when only completing a fraction x of the route as

$$\#Ped(x) = xL\frac{\#Ped}{d}.$$

In words, we assume that the number of infractions scales linearly in x. For example, if the agent has traveled $d = 4$ kilometers and incurred $\#Ped = 4$ pedestrian infraction in the process, then we assume that the number of pedestrian infractions would be 2 if it had only traveled 2 kilometers (or 1.5 if it had only traveled 1.5 km). This is clearly not satisfied in practice (the true number of infractions is always a natural number, and the probability to incur an infraction is not uniform along the route), but it is a reasonable estimate without detailed modeling of the test routes.

With this assumption, the infraction score can be expressed as

$$\begin{aligned}
IS(x) &= 0.5^{\#Ped(x)} * 0.6^{\#Veh(x)} * 0.65^{\#Stat(x)} * 0.7^{\#Red(x)} * 0.7^{\#Yie(x)} * 0.7^{\#Sce(x)} * 0.8^{\#Sto(x)} \\
&\approx 0.5^{xL\frac{\#Ped}{d}} * 0.6^{xL\frac{\#Veh}{d}} * 0.65^{xL\frac{\#Stat}{d}} * 0.7^{xL\frac{\#Red}{d}} * 0.7^{xL\frac{\#Yie}{d}} * 0.7^{xL\frac{\#Sce}{d}} * 0.8^{xL\frac{\#Sto}{d}} \\
&= \left(0.5^{\frac{\#Ped}{d}} * 0.6^{\frac{\#Veh}{d}} * 0.65^{\frac{\#Stat}{d}} * 0.7^{\frac{\#Red}{d}} * 0.7^{\frac{\#Yie}{d}} * 0.7^{\frac{\#Sce}{d}} * 0.8^{\frac{\#Sto}{d}}\right)^{xL} \\
&= I^{xL}
\end{aligned}$$

and we arrive at our driving score approximation.

6.1.3 Leaderboard Score Maximization in Practice

Here, we describe how we use the idea described above to increase our score on the leaderboard. First, it is necessary to estimate I, for which we use infraction statistics collected in evaluations of our leaderboard model (no filtering) on the official Town13 evaluation routes. We obtain $I = 0.43$, which yields $x_{max} = 0.115$. Thus, our model should theoretically stop at $d = L \cdot x_{max} = 1.18$km to maximize expected DS. However, we need to account for distribution shift between the validation routes on Town13

Early stopping	**DS**↑	**RC**↑	**IS**↑	$\widehat{\textbf{DS}}$↑	**RC**↑	**I**↑
✗	0.70	70.1	0.04	5.02	70.1	0.05
✓	4.81	8.0	0.59	0.98	8.0	0.12

Table 6.1: **Early stopping** after 1km on the 20 validation routes, tested with our leaderboard model (no filtering). The modified driving score $\widehat{\text{DS}}$ proposed in Section 6.1.4 using the infraction coefficient I and infraction penalties scaled by a factor of 0.2 successfully removes the incentive for early stopping.

and test routes on Town14. Early leaderboard results suggested that the scenario density is lower on the test routes than on the validation routes, leading to a lower number of infractions overall. To account for this, we heuristically choose $d = 1.5$km in practice.

We explored two methods for tracking the distance traveled: discrete integration using the agent's speed sensor and summing the distances between each pair of GPS ticks recorded by the agent's GPS sensor. We found the latter method to be significantly less accurate, which is likely due to the additive noise which CARLA adds to the GPS sensor. Consequently, we use the first method in practice. When the vehicle reaches the distance threshold, we set the target speed to zero to bring the ego vehicle to a halt and trigger a simulator timeout after 180 seconds.

Early stopping significantly affects all main performance metrics obtained on long Town13 validation routes, as shown in Table 6.1. While the original model without early stopping achieves much higher route completion (70.1 on average), it accumulates enough infractions along the way to severely compromise its driving score. Early stopping trades off route completion for a much better infraction score, increasing overall driving score roughly sevenfold. As for the official leaderboard, Table 5.1 shows that all high-ranking submissions to the Leaderboard 2.0 test server have a route completion of less than 18.1. This indicates that these methods employ some form of early stopping, whether explicitly or implicitly, and the leading entry, CarLLaVA, also acknowledges this strategy in their technical report [RCM+24].

6.1.4 Eliminating the Tradeoff Problem

As described in the previous sections, the tradeoff introduced by the performance metrics used in Leaderboard 2.0 forces participants to terminate evaluations early to remain competitive, which is counterproductive. Therefore, we recommend adjusting the performance metrics for future challenges. Instead of using the infraction score (IS) (which uses the absolute number of infractions in the exponents, see Section 3.1.3) for driving score calculation, we propose using the infraction coefficient (I) as defined above, which incorporates infraction *frequencies*. Put simply, this means dividing the

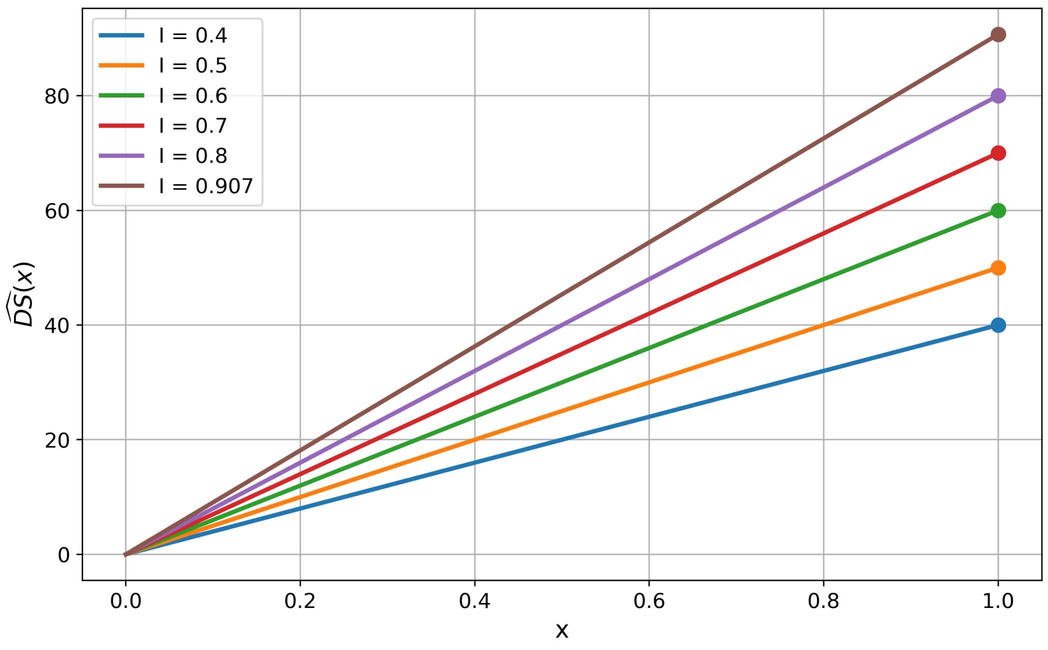


Figure 6.2: **Approximate driving score with our proposed adjustment** as a function of route completion fraction x for different infraction coefficients I. $\widehat{DS}(x)$ increases linearly with x, eliminating the incentive for early stopping.

exponents by the distance d traveled by the agent:

$$I = 0.5^{\frac{\#Ped}{d}} * 0.6^{\frac{\#Veh}{d}} * 0.65^{\frac{\#Stat}{d}} * 0.7^{\frac{\#Red}{d}} * 0.7^{\frac{\#Yie}{d}} * 0.7^{\frac{\#Sce}{d}} * 0.8^{\frac{\#Sto}{d}}$$
$$\widehat{DS} = RC \times I$$

With this normalization and using a similar approximation as before, the driving score can be modeled as a function of the route completion fraction x as follows.

$$\widehat{DS}(x) = RC(x) \times I(x) \approx 100xI,\ x \in [0,1].$$

This function, depicted in Figure 6.2 for varying I, increases linearly with x. This eliminates the incentive to stop early, since the maximum driving score is always reached at $x = 1$, i.e., full route completion. As a side effect, this change leads to an increase in average driving scores achieved with identical models compared to the original formula. If this increase is not desired, it can be corrected by scaling down the penalty factors for all infractions, which reduces the expected driving scores while maintaining the maximum score at 100.

Table 6.1 illustrates the effectiveness of our proposed modifiation. Here, we have scaled all infraction penalties by a factor of 0.2 to keep the resulting driving scores in a similar range as with the original formula. Concretely, we apply a base penalty of $0.2 * 0.5 = 0.1$ for *Ped*, $0.2 * 0.6 = 0.12$ for *Veh*, and so forth. Comparing this to the original metrics, it is clear that the revised driving score calculation successfully discourages early stopping, since the model that stops after one kilometer now receives a much lower driving score, as intended.

6.2 Qualitative Analysis of Driving Performance

In this section, we dive deeper into the strengths and weaknesses of our models, analyze individual scenario performance and point out behavioral failure modes.

6.2.1 Overview

Figure 6.3 provides an overview of how different variants of our model perform on each scenario in the *Town13 short* evaluation benchmark (see Tables 5.5 and 5.6 for averaged metrics). When interpreting these results, it is important to consider the substantial scenario-level evaluation variance discussed in Section 4.2.1. Certain scenario types, such as EnterActorFlow ($N = 4$), HazardAtSideLane ($N = 3$), and HighwayCutIn ($N = 3$), have few instances in the validation routes, which may limit the significance of performance differences between individual models. To mitigate this variance, we include data from nine different training runs and evaluations (three per configuration) and rank the scenarios based on the average driving score across all evaluations, which increases the total sample size by a factor of 9. As a result, the figure provides a more reliable indication of which scenarios our models typically excel in and which they struggle with.

6.2.2 Failure Cases

Here, we visualize and discuss some of the behavioral failure modes encountered in the problematic scenarios and discuss potential solutions. We focus on scenarios that appear more frequently in the evaluation routes and use a single seed of the Base model (Mean DS: 85.3) for this analysis. The visualizations include the front camera image and a birds-eye-view image showing LiDAR hits, the model's checkpoint predictions, the target points used as input and the auxiliary BEV perception predictions using the following colors:

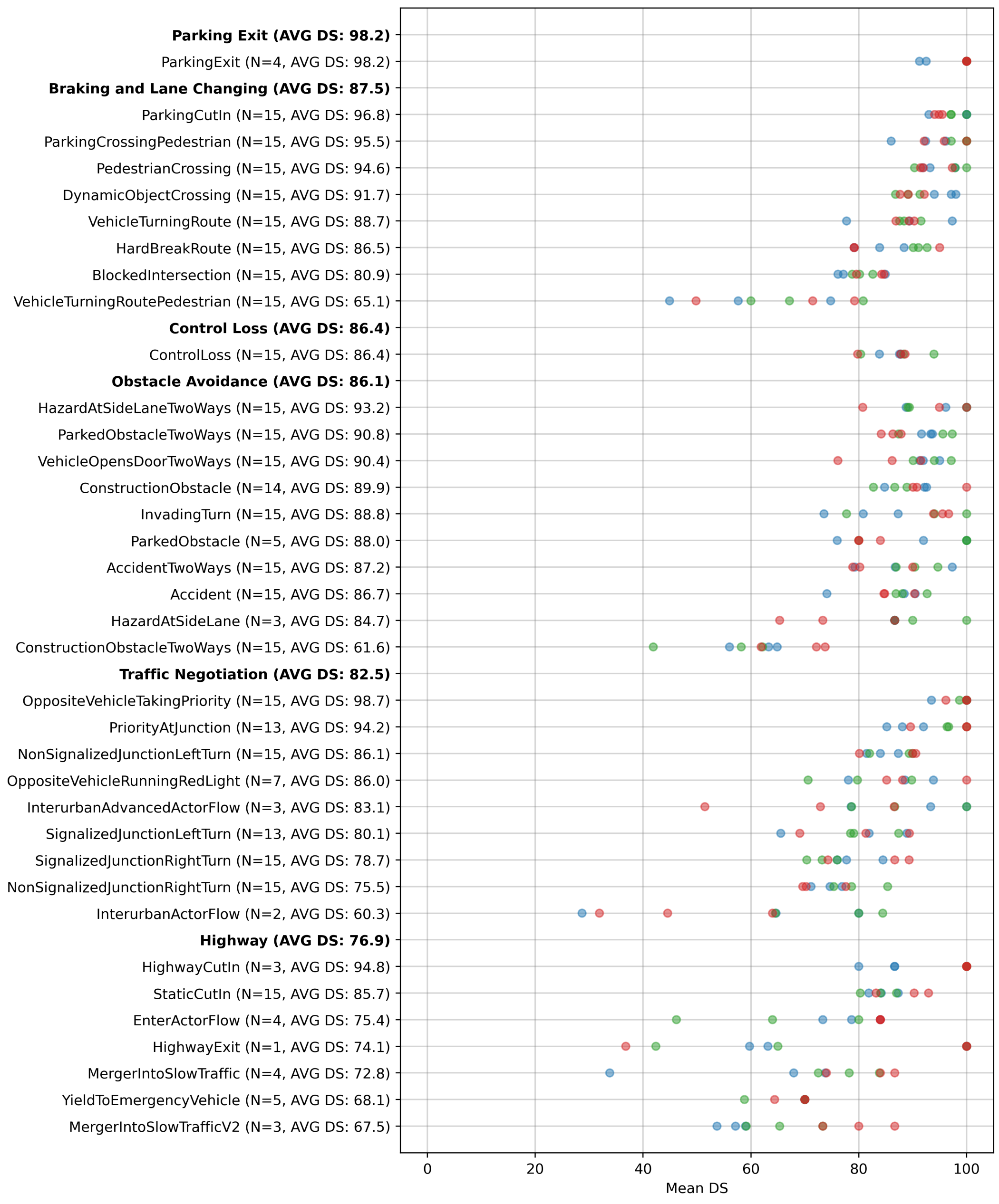


Figure 6.3: **Scenario difficulty within *Town13 short*** averaged over different model configurations. N is the number of routes in the benchmark, *AVG DS* is the average driving score across all 9 included models (total sample size $9 \cdot N$). Includes 3 seeds each of Base (blue), two-stage training (green) and extended LiDAR range (red).

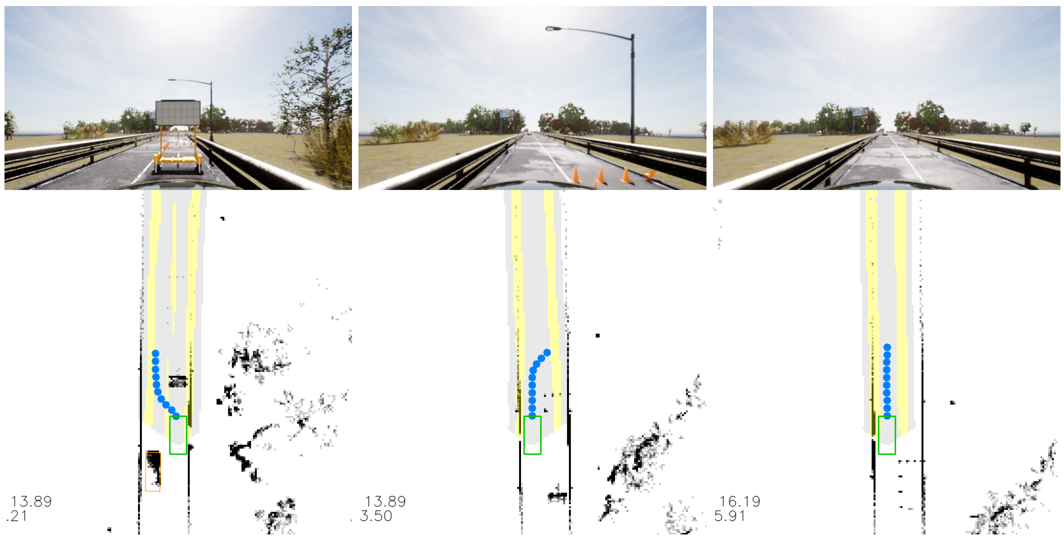


Figure 6.4: **Scenario: ConstructionObstacleTwoWays.** Overreliance on camera image - When traffic cones aren't visible any more, the ego forgets to merge back into its original lane.

⬜	**Grey**: Road (semantics)
🟨	**Yellow**: Road marking (semantics)
🟩	**Light Green**: Green traffic light (semantics)
🟧	**Light Orange**: Vehicle (semantics)
🟩	**Green**: Ego vehicle and pedestrian (bounding box)
🟧	**Orange**: Vehicle (bounding box)
🔵	**Blue**: Path predictions
🔴	**Red**: Target point (used as model input)

ConstructionObstacleTwoWays

With an average performance 61.6DS, this scenario is highly problematic for all models. In this scenario, the ego encounters a construction obstacle blocking its own lane and must pass it by moving into an adjacent lane with oncoming traffic. Figure 6.4 illustrates a common failure mode, where the ego fails to merge back to its original lane, which frequently leads to vehicle collisions. At first, the ego successfully waits for a sufficiently large gap in the oncoming traffic and switches to the adjacent lane. As long as the traffic cones marking the construction site are visible in the camera image, the model's path predictions correctly indicate a lane change back to the original lane. However, as shown in the final frame, once the traffic cones disappear from the camera image, the model erroneously predicts staying in the left lane. Notably, the traffic cones are still visible in the LiDAR image, suggesting an overreliance on the camera image for this scenario. After staying in the wrong lane,

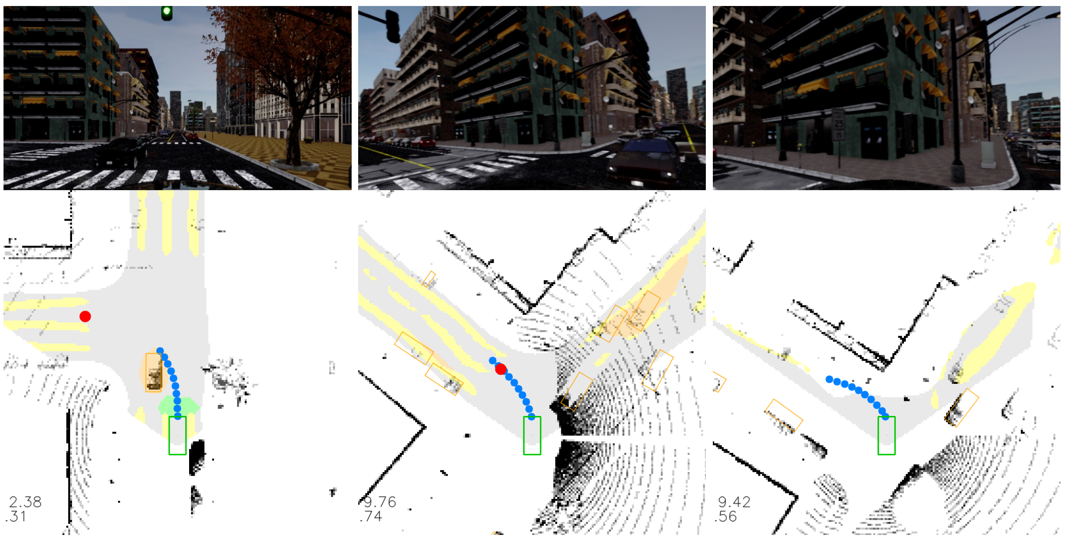


Figure 6.5: **Scenario: SignalizedJunctionLeftTurn.** Oncoming traffic is neglected in left turn after traffic light, leading to vehicle collision and subsequent successful recovery.

the ego often collides with an oncoming vehicle.

There are several potential solutions to this problem. First, a multi-frame model could use past camera frames, where the cones are visible, for path prediction. Second, adding a rear camera could help the model detect this situation in a single-frame setting. Third, incorporating a memory of past predictions could enable the model to recall the correct lane change from previous frames without explicitly using past inputs or additional sensors. This could be implemented in a probabilistic framework, treating past predictions as prior knowledge and calculating maximum a posteriori estimates for checkpoints using the predictions from the current frame.

SignalizedJunctionLeftTurn

This scenario, along with SignalizedJunctionRightTurn, NonSignalizedJunctionLeft-Turn and NonSignalizedJunctionRightTurn, belongs to a group of scenarios where the main challenge lies in negotiating with other vehicles in intersections. These scenarios are among the more difficult ones for our models, with average driving scores roughly ranging from 75 to 86. Figure 6.5 shows a common failure case where the ego does not react adequately to another actor, leading to a vehicle collision. After the traffic light switches to green, the vehicle accelerates and turns without reacting to the oncoming vehicle in the lane it needs to cross. Despite the collision, the ego demonstrates recovery abilities, returning to the road and completing the route. As noted in previous work with TransFuser++ [JCG23], these models often have a strong bias to steer towards the next target point, which may help the model recover also in this case.

It is worth noting that the bounding box of the other vehicle is predicted by the model (see the second frame), indicating that this is not a failure in the perception part of the model. Furthermore, as shown in the experiment with ground truth velocities as input (see Section 5.6), vehicle collisions are typically not due to an inability to judge other actors' speeds. Reliably solving this scenario may require additional data or focused fine-tuning for the model to learn to predict other vehicle's paths more robustly and thus avoid collisions. Potentially, a model-based approach with a world model that describes other actors' dynamics may also do better here.

VehicleTurningRoutePedestrian

In this scenario, the ego must make an unproteced turn through dense traffic and encounters a pedestrian on the road during or directly after the turn. This combination of hazards make it one of the most challenging scenarios in Leaderboard 2.0, where our models achieve an average driving score of 65.1.

Figure 6.6 shows two failure cases in this scenario. In the first case (top), the model fails to execute a turn through very dense traffic, where the margins for selecting the right moment to accelerate are extremely narrow. After colliding with a vehicle, the model also collides with a pedestrian that walks into the ego vehicle while it is stationary. The second example (bottom) depicts an instance of this scenario at night, where the ego does not collide with another vehicle, but fails to recognize the pedestrian hazard in time. Note that the pedestrian is barely visible until illuminated by the ego's headlights, which is only a couple of frames before the collision. This failure is likely due to covariate shift, since the expert would brake earlier in this situation, even before the pedestrian becomes visible in the RGB image. By the time the pedestrian is revealed by the headlights, the model is already outside its training distribution, where it has not learned a braking reflex. This issue could potentially be mitigated with on-policy methods like DAgger [RGB11], which involve querying the expert during failure states (such as when the pedestrian appears in the headlights) to teach the model how to respond appropriately in such situations.

In our view, this scenario may remain one of the most challenging, as it is hard to come up with simple adjustments that promise immediate improvements in dense-traffic turns. In addition to on-policy methods, potential solutions include adjusting the expert's behavior to accelerate faster through small gaps in traffic to avoid vehicle collisions (which would, however, require an even faster reaction to the pedestrian hazard) or fine-tuning the model on additional training data for these situations.

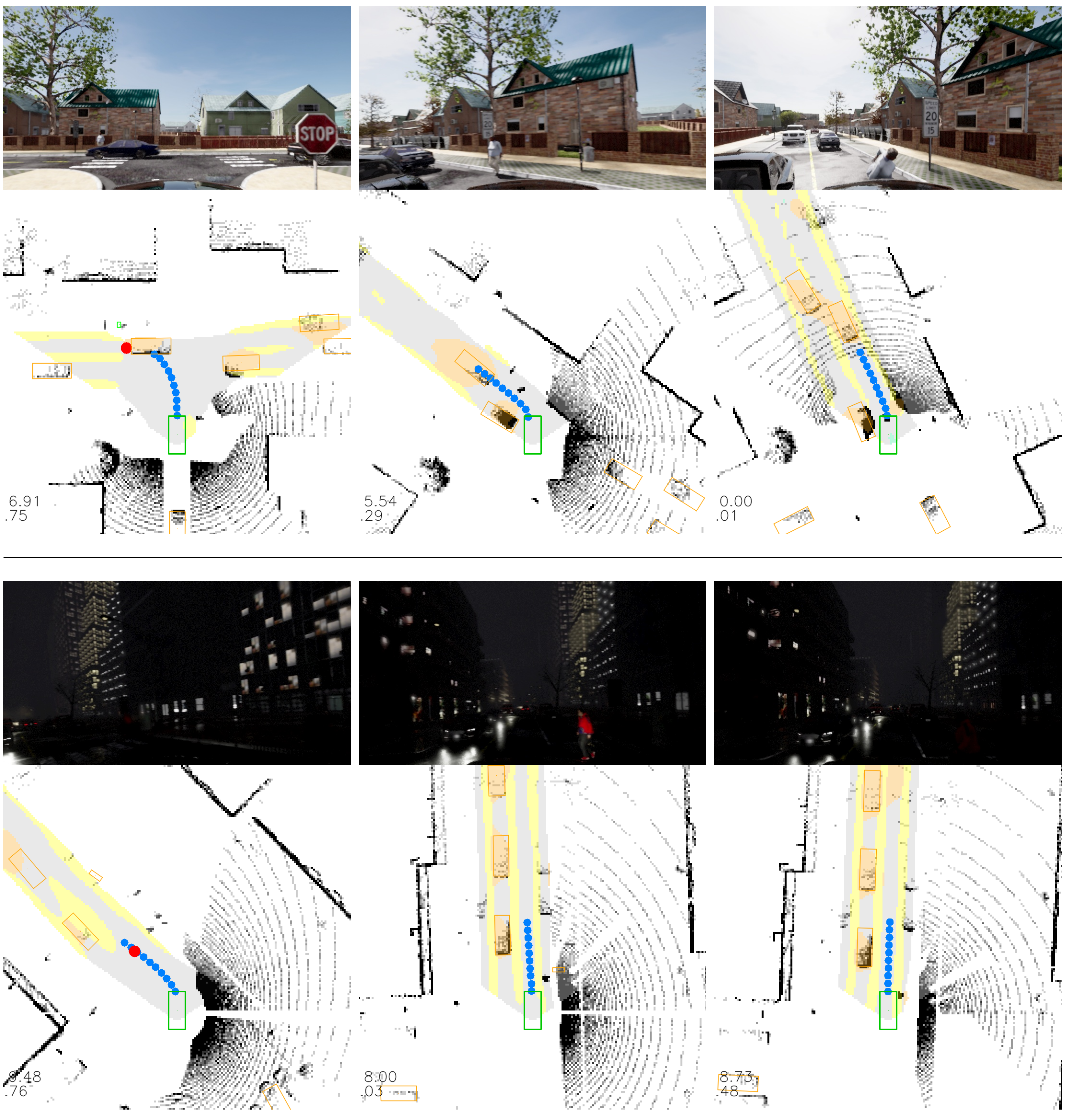


Figure 6.6: **Scenario: VehicleTurningRoutePedestrian.** Top: Failure to perform unprotected turn through dense traffic with crossing pedestrian, producing vehicle and pedestrian collision. Bottom: Failure to recognize pedestrian at night, braking response not triggered once pedetrian is visible in the headlights, producing collision.

Other scenarios

- **YieldToEmergencyVehicle.** In this new Leaderboard 2.0 scenario, the ego must yield to an emergency vehicle approaching from behind on a multi-lane highway. All our models fail in this scenario, since it is impossible to distinguish emergency vehicles from regular vehicles from the LiDAR image alone. Thus, solving this scenario requires the addition of a back camera.
- **Navigating in dense traffic.** Several other low-performance scenarios such as MergerIntoSlowTraffic(V2), HighwayExit, EnterActorFlow and others, require navigating in dense traffic with other vehicles in front of, next to, and/or behind the ego at varying speeds. Ours models are currently not able to handle these situations to a satisfying degree, frequently colliding with other actors. Adding more cameras to the rear or sides of the ego vehicle could potentially mitigate this issue at the cost of additional parameters and compute requirements. Moreover, many of these scenarios occur infrequently in the training routes as shown in Figure 4.1, leading to heavy upsampling of the existing instances in the training set. Collecting more data in these situations could thus improve perfomance. If additional training instances become available, it could further beneficial to finetune the model on such dense-traffic situations.

6.3 Future Research Directions

We propose several promising research directions aimed at developing more robust and safer models in CARLA. Drawing from our findings, we begin with targeted adaptations and then expand to broader, more general strategies.

- **Dataset Balancing and Expansion.** According to our analysis in the previous section, our models tend to underperform in scenarios with a low number of instances. Enhancing the dataset by manually creating and collecting data for these underrepresented scenarios, rather than relying solely on the official training and validation routes, could improve performance in rare situations. Additionally, scaling up the dataset, particularly by focusing on challenging situations such as lane changes in dense traffic, turns through narrow gaps in oncoming traffic, and low-light conditions, could improve model reliability.
- **Sensor Setup.** Integrating additional sensors, particularly rear and side cameras, could provide the model with a more comprehensive 360-degree perception of its environment. This is particularly crucial for scenarios like YieldToEmergencyVehicle, where the model must detect and respond to vehicles approaching from behind.
- **Temporal Context.** To address issues where the model fails to recall crucial past information, such as in the ConstructionObstacleTwoWays scenario, future models could benefit from integrating temporal context instead of basing

predictions on a single frame only. By utilizing past camera frames with video perception modules or a memory of past predictions, the model could remember and execute actions which are hard to predict from the current frame.

Looking ahead, we also explore broader concepts that could significantly enhance the performance of driving models in CARLA.

- **Mixture of Experts.** Instead of training a single model to handle all driving scenarios, it may be more effective to employ a mixture of expert models, each specialized in a specific mode of driving. These modes could range from broader categories like highway, interurban, and urban driving to more granular ones, such as expert models tailored to specific scenarios. Each expert could be fine-tuned from a base model using techniques like Low-Rank Adaptation (LoRA), which has recently shown success in language models [HSW$^+$21]. LoRA works by adjusting only a subset of parameters, making fine-tuning more efficient and targeted. In addition to training individual experts, this approach would require a weighting function to determine which expert to activate in a given situation. This function could be trained in a supervised manner, where the labels correspond to different driving modes or scenario types.
- **Learning from Experience.** In the future, models could improve by using more advanced learning methods that help them adapt and learn from experience, rather than just copying expert behavior (behavior cloning). As driving situations become more complex, simply mimicking experts may not be enough to handle all rare or unusual cases. Using reinforcement learning or other experience-based techniques could allow models to make better decisions in a wider range of situations, mitigating covariate shift. To reduce the high computational cost of online reinforcement learning, which requires expensive simulator interactions in the training loop, a possible solution is to use IL-bootstrapped RL [KST$^+$22]. In this approach, the RL model is initialized with an IL policy, essentially treating RL as an additional training stage after our current training routine.
- **Model-Based Methods.** Incorporating model-based methods, where the model learns to predict the environment and anticipate future states, could improve decision-making [HPBL24]. In CARLA, the behavior of other actors is relatively simple, with pedestrians typically moving at a constant speed without changes of direction, and vehicles moving in a predictable way according to hard-coded rules specified by the CARLA Traffic manager. In this setting, model-based methods seem suitable, as predicting the environment and especially other actors' behaviors is often one of the main challenges with these approaches. However, when applying model-based methods in a non-privileged agent using sensor inputs only, the model might still not be able to perceive all necessary information, such as traffic light states for oncoming traffic, making

prediction more challenging. Despite this, a robust world model could simulate possible outcomes of different actions, leading to more informed and safer driving decisions, particularly in dynamic and unpredictable situations not encountered in the training data, where foresight is crucial.

6.4 Conclusion

In this thesis, we have refined the entire machine learning pipeline for CARLA Leaderboard 2.0, creating a custom dataset and evaluation benchmark and identifying key architectural adaptations necessary for the new evaluation setting. We have demonstrated that expert driving style, beyond just expert performance, has a significant impact on the performance of downstream IL models. Additionally, we have found that using frequency-based class weights in target speed classification is detrimental due to the inherent inner-class diversity in frame importance and proposed a simple heuristic that effectively captures important frames by detecting changes in target labels.

Incorporating these findings, we have developed the top-performing open-source model for CARLA Leaderboard 2.0 to date, including ablation studies and qualitative failure analyses to further evaluate its strengths and limitations. Additionally, our investigation into the design of performance metrics revealed a flaw that incentivizes premature termination of evaluation routes. To address this, we have proposed an adjusted metric that eliminates this incentive for a fairer assessment of model performance in future challenges.

By publishing our dataset, benchmark, models, and code, we provide a toolkit for the community to build upon. We hope this work will serve as a starting point for addressing the issues we've identified and that future research will drive advances in leaderboard performance akin to those seen with Leaderboard 1.0, which would represent a significant step toward reliable end-to-end autonomous vehicles ready for widespread use.

Bibliography

[AD21] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

[AK21] Andrei Aksjonov and Ville Kyrki. Rule-based decision-making system for autonomous vehicles at intersections with mixed traffic environment. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 660–666, 2021.

[Bei24] Jens Beißwenger. Pdm-lite: A rule-based planner for carla leaderboard 2.0. Technical report, University of Tübingen, 2024. `https://github.com/autonomousvision/carla_garage/blob/leaderboard_2/doc/report.pdf`.

[BGC+19] Claudine Santos Badue, Rânik Guidolini, Raphael V. Carneiro, Pedro Azevedo, Vinicius B. Cardoso, Avelino Forechi, Luan F. R. Jesus, Rodrigo Berriel, Thiago Meireles Paixão, Filipe Wall Mutz, Thiago Oliveira-Santos, and Alberto Ferreira de Souza. Self-driving cars: A survey. *arXiv*, 1901.04407, 2019.

[BS95] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, 1995.

[CBL+20] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[CK22] Dian Chen and Philipp Krähenbühl. Learning from all vehicles. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17201–17210, 2022.

[CKT+22] Holger Caesar, Juraj Kabzan, Kok Seang Tan, Whye Kit Fong, Eric Wolff, Alex Lang, Luke Fletcher, Oscar Beijbom, and Sammy Omari. Nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles. *arXiv*, 2106.11810, 2022.

[CPJ+22] Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. Transfuser: Imitation with transformer-based sensor fusion for autonomous driving. *arXiv*, 2205.15997, 2022.

[CUS⁺21] Jonathan Chang, Masatoshi Uehara, Dhruv Sreenivas, Rahul Kidambi, and Wen Sun. Mitigating covariate shift in imitation learning via offline data with partial coverage. In *Advances in Neural Information Processing Systems*, volume 34, pages 965–979. Curran Associates, Inc., 2021.

[CWC⁺24] Li Chen, Penghao Wu, Kashyap Chitta, Bernhard Jaeger, Andreas Geiger, and Hongyang Li. End-to-end autonomous driving: Challenges and frontiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–20, 2024.

[DBX⁺19] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6568–6577, 2019.

[DRC⁺17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 13–15 Nov 2017.

[FOV⁺24] Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, Aviral Kumar, and Rishabh Agarwal. Stop regressing: Training value functions via classification for scalable deep rl. *arXiv preprint*, 2403.03950, 2024.

[Gam] Epic Games. Unreal engine. `https://www.unrealengine.com/`. Accessed: 2024-09-06.

[GKM⁺20] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S. Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, Tiffany Fernandez, Martin Jänicke, Sudesh Mirashi, Chiragkumar Savani, Martin Sturm, Oleksandr Vorobiov, Martin Oelker, Sebastian Garreis, and Peter Schuberth. A2d2: Audi autonomous driving dataset. *arXiv*, 2004.06320, 2020.

[HE16] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[HPBL24] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap.

Mastering diverse domains through world models. *arXiv*, 2301.04104, 2024.

[HSW+21] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv*, 2106.09685, 2021.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[JCG23] Bernhard Jaeger, Kashyap Chitta, and Andreas Geiger. Hidden biases of end-to-end driving models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8206–8215, Los Alamitos, CA, USA, oct 2023. IEEE Computer Society.

[KST+22] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2022.

[KWB+21] Liyiming Ke, Jingqiang Wang, Tapomayukh Bhattacharjee, Byron Boots, and Siddhartha Srinivasa. Grasping with chopsticks: Combating covariate shift in model-free imitation learning for fine manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6185–6191, 2021.

[Lea] CARLA Autonomous Driving Leaderboard. `https://leaderboard.carla.org/`. Accessed: 2024-09-06.

[LH17] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

[LH19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv*, 1711.05101, 2019.

[LLL+24] Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. Llava-next: Improved reasoning, ocr, and world knowledge. `https://llava-vl.github.io/blog/2024-01-30-llava-next/`, January 2024.

[MGK+17] Rowan McAllister, Yarin Gal, Alex Kendall, Mark van der Wilk, Amar Shah, Roberto Cipolla, and Adrian Weller. Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 4745–4753, 2017.

[MYDM22] Luc Mero, Dewei Yi, Mehrdad Dianati, and Alexandros Mouzakitis. A survey on imitation learning techniques for end-to-end autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23:1–20, 09 2022.

[Org23] World Health Organization. *Global status report on road safety 2023*. World Health Organization, 2023.

[PAdNdLF17] Philip Polack, Florent Altché, Brigitte d'Andréa Novel, and Arnaud de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 812–818, 2017.

[PBOB$^+$20] Aditya Prakash, Aseem Behl, Eshed Ohn-Bar, Kashyap Chitta, and Andreas Geiger. Exploring data aggregation in policy learning for vision-based urban autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[PSL$^+$24] Jongjin Park, Younggyo Seo, Chang Liu, Li Zhao, Tao Qin, Jinwoo Shin, and Tie-Yan Liu. Object-aware regularization for addressing causal confusion in imitation learning. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2024. Curran Associates Inc.

[RCM$^+$24] Katrin Renz, Long Chen, Ana-Maria Marcu, Jan Hünermann, Benoit Hanotte, Alice Karnsund, Jamie Shotton, Elahe Arani, and Oleg Sinavski. Carllava: Vision language models for camera-only closed-loop driving. *arXiv*, 2406.10165, 2024.

[RGB11] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[RGdS$^+$24] Luis Alberto Rosero, Iago Pachêco Gomes, Júnior Anderson Rodrigues da Silva, Carlos Andre Braile Przewodowski Filho, Denis Fernando Wolf, and Fernando Santos Osório. Integrating modular pipelines with end-to-end learning: A hybrid approach for robust and reliable autonomous driving systems. *Sensors*, 2024.

[RKG$^+$20] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10425–10433, 2020.

[SGS$^+$23] Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and

Ari S. Morcos. Beyond neural scaling laws: beating power law scaling via data pruning. *arXiv*, 2206.14486, 2023.

[SKD+20] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Sheng Zhao, Shuyang Cheng, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. *arXiv*, 1912.04838, 2020.

[SWC+22] Hao Shao, Letian Wang, RuoBing Chen, Hongsheng Li, and Yu Liu. Safety-enhanced autonomous driving using interpretable sensor fusion transformer. *arXiv*, 2207.14024, 2022.

[THH00] Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical review E*, 2000.

[TMS+22] Ardi Tampuu, Tambet Matiisen, Maksym Semikin, Dmytro Fishman, and Naveed Muhammad. A survey of end-to-end driving: Architectures and training methods. *IEEE Transactions on Neural Networks and Learning Systems*, 33(4):1364–1384, 2022.

[TPJR18] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, page 303–314, New York, NY, USA, 2018. Association for Computing Machinery.

[WJC+22] Penghao Wu, Xiaosong Jia, Li Chen, Junchi Yan, Hongyang Li, and Yu Qiao. Trajectory-guided control prediction for end-to-end autonomous driving: A simple yet strong baseline. *arXiv*, 2206.08129, 2022.

[WLQ+21] Chuan Wen, Jierui Lin, Jianing Qian, Yang Gao, and Dinesh Jayaraman. Keyframe-focused visual imitation learning. *arXiv*, 2106.06452, 2021.

[WQA+21] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, Deva Ramanan, Peter Carr, and James Hays. Argoverse 2: Next generation datasets for self-driving perception and forecasting. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

[XMC+21] Wei Xiao, Noushin Mehdipour, Anne Collin, Amitai Y. Bin-Nun, Emilio Frazzoli, Radboud Duintjer Tebbens, and Calin Belta. Rule-based optimal control for autonomous driving. In *Proceedings of* *the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, ICCPS '21, page 143–154, New York, NY, USA, 2021. Association for Computing Machinery.

[YLCT20] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020.

[YXP+22] Shuo Yang, Zeke Xie, Hanyu Peng, Minjing Xu, Mingming Sun, and P. Li. Dataset pruning: Reducing training data by examining generalization influence. *arXiv*, 2205.09329, 2022.

[ZC17] Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end simulated driving. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 2891–2897. AAAI Press, 2017.

[ZER+24] Weize Zhang, Mohammed Elmahgiubi, Kasra Rezaee, Behzad Khamidehi, Hamidreza Mirkhani, Fazel Arasteh, Chunlin Li, Muhammad Ahsan Kaleem, Eduardo R. Corral-Soto, Dhruv Sharma, and Tongtong Cao. Analysis of a modular autonomous driving architecture: The top submission to carla leaderboard 2.0 challenge. *arXiv*, 2405.01394, 2024.