Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Artificial Intelligence

End-to-end Autonomous Driving in Simulated Environments

Master's Thesis

Author
Péter Gyula Gyömbér

Advisor
dr. Bálint Gyires-Tóth
András Béres

June 1, 2025

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Gyömbér Péter Gyula*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2025. június 1.

Gyömbér Péter Gyula
hallgató

Kivonat

A diplomatervem az end-to-end önvezető algoritmusok fejlesztésével, illetve szimulátorokban történő kiértékelésével foglalkozik. A munkám során nagy hangsúlyt kap a NAVSIM szimulációs keretrendszer, ami egy fejlett nyílt hurok (open-loop) alapú teljesítménymérő keretrendszer. A NAVSIM számos újonnan felmerülő promlémára nyújt megoldást, többek között az adathalmaz kiegyensúlyozatlanságára, valamint a mérési metrikák hiteltelenségére. Feltételezve, hogy az önvezető jármű rövid idő alatt nem befolyásolja a környezetét, olyan metrikákat is alkalmaz, amik jól közelítik a zárt hurok (closed-loop) alapú szimulátorokban használtakat.

A dolgozat során részletesen bemutatom a használt szimulációs keretrendszert, beleértve az adatok előkészítését, az ágensek létrehozását, illetve a kiértékelési módszertanokat. Az elérhető alapmodellek ismertetése mellett egyedi modellek fejlesztését is bemutatom, amelyek különféle megközelítéseken alapszank. Ide tartoznak például a transzformer-fúzió alapú modellek, valamint a nagy nyelvi modellekre (LLMs) épülő megoldások.

Az eredmények vizsgálata betekintést nyújt az önvezető autók tervezése során felmerülő aktuális kihívásokba és azok lehetséges megoldásaiba. Továbbá a dolgozatban újfajta irányelveket is elemezek, amik későbbi kutatások alpjait képezhetik. A szimulátorokban történő kiértékelés fontos előfeltétele az önvezető algoritmusok valós környzetekben történő alkalmazhatóságának. Elengedhetetlen tehát fejlett keretrendszerek használata, amik megfelelően támogatni tudják a biztonságos és hatékony algortimusok fejlesztését.

Abstract

This thesis focuses on the development of end-to-end autonomous driving algorithms and their evaluation in simulators. I place a key emphasis on the NAVSIM simulation framework, which is an advanced open-loop benchmarking framework. NAVSIM provides solutions to several emerging issues, including dataset imbalances and the credibility of evaluation metrics. Assuming that the autonomous vehicle does not significantly affect its environment within a short time, it uses metrics that closely approximate those used in closed-loop simulators.

The thesis provides an in-depth presentation of the simulation framework, including data preparation, agent implementation, and evaluation methodologies. In addition to reviewing baseline models available in the framework, in this study, I introduce several custom-designed agents that reflect different modeling strategies, including transformer-based fusion models and experimental approaches leveraging large language models (LLMs).

Through comparative evaluation of these agents, the thesis offers valuable insights into key challenges in designing autonomous driving algorithms and explores possible directions for their resolution. Furthermore in this thesis, I discuss novel perspectives that could serve as foundations for future research. Evaluation in simulation is a critical step toward real-world deployment of autonomous systems, therefore, the adoption of advanced and reliable frameworks is essential to support the development of safe and effective driving algorithms.

Chapter 1

Introduction

Autonomous driving aims to enable vehicles to navigate and make decisions without human intervention. This can be achieved through two primary approaches: modular systems, where tasks are handled separately, and end-to-end systems, which integrate all tasks into a single process. Modular systems mostly follow rule-based methods to handle various scenarios including perception, planning, and control [49]. Dealing with rare scenarios and unseen traffic conditions presents a significant challenge for these systems. End-to-end solutions simplify the process by removing intermediate steps and focusing on predicting future vehicle control outputs directly from input data in a continuous manner. The increasing interest in autonomous driving research also calls for the development of more reliable evaluation methods to compare different approaches. It is extremely challenging to properly evaluate the performance of self-driving agents and many of the widely-used evaluation benchmarks fall short in several aspects [15]. The NAVSIM[1] framework was introduced in recent years to take a step ahead, and overcome many of the existing limitations experienced in other simulators. It seeks to combine the strengths of both open-loop and closed-loop benchmarking metrics, resulting in a computationally efficient, non-reactive simulation environment that provides representative scoring outcomes.

This thesis investigates state-of-the-art end-to-end autonomous driving approaches using the NAVSIM simulation framework. The rationale for selecting NAVSIM is supported by a comparative analysis with other available simulators, highlighting its advantages. Particular attention is given to the differences between open-loop and closed-loop evaluation methods, which is an important topic in general, but especially relevant in the context of NAVSIM, as it integrates features of both approaches. Once the motivation for this choice is established, the framework is introduced in detail, with a focus on its dataset and scoring function. Later chapters cover the implementation process, demonstrating how new end-to-end models can be integrated into the framework. NAVSIM includes several baseline agents ready for training and evaluation. Building on these foundations, custom agents will be developed with the goal of improving overall performance. While setting up a basic model is relatively straightforward, reaching state-of-the-art performance remains a considerable challenge that this work aims to address.

[1]NAVSIM is a Data-Driven Non-Reactive Autonomous Vehicle Simulation and Benchmarking framework. Currently available at: `https://github.com/autonomousvision/navsim`

The *Autonomous Grand Challenge*[2] was hosted at the CVPR (Conference on Computer Vision and Pattern Recognition) workshop in 2024, where the category *End-to-End Driving at Scale* utilized the NAVSIM framework. The winning team's solution [36], built upon the TransFuser [10] baseline, showcasing the power of this base model. In addition to the TransFuser-based models, a custom model leveraging Large Language Models (LLMs) will also be introduced.

Finally, a dedicated chapter focuses on the evaluation of the selected models, where their performance is compared across key metrics. Due to the scope limitations of this thesis, only a subset of top-performing models is included in the benchmarking process. However, the implementation approach and methodology are thoroughly documented, providing transferable insights that can support the adaptation and integration of additional models in future research.

***Thesis Structure Overview*:** This thesis is organized to systematically expand upon the key ideas introduced above. Following this introductory section, the content is divided into four main chapters, each dedicated to a central aspect of the research. The **Related Work** chapter provides the necessary background, covering existing algorithmic approaches, simulator frameworks, and relevant regulations. Chapter **NAVSIM - The Benchmarking Framework** presents an in-depth exploration of the NAVSIM framework, beginning with a high-level overview and proceeding to detailed technical aspects. The **Proposed Work** chapter discusses the development, training, and evaluation of both baseline and custom models within the NAVSIM environment. Finally, the **Evaluation Methods and Results** chapter evaluates the performance of the selected models, drawing comparisons and summarizing the key findings from the experimental results.

[2]Autonomous Grand Challenge is a competition, focusing on Foundation Models for Autonomous Systems. More details can be found here: `https://opendrivelab.com/challenge2024/`

Chapter 2

Related Work

2.1 Autonomous Driving Approaches

Autonomous driving systems can be developed using a variety of architectural paradigms, each reflecting different trade-offs between interpretability, flexibility, and performance. Traditional approaches rely on modular pipelines that break down the task into discrete stages such as perception, prediction, planning and control. In contrast, end-to-end methods leverage data-driven models to learn driving behavior directly from raw sensory inputs, often improving adaptability and scalability. Hybrid approaches also exist, aiming to combine the strengths of both worlds.

2.1.1 Traditional Modular Pipelines

Modular systems structure autonomous driving as a sequential pipeline composed of distinct, interconnected modules. These modules collectively handle the full driving task by dividing it into subtasks such as localization and mapping, perception, prediction, decision-making, planning, control, and human-machine interaction.

The pipeline typically begins with processing raw sensor data (e.g., cameras, LiDAR[1], radar) through localization and perception modules, followed by scene understanding, trajectory prediction, and decision-making logic. Ultimately, control modules translate planned actions into motor commands to guide the vehicle.

One of the core strengths of modular approaches is that they leverage decades of research in specialized fields like robotics, computer vision, and vehicle dynamics. This makes the system highly interpretable, facilitates debugging, and allows reuse of established algorithms. Additionally, their composable nature supports flexibility and redundancy, for example, safety layers or rule-based overrides can be added atop complex planning modules without modifying their internal logic.

However, modular pipelines face key challenges such as error propagation, where upstream errors (e.g., misdetections) cascade through the system, and engineering over-complexity as module interfaces must be properly tuned [52, see p. 3–4].

[1]Light Detection and Ranging (LiDAR) is a sensor technology that measures distances by emitting laser pulses and capturing their reflections to create precise 3D maps of the environment.

2.1.2 End-to-end Autonomous Driving

End-to-end systems aim to unify the entire autonomous driving pipeline (perception, prediction and planning) into a single, jointly trainable model. This design stands in contrast to classical modular systems and offers several advantages. It simplifies the architecture by consolidating all components into one model, reducing overall system complexity. Since the whole model is optimized toward the final driving task, it ensures better task alignment. Computational efficiency is improved through shared backbones, and system performance can scale effectively with more data and larger models [9, see p. 1–2].

While modular systems often suffer from mismatched optimization goals across stages, end-to-end training attempts to resolve this misalignment by optimizing everything toward a unified target. Although it is often viewed as black-box model, many state-of-the-art end-to-end systems include interpretable intermediate outputs, combining transparency with the benefits of joint training. However, in end-to-end systems it can be harder to localize the source of errors [9, see p. 1–2].

End-to-end approaches are typically categorized into two main paradigms: Reinforcement Learning (RL) and Imitation Learning (IL) [9, see p. 3].

2.1.2.1 Reinforcement Learning

Reinforcement Learning (RL) [27] is a learning paradigm where agents improve their behavior through trial and error by interacting with an environment and receiving feedback in the form of rewards. In autonomous driving, RL trains policies to maximize long-term returns, often using deep neural networks. However, due to its high data demands and safety risks, RL is mostly applied in simulation, and has yet to outperform imitation learning in complex real-world driving tasks [9, see p. 4].

2.1.2.2 Imitation Learning

Imitation Learning (IL) [53], also known as learning from demonstrations, trains an agent to mimic the behavior of an expert. It requires a dataset, where each trajectory is a sequence of state-action pairs, collected under the expert's policy π^*. The objective is to learn a policy π that closely matches π^* . This policy can output either planned trajectories or direct control signals [9, see p. 3].

Early IL methods typically predicted control outputs due to ease of data collection. However, predicting controls at individual time steps often leads to discontinuous maneuvers and ties the learned policy to a specific vehicle's dynamics, limiting generalization. As an alternative, some approaches predict intermediate waypoints or trajectories, which extend over longer horizons and require separate controllers to convert them into control commands [9, see p. 3].

A prominent IL approach is **behavior cloning (BC)** [2], which frames the task as supervised learning. It is advantageous due to its simplicity and efficiency. However, there are some common issues. During training, it treats each state as independently and identically distributed (which differs from the distribution of the testing data),

resulting in an important problem known as covariate shift. To address this, DAgger (Dataset Aggregation) [41] has been adopted, which first lets the agent act, then it collects expert corrections on its behavior to iteratively retrain on states it is likely to encounter. Another common problem with BC is causal confusion, where the imitator exploits and relies on false correlations between certain input components and output signals [9, see p. 4].

Another family of methods includes Inverse Optimal Control (IOC), also known as Inverse Reinforcement Learning (IRL) [56], where the agent aims to recover the underlying reward function that explains the expert behavior, often modeling it as a linear combination of features. However, in complex, high-dimensional tasks like autonomous driving, explicitly defining or optimizing such a reward is difficult. Generative Adversarial Imitation Learning (GAIL) [24] addresses this by framing the reward learning as an adversarial game, training a discriminator to distinguish between expert and agent behavior, similar to Generative Adversarial Networks (GANs). More recent approaches extend this by learning cost functions optimized alongside auxiliary tasks, treating trajectory selection as a cost minimization problem [9, see p. 4].

Driving with Large Language Models (LLMs) [49] is a recent and promising direction within the domain of Imitation Learning for autonomous driving. Leveraging the rapid advancements of LLMs, these models can be applied to specific sub-tasks such as route reasoning, decision-making in ambiguous situations, or interpreting high-level instructions. Their ability to generalize across diverse contexts and incorporate world knowledge makes them a valuable tool, particularly when combined with traditional perception and control modules. While still in early stages, integrating LLMs introduces a new paradigm for designing interpretable and flexible decision-making systems in autonomous driving.

2.2 Simulation Frameworks

Evaluating end-to-end driving models requires reliable simulation frameworks that can accurately reflect real-world complexities. Simulation not only provides a safe and cost-effective environment for testing but also enables large-scale, repeatable experiments. This section outlines the simulation approaches commonly used for benchmarking autonomous driving systems.

2.2.1 Open-loop vs Closed-loop Evaluation

Benchmarking frameworks can generally be categorized into open-loop and closed-loop evaluations, each offering distinct advantages and challenges. Understanding the differences between these two evaluation setups is crucial for selecting the appropriate method depending on the goals of the study.

2.2.1.1 Open-loop Benchmarking

Open-loop metrics assess a system's performance in a non-reactive manner. Once initial conditions or input data are provided, the experiment proceeds without altering the environment's state or receiving new information throughout the test. For benchmarking end-to-end autonomous systems, open-loop metrics can be used to evaluate performance under fixed traffic conditions represented by sensor data, such as camera images or LiDAR point clouds, along with ego[2]-state information, like speed, acceleration, and heading direction. In end-to-end systems, the desired output is typically a trajectory (often represented as a sequence of waypoints) or a set of ego control commands that guide the vehicle to achieve optimal driving behavior. Scores can be calculated by comparing the predicted outputs with the desired outputs. From this point forward, the outputs will be referred to as trajectories, since NAVSIM operates with sequences of waypoints, although driving commands could also be utilized. The optimal trajectory (also referred to as the ground truth) is often represented by a human expert trajectory, which is defined during the data preparation phase.

The advantage of open-loop benchmarking is that agents can be evaluated based on well-representative sensor data collected during real-world driving conditions. Another benefit of rule-based approaches is their scalability, as they can handle increased data efficiently while remaining computationally effective. However, the main drawback is that, in this case, the agent cannot interact with other traffic participants, leaving an important aspect of the benchmarking process unanswered. Comparing the predicted trajectory directly with the path of a human expert can be misleading, as it considers only a single trajectory as the optimal solution. In reality, multiple optimal trajectories can exist for a given traffic condition. As a result, this benchmarking method may poorly evaluate models that perform well, but do not follow the predefined trajectory exactly. In addition, several issues concerning evaluation with displacement errors have surfaced recently, particularly on the nuScenes [4] dataset. Furthermore, nuScenes lacks standardized planning metrics, leading to inconsistencies when comparing results across different models and experiments [15, see p. 3].

2.2.1.2 Closed-loop Benchmarking

Closed-loop evaluation involves creating a graphics-based rendered environment where traffic participants follow predefined behaviors. This environment acts like a playground for the ego-vehicle, enabling real-time interactions to be provisioned and visualized. By placing the ego-vehicle in such an environment, it becomes possible to collect downstream driving statistics, including collision rates, traffic rule compliance, and driving comfort. In sensor simulators like CARLA [16] or MetaDrive [35] there is a domain gap in visual fidelity and sensor characteristics compared to real-world conditions, which affects the accuracy of model training and evaluation. Data-driven simulators designed for motion planning, such as nuPlan [30], incorporate real-world traffic recordings but lack support for image or LiDAR-

[2]Ego refers to the self-driving vehicle, also used as the main point of reference.

based approaches, which limits their applicability for end-to-end perception and planning tasks that rely on raw visual or sensory inputs [15, see p. 3].

In conclusion, closed-loop evaluation effectively measures how the ego-vehicle interacts with and responds to the dynamics of its environment, making it particularly suited for detailed scoring functions that capture various aspects of the driving agent's performance. Graphical simulators also allow for the creation of rare and challenging traffic scenarios, which is valuable for testing the robustness of autonomous agents. However, the trade-off is that closed-loop evaluation cannot rely on real-world data, potentially leading to discrepancies when applying these agents in real-life situations. While promising methods for synthetic image [44] and LiDAR [38] data generation exist, achieving accurate and efficient simulation of sensors purely from data remains an open challenge in the field.

2.2.2 The Simulator Landscape

Selecting the right simulator for benchmarking autonomous driving agents is a non-trivial task. Algorithms, ranging from classic modular pipelines to end-to-end systems, such as reinforcement learning, and imitation learning, can perform differently depending on the simulation environment. As these algorithms evolve, so too must the simulators, to accurately reflect their real-world driving performance. However, both open-loop and closed-loop simulators have their own limitations, even among widely adopted frameworks. A key challenge remains the simulation-to-reality gap, where agent behavior diverges between simulated and real-world settings. This section introduces alternative open-source, actively maintained simulators that can be used in place of NAVSIM for research purposes. Evaluating algorithms across multiple simulation environments is often beneficial, particularly with newer simulators such as NAVSIM and Bench2Drive [29]. These simulators offer notable improvements over platforms like CARLA or nuPlan, aiming for greater reliability and realism.

2.2.2.1 CARLA

CARLA (Car Learning to Act) is an open-source simulator designed for autonomous driving research, with a primary focus on benchmarking driving in urban environments. Introduced in 2017, it has become one of the most widely used closed-loop frameworks in the field. From the beginning, CARLA was developed to support the training and validation of autonomous systems in complex urban settings [16, see p. 1–2].

The simulation platform allows for a flexible configuration of sensor suites and provides a variety of signals that can be used to train driving strategies, including Global Positioning System (GPS) coordinates, speed, acceleration, and detailed data on collisions and other infractions. Users can specify a wide range of environmental conditions, such as weather and time of day. Additionally, CARLA offers digital assets specifically created for the simulation, enabling detailed customization of the virtual environment [16, see p. 2–3].

CARLA grants full control over both static and dynamic actors, map generation, and many other features. It is well-suited for benchmarking the performance of both classic modular pipelines and end-to-end deep neural networks, which directly map sensory inputs to driving commands [16, see p. 2–3].

The platform follows a client-server architecture, where the server handles environment rendering and provides sensor data to the client. In turn, the client can send driving commands to the server to update the agent's state, as well as meta-commands to modify the simulator's behavior, reset the environment, or adjust environmental properties and sensor setups. CARLA supports multiple sensor modalities, including RGB cameras, LiDAR, radar, and much more [16, see p. 3].

2.2.2.2 MetaDrive

MetaDrive is a driving simulation platform designed to support research on generalizable reinforcement learning (RL) algorithms. Its key feature is compositionality, enabling the generation of an infinite number of diverse driving scenarios through procedural generation (PG), real-world dataset importing (such as Waymo [17] and Argoverse [5]), and domain randomization. This makes MetaDrive especially suited for studying RL generalization across varied environments [35, see p. 1–2].

MetaDrive supports both single-agent and multi-agent settings. It also emphasizes a computationally efficient trade-off between visual rendering and physical simulation, making it lightweight compared to more graphically intensive simulation platforms [35, see p. 1–3].

Scenario composition in MetaDrive follows a hierarchical structure: the environment creates managers based on user specifications, which then spawn objects and assign them policies. During simulation, these objects act autonomously while managers monitor their states, handle object recycling, and trigger new object creation as needed [35, see p. 4].

The simulator provides diverse sensory inputs, including low-level sensors like LiDAR, RGB cameras and depth cameras. These sensors are customizable and adjustable for noise, field of view, and laser count. Additionally, MetaDrive can supply high-level scene data, such as road geometry (bending angle, direction, length) and nearby vehicle attributes (velocity, heading, profile). However, MetaDrive's rendering is not as photorealistic as other simulators like CARLA, and it does not support traffic participants such as pedestrians and bicyclists [35, see p. 8, 12].

2.2.2.3 NuPlan

NuPlan is a large-scale autonomous driving simulator designed to evaluate machine learning-based planners. It features a real-world dataset with 1,282 hours of driving scenarios collected from four cities (Boston, Pittsburgh, Las Vegas, and Singapore), focusing on diverse and challenging situations. NuPlan uses manually driven data to ensure natural driving behavior and high-quality offline labeling to improve perception accuracy. This offline perception system can automatically label objects and capture traffic light statuses. The framework provides LiDAR point clouds and sen-

sor images, as well as detailed human-annotated 2D high definition semantic maps of the driving locations [30, see p. 1–3].

The simulation framework provided by nuPlan is modular and flexible, enabling to work with different datasets and setups. NuPlan supports both open-loop and closed-loop simulation modes. The simulation is initialized with the real-world observations captured in the dataset, then an agent model can be used to predict the future trajectories of all agents. In the last stage, a controller converts the selected route into a feasible trajectory [30, see p. 4].

The simulation can either playback the actions recorded in the dataset (open-loop) or allow the simulation to deviate from the recording by incorporating the ego's actions (closed-loop). In closed-loop settings, it includes reactive agent models based on the Intelligent Driver Model (IDM) [1] to simulate realistic agent behaviors. The simulator evaluates planners using a comprehensive set of metrics for safety, compliance, progress, and comfort, normalized to a $0 - 1$ range [30, see p. 4–5].

2.2.2.4 Waymax

Waymax is a data-driven, hardware-accelerated simulator designed for autonomous driving research, focusing on realistic, multi-agent scenarios. It leverages real-world driving data, primarily from the Waymo Open Motion Dataset [17], to initialize diverse driving situations, including interactions with pedestrians, cyclists, and traffic signals. By combining both learned and hard-coded behavior models, Waymax enhances simulation realism and supports large-scale simulation with machine learning integration. It is a differentiable closed-loop simulator, which enables efficient benchmarking of imitation and reinforcement learning algorithms for autonomous vehicle planning [20, see p. 1–2].

Unlike perception-focused simulators, Waymax specifically addresses the driving policy modeling problem, prioritizing behavior over sensor simulation. Consequently, it does not support sensor data simulation, as it is designed for decision-making and control tasks. This approach allows for standard training and evaluation workflows, providing reliable benchmarking in both open and closed-loop settings. The framework provides a comprehensive set of metrics for evaluating safety, comfort, and progress, including collision detection, route progress, and kinematic feasibility [20, see p. 2–5].

Waymax's architecture is modular and flexible, making it highly customizable for research applications. The core libraries consist of (1) a set of common data structures, (2) a distributed data-loading library, (3) simulator components including metrics and dynamics, and (4) a Gym-like environment interface. These components are designed to be interchangeable and can also function independently, giving users the freedom to modify or extend the simulator according to their specific needs [20, see p. 2–6].

2.2.2.5 Bench2Drive

Bench2Drive is a large-scale benchmark for evaluating end-to-end autonomous driving systems in a closed-loop setting. It includes a richly annotated dataset collected in the CARLA simulator, offering diverse traffic scenarios and detailed performance analysis. The framework supports both low-level sensor inputs such as LiDAR, cameras, and radar, and high-level sensor information, including a Bird's Eye View (BEV) camera and access to High-Definition maps[3] (HD maps) [29, see p. 1, 5].

The dataset contains over 2 million frames across 13,638 short clips, each about 150 meters long and segmented by driving scenarios. This segmentation supports the learning of specific driving skills, categorized into five high-level abilities: Merging, Overtaking, Emergency Braking, Traffic Sign Compliance, and Giving Way. Data was collected using Think2Drive [34], a reinforcement learning-based expert agent capable of solving all the benchmarked scenarios. The dataset ensures uniform distribution across 23 weather conditions, 12 towns, and all scenario types, avoiding biases common in other datasets. To support different research scales, it offers three subsets: mini, base, and full, ranging from debugging-scale to large-scale training [29, see p. 2, 5–6].

Evaluation is conducted over 220 short routes, with five variations per scenario to test different towns and weather. Each route targets one of 44 specific driving situations, allowing precise and low-variance measurement of driving capabilities. Two main metrics are used: Success Rate, which captures whether routes are completed without infractions, and Driving Score, which considers completion and penalizes errors based on severity. Bench2Drive introduces Efficiency and Smoothness as additional metrics. Efficiency measures the ego vehicle's speed compared to surrounding traffic, while Smoothness measures jerk, acceleration, and yaw-related metrics [29, see p. 6–8].

2.2.3 Comparison and Overview

Table 2.1 compares the simulation frameworks introduced, outlining their key characteristics. CARLA, MetaDrive, and Bench2Drive do not support real-world data integration, leaving the simulation-to-reality gap unaddressed. However, they excel at modeling interactions and traffic dynamics, which is crucial for evaluating autonomous agents in complex scenarios. MetaDrive is a lightweight simulator that allows rapid creation of diverse driving environments, though its limited rendering fidelity places it further from real-world realism compared to CARLA. CARLA remains a widely used and actively maintained closed-loop simulator since 2017, making it a reliable choice. Bench2Drive, built on top of CARLA, introduces a unified training dataset and offers fine-grained scenario-based benchmarking, establishing itself as a promising next-generation tool for closed-loop evaluation. In contrast, nuPlan and Waymax support real-world data integration and closed-loop execution, but primarily target planning rather than full end-to-end driving. Even though they support both open- and closed-loop modes, the open-loop setup still relies on non-

[3] HD maps are high-resolution maps that provide detailed information about the road environment.

reactive agents. NAVSIM, built upon nuPlan, addresses some limitations of earlier simulators (these limitations will be discussed later). It supports real-world data integration and is specifically designed for benchmarking end-to-end driving agents using rich, multi-sensor input.

Simulator	Released	Real Data	Closed-Loop	Sensor	E2E-Sim
CARLA	2017		✓	✓	✓
MetaDrive	2021		✓	✓	✓
NuPlan	2021	✓	✓	✓	
Waymax	2023	✓	✓		
Bench2Drive	2024		✓	✓	✓
NAVSIM	2024	✓		✓	✓

Table 2.1: Comparison of popular, actively maintained, open-source simulation frameworks.

Open-loop and closed-loop simulators each offer distinct advantages and limitations, making it advisable to evaluate self-driving agents in both setups before considering real-world deployment. Among the most widely used frameworks, Bench2Drive (closed-loop) and NAVSIM (open-loop) were found as excellent options. Both are built upon mature simulators (CARLA and nuPlan) and address key shortcomings observed in earlier systems, offering more reliable and insightful benchmarking capabilities.

For the purposes of this work, NAVSIM is selected to evaluate the performance of different end-to-end driving models. A detailed overview of NAVSIM's architecture, features, and advantages will be provided in a later chapter.

2.2.4 Simulation-to-Real: Safety Considerations

One of the fundamental challenges in autonomous driving is closing the simulation-to-reality gap, the difference between how an autonomous driving (AD) system performs in simulation versus in the real world. Simulations struggle to fully replicate the complexities and unpredictability of real-world driving conditions, leading to the risk that an AD system might perform well in simulation but fail in reality due to unmodeled factors such as rare edge cases, unexpected human behaviors, or varying environmental conditions [55, see p. 1].

To mitigate these risks, various approaches have been developed to generate training data and experiences with minimal mismatch between simulated and real-world environments.

One strategy for transferring knowledge from simulation to the real world is through zero-shot or direct transfer, where models trained entirely in simulation are deployed without further retraining. This approach relies either on building highly realistic simulators or generating a large enough variety of simulated experiences to ensure sufficient coverage of real-world scenarios. To further improve transferability, system identification techniques can be employed to construct precise models of real-world

vehicle dynamics, reducing discrepancies between simulated and actual behavior. Another technique, domain randomization, aims to address the simulation-reality gap by extensively randomizing simulation parameters, such as lighting, textures, object placements, and sensor noise, so that the model learns to generalize across a wide range of conditions, increasing the chance that real-world environments fall within the range of variations encountered during training. In contrast, domain adaptation methods use data from a source domain to improve the performance of a model on a different target domain, where data is often less available. Since the source and target domains typically have different feature spaces, these techniques aim to unify the two by minimizing their distributional differences, making it easier to transfer knowledge and improving performance on real-world data [55, see p. 3, 5].

The importance of profound simulation-to-real strategies is underscored by real-world accidents involving autonomous vehicles. In the 2018 Uber crash in Tempe, Arizona, a self-driving Volvo XC90 operating in fully autonomous mode fatally struck a pedestrian at night under low-visibility conditions. Although the vehicle's systems reportedly failed to detect the pedestrian, other systems like Intel's MobileEye were able to recognize the pedestrian shortly before the collision [31]. Similarly, the 2016 Tesla crash in Florida involved a Model S operating under Autopilot mode that failed to distinguish the white side of a tractor-trailer against a bright sky, resulting in a fatal accident when the vehicle passed underneath the trailer [28].

These incidents highlight that simulators should complement, not replace, real-world testing. Simulation must be recognized as one component of a comprehensive validation process that also relies heavily on extensive on-road evaluations to ensure safety and robustness before deployment.

2.3 Regulations

AI systems introduce unique safety challenges compared to traditional software applications, such as biases in datasets, lack of transparency, and limited explainability of model components. To mitigate these risks and promote human-centric trust, regulatory frameworks have been established to guide the development, deployment, and use of AI systems, ensuring they meet standards for robustness and accountability. Developers are required to comply with specific legal and technical requirements to align with these frameworks. In the domain of autonomous driving, such safety considerations are particularly critical, as system failures can have life-threatening consequences. The European Union's Artificial Intelligence Act (AI Act) [45] specifically targets the regulation of AI technologies, establishing legal obligations based on risk levels. In parallel, the General Data Protection Regulation (GDPR) [54] governs the handling of personal data, including that used by AI systems. Together, these frameworks aim to ensure that AI technologies are developed and applied in a responsible, secure, and privacy-preserving manner.

2.3.1 AI Act

The AI Act is a comprehensive regulatory framework proposed in April 2021, and was officially approved in May 2024. It is the first major legal framework in the world that seeks to regulate artificial intelligence based on its potential risk to individuals and society. The act aims to ensure the safety, transparency, and accountability of AI systems while fostering innovation within the European Union (EU).

The AI Act classifies AI systems into four risk categories:

1. **Unacceptable Risk**: AI systems that pose a clear threat to safety, livelihoods, or rights are banned.
2. **High Risk**: These systems are permitted but strictly regulated.
3. **Limited Risk**: Systems that require some transparency, such as informing users they are interacting with AI.
4. **Minimal Risk**: Applications that are allowed with no restrictions (e.g. AI in video games or spam filters).

Autonomous driving systems are categorized as high-risk AI under the AI Act, given their potential impact on safety and fundamental rights. The Act defines several mandatory requirements to ensure these systems are developed and deployed responsibly [45, see p. 55–61]:

- **Risk Management:** A continuous, lifecycle-wide process must identify, evaluate, and mitigate risks to safety and rights. Systems must be tested to ensure effectiveness of the chosen mitigation strategies, particularly for vulnerable groups.
- **Data Governance:** Training, validation, and test datasets must be relevant, representative, complete, and as error-free as possible. Providers must also address bias and may use special categories of personal data under strict conditions to correct it.
- **Technical Documentation:** Comprehensive documentation is required prior to deployment, including system architecture, intended purpose, data usage, risk assessment methods, and performance metrics, to facilitate regulatory oversight.
- **Logging and Traceability:** High-risk AI systems must log operational data throughout their lifecycle. Logs should include usage timestamps, accessed databases, data matches, and verification details to enable traceability and accountability.
- **Transparency and Instructions:** Systems must come with clear, accessible instructions covering the provider's identity, system capabilities and limitations, potential risks, output interpretation, and maintenance requirements.

- **Human Oversight:** Oversight mechanisms must allow human operators to understand, monitor, and override system behavior when necessary.
- **Accuracy, Robustness, and Security:** Systems must perform reliably, with declared accuracy levels, resilience to faults, fallback mechanisms, and protections against adversarial threats. Design must also aim to reduce biased outputs.

To satisfy the mentioned requirements, careful design must be integrated throughout the entire lifecycle of autonomous driving system development. This includes early-stage considerations such as selecting appropriate datasets with minimal bias, evaluating models across diverse simulation platforms, maintaining comprehensive documentation of model behavior and limitations, and implementing detailed logging and visualization tools during training and testing. These measures are also presented in this thesis to support transparency and compliance. Although end-to-end autonomous driving systems, the primary focus of this work, offer improved robustness and performance in complex traffic scenarios, their inherent lack of transparency poses extra challenges for regulatory compliance.

2.3.2 General Data Protection Regulation

GDPR is a comprehensive data protection law that came into effect on May 25, 2018, across the European Union. Its primary goal is to protect the personal data and privacy of individuals while ensuring the free movement of such data within the EU. It requires organizations to process personal data lawfully, transparently, and for clearly defined purposes.

One of the key obligations under GDPR is data minimization, which requires organizations to collect and process only the personal data that is strictly necessary for a particular purpose. This principle helps reduce the risk of misuse, exposure, or unauthorized access to personal data by ensuring that only relevant information is collected and stored.

Another critical requirement is security through pseudonymization or encryption. Pseudonymization involves replacing identifying fields within a data set with artificial identifiers, making it more difficult to directly link the data to specific individuals. On the other hand, encryption encodes the personal data, rendering it unreadable without the correct decryption key.

These methods are designed to protect data from unauthorized access, particularly in the event of a data breach, and ensure that personal information remains secure throughout its lifecycle.

Chapter 3

Objectives

The objective of this thesis is to investigate and benchmark end-to-end autonomous driving agents within a simulation-based evaluation framework, with a particular focus on the role of simulators in ensuring safe and credible model deployment in real-world applications. Recognizing that the choice of simulator significantly impacts the validity of performance assessments, this work aims to emphasize the importance of not only improving driving models but also critically evaluating the simulation environments themselves.

To achieve this, the thesis explores the NAVSIM framework in depth, analyzing its dataset, scoring functions, and evaluation procedures. Baseline agents will be reviewed to develop a deeper understanding of the framework, followed by the design and integration of custom models. These include enhancements to the TransFuser Agent, a state-of-the-art imitation learning model, and an investigation into the novel use of large language models for autonomous driving, evaluated through the implementation of the Language Model Agent.

All models will be systematically benchmarked and compared across standardized metrics and diverse traffic scenarios to highlight architectural strengths and weaknesses. In addition to technical aspects, this thesis also considers human-centered factors to ensure that these technologies are designed and deployed in a manner that is safe, fair, and ethically responsible.

Chapter 4

NAVSIM - The Benchmarking Framework

Benchmarking the performance of autonomous driving systems in a clear and representative manner is essential before deploying them in real-world scenarios. This means that the evaluation must effectively differentiate between well- and poorly performing models, assigning scores that realistically reflect how these models would perform in real-world driving conditions. The results should remain consistent for the same model, even when there are slight variations in the input data, random seed, or less critical hyperparameters. Another notable aspect of the simulator is its processing speed and scalability, particularly in handling larger datasets and more complex models with millions of parameters. In practice, a self-driving agent receives information about its environment through sensors placed on the vehicle. This means that, when training these models, large volumes of sensor data must be managed, which can capture rare and complex traffic conditions. Open-loop and closed-loop simulators each approach these criteria differently, offering distinct advantages and drawbacks. Consequently, no single simulator can fully replicate real-world conditions. Testing models in multiple simulators is recommended to gain greater confidence in their real-world performance prior to deployment. Nevertheless, NAVSIM serves as a robust foundation for benchmarking end-to-end self-driving agents, a topic that is explored in detail in this chapter.

4.1 Simulation Framework

As previously mentioned, in NAVSIM the agent does not interact with its environment, therefore it should be categorized as an open-loop benchmarking framework. However, NAVSIM advances traditional open-loop benchmarking by approximating elements of closed-loop evaluation, aiming to combine the advantages of both methods. Here are some key issues with other popular benchmarking frameworks that NAVSIM seeks to overcome [15, see p. 1–2]:

1. Dataset is not relevant for planning, which lead to blind driving policies[1] can achieve state-of-the-art displacement errors.

2. Existing metrics often misrepresent the relative accuracy of trajectories.

3. Driving agents can not interact with each other.

4. Evaluation metrics are not standardized.

Using datasets that were not primarily created for planning tasks, such as nuScenes [4] (which was designed for perception tasks), can lead to undesirable outcomes. For example, blind driving policies might achieve similar results to sensor-based models, which must be avoidable at any cost (1). NAVSIM addresses this issue by sampling over 100k challenging scenarios from the nuPlan dataset and filtering out the majority of the data, which consists largely of straight driving information that is less representative for trajectory planning. Traditional open-loop metrics, like the average displacement error (ADE) often misrepresent the relative accuracy of trajectories (2), since they assume that only one trajectory is optimal for each scenario. To address this limitation, NAVSIM identifies a set of diverse, efficient, and principled metrics that cover multiple facets of the autonomous driving task. In an open-loop setup, the agent can not interact with the environment by definition, but would be desirable to achieve similar behavior (3). While sensor simulation could potentially address this issue, the domain gap remains too large at present of writing, so it has to be solved differently. NAVSIM makes an assumption, that the driving agent should not influence its environment within a short time horizon. Even having this strong assumption, the results still able to correlate with closed-loop benchmarking metrics, as proved in the original paper [15]. Finally, the framework is capable of assessing model performance in a more standardized way through its scoring method and the OpenScene[2] [13] dataset, effectively addressing a common issue that may be overlooked in other simulators (4). It is interesting to mention that, the performances of the best methods developed in both open-loop and closed-loop settings are similar, despite a vast difference in computational requirements for their training [15, see p. 2].

4.1.1 Dataset

NAVSIM operates with the OpenScene [13] dataset, which is a redistribution of the largest annotated publicly available dataset for autonomous driving, nuPlan. The dataset was sampled with the frequency of 2Hz, reducing the storage requirements by 90% from over 20TB to 2TB. Resulting in 120 hours of driving data, recorded from four major cities: Boston, Pittsburgh, Las Vegas, and Singapore. [13]. The agent input consists of 8 camera images, each with a resolution of 1920×1080 pixels (Figure 4.1), and LiDAR point cloud (Figure 4.2), merged from 5 sensors. The input includes the current time-step and optionally 3 past frames, totaling 1.5s

[1] Blind driving policy refers to a naive policy, like an agent driving at constant speed, completely disregarding the surrounding environment.

[2] OpenScene is a compact redistribution of the large-scale nuPlan dataset.

at 2Hz [15, see p. 5]. Additionally, NAVSIM is compatible with any dataset that includes annotated HD maps, object bounding boxes, and sensor data, it just needs to be converted to the right format before use. Sensor data is stored under the `\sensor_blobs\` directory.

Figure 4.1: The central frame displays the ego-vehicle from a Bird's Eye View (BEV) perspective, while the surrounding eight frames show the views from the cameras mounted on the vehicle. This image was generated using the NAVSIM framework, adding additional visual features to the default recordings, including object bounding boxes and the BEV representation.

In addition to image and LiDAR sensor data, ego-state information is available for each scenario. Ego-state features include numerical values describing the vehicle's current status, such as velocity, acceleration, and driving command (left, forward, right, or unknown). This data is stored in JavaScript Object Notation (JSON) format, identified by the *token_id* property. JSON documents also include properties like scene, frame, and roadblock identifiers, along with traffic light states, annotations, and references to related sensor data. These log files are serialized into pickle (.pkl) format and stored under the `\navsim_logs\` directory. Since logs contain references to sensor data files, their relative paths must adhere to the guidelines in the official NAVSIM GitHub repository documentation [11] to prevent errors.

NAVSIM also supports dataset splits, enabling standardized training and evaluation of autonomous driving agents. These splits further divide the original OpenScene dataset, providing greater control and flexibility for different applications. Instead of listing all available splits, the focus will be on the *navtrain* split, which has been found to be the most useful and is also recommended for most scenarios. The *navtrain* split is a filtered version of the *trainval* split, retaining only the most challenging scenarios. This reduction cuts down sensor blob storage requirements from

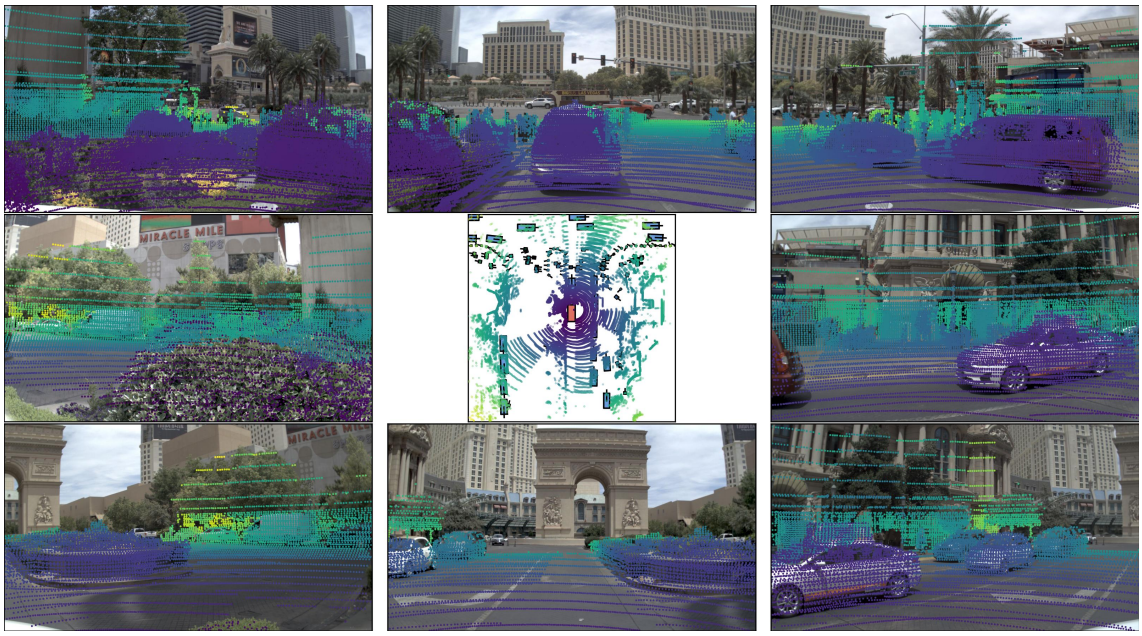


Figure 4.2: LiDAR sensor data is projected onto the surrounding camera views, while the middle frame presents it from the BEV perspective.

2100GB to 445GB, allowing models to train faster, saving computational resources, and enabling quicker iteration across different configurations.

The framework supports data preparation through the *DataLoader* and *SceneFilter* classes, allowing for efficient management and preprocessing of data. The separation of train, validation, and test sets can be easily configured via configuration files, streamlining setup and enabling flexibility in dataset usage. Before the training phase, NAVSIM caches the data being used to enhance performance. By default, this caching process runs automatically, overwriting any previously cached data. While this can be beneficial in certain cases, it can also lead to unnecessary resource usage. To avoid updating the cache each time, an existing cache path can be specified as a parameter, allowing the model to utilize this pre-existing cache for training.

4.1.2 Scoring Function

NAVSIM evaluates results in a non-reactive, open-loop setup, where driving agents are queried only at the initial frame of each scene for scoring purposes. The predicted trajectory remains fixed for the entire duration of the scene, with a default time horizon set to 4 seconds. Within this period, frames are sampled at a frequency of 10Hz, resulting in 40 frames per scene. This lack of environmental feedback increases the difficulty of the agent's task, as it must predict a suitable trajectory without any real-time adjustments. The main advantage of NAVSIM compared to traditional open-loop benchmarking is that it does not limit trajectory comparison to a single human expert path. Instead, it incorporates simulated outcomes to compute more relevant and comprehensive metrics, providing a more robust evaluation of the model's performance. Simulation steps rely on a Linear Quadratic Regulator (LQR)

[33] controller, which calculates optimal steering and acceleration values for each frame.

NAVSIM evaluates driving agents through a two-step scoring process. First, individual subscores are computed, each in the range of [0, 1]. These subscores are then aggregated into the PDM Score (PDMS) $\in [0, 1]$. Named after the Predictive Driver Model (PDM) [14], this scoring approach is a reimplementation of the state-of-the-art rule-based planner that uses the metric to evaluate trajectory proposals during closed-loop simulation in nuPlan. A higher score indicates a better performance. PDMS is adaptable, allowing subscores to be added or removed, aggregation parameters to be modified, or subscores to be made more challenging (e.g., by adjusting internal thresholds). It is calculated per frame and averaged across frames. NAVSIM uses the following aggregation of subscores to calculate PDMS [15, see p. 4]:

$$\text{PDMS} = \underbrace{\left(\prod_{m \in \{\text{NC,DAC}\}} \text{score}_m \right)}_{\text{penalties}} \times \underbrace{\left(\frac{\sum_{w \in \{\text{EP,TTC,C}\}} \text{weight}_w \times \text{score}_w}{\sum_{w \in \{\text{EP,TTC,C}\}} \text{weight}_w} \right)}_{\text{weighted average}} \tag{4.1}$$

As shown in Equation 4.1, the PDMS score is calculated by multiplying two primary components: *penalties* and *weighted average*. These components are determined by the significance of their respective subscores. The penalties component penalizes the agent when more critical events occur, such as collisions, using a factor < 1. The weighted average adds granularity to the PDMS calculation by assigning weights to the subscores, which represent less critical factors, like progress or comfort. These weights can be adjusted for specific use cases, and they are robust to change.

Penalties are applied to punish the unacceptable behavior of the driving-agent, that violates traffic rules and could potentially lead to accidents. Avoiding collisions and staying on the road are crucial for motion planning, as they ensure compliance with the rules and the safety of pedestrians and other road users. Failing to drive with no collisions (NC) with road users (vehicles, pedestrians, or bicycles) or violating drivable area compliance (DAC) rules results in hard penalties, setting $\text{score}_{NC} = 0$ or $\text{score}_{DAC} = 0$, respectively. This leads to a PDMS of 0 for the current scene. Collisions deemed "not at fault" in the non-reactive environment, such as when the ego vehicle is stationary, are excluded. For collisions with static objects, a softer penalty is applied, assigning $\text{score}_{NC} = 0.5$.

Weighted average The weighted average considers ego progress (EP), time-to-collision (TTC), and comfort (C). The ego progress subscore, score_{EP}, represents the agent's progress along the route center as a ratio of an approximated safe upper bound derived from the PDM-Closed planner [14]. PDM-Closed computes a progress value without collisions or off-road driving using a search-based strategy based on trajectory proposals. The final ratio is clipped to [0, 1], discarding low or negative progress scores when the upper bound is below 5 meters. The TTC subscore ensures that driving agents maintain safe margins from other vehicles. Initially set to a default value of 1, this subscore is reduced to 0 if, at any simulation step within a

4-second horizon, the time-to-collision falls below a specified threshold. For TTC calculation, the agent's position is projected forward with constant velocity and heading. Finally, the comfort subscore is calculated by comparing the trajectory's acceleration and jerk against predetermined thresholds.

Based on the PDM-Closed planner and the 2023 nuPlan challenge, the weights are defined with the following values by default: $\text{weight}_{EP} = 5$, $\text{weight}_{TTC} = 5$ and $\text{weight}_{C} = 2$. Modifying these values should not significantly affect the final PDM score, as the top three ranking models in the NAVSIM challenge remained consistent even when equal weights were assigned [15, see p. 4].

***Note**: The PDMS and all subscores are defined within the range of [0, 1]. However, they are multiplied by 100 when displayed on the leaderboard or used to compare agent performance. This convention will also be followed in the subsequent chapters.*

4.1.3 Privacy and Fairness

Bias in datasets is a well-recognized challenge in machine learning and can significantly affect the fairness and reliability of autonomous driving systems. When certain road types, weather conditions, or geographic regions are underrepresented, models trained on such data may generalize poorly and exhibit inconsistent or inequitable performance. For example, a dataset dominated by urban scenes may lead to weaker performance in rural environments. Additionally, the use of human drivers for data collection introduces behavioral biases, shaped by personal habits, cultural norms, or risk tolerances, which can be unintentionally learned and reproduced by the model. While expert human drivers can provide high-quality demonstrations of safe and context-aware behavior, their individual styles may still influence the system in unintended ways. The OpenScene dataset attempts to mitigate such issues by incorporating driving data from multiple geographical regions. This diversity helps to reduce location-specific bias, though some areas are still missing due to practical limitations in data collection. While this may not pose a critical issue if the deployment environment is similar to the collected data, broader generalization likely requires additional data from new drivers and new locations. Scenario diversity is another crucial factor. NAVSIM filters out straight-driving segments, which constitute the majority of the dataset. This improves the benchmarking process by balancing the scenario distribution, enabling a more accurate evaluation of model decision-making capabilities.

It is also important to mention that the OpenScene dataset contains visual data such as pedestrian faces and vehicle license plates, which are considered personal data under the GDPR. Although the data was originally collected in the United States and Asia, where GDPR does not apply directly, the use of such data within the EU still raises privacy concerns. While no explicit personal identifiers such as names or addresses are included, the presence of identifiable visual information may still allow for the indirect identification of individuals. This poses potential GDPR compliance challenges that requires additional safeguards such as anonymization or blurring. These become especially important in commercial applications, where compliance is strictly enforced and noncompliance may result in legal or financial consequences.

Chapter 5

Proposed Work

In this chapter, the experiments conducted using the NAVSIM framework will be discussed. The chapter will begin by introducing the environment in which the experiments were performed, emphasizing the key considerations related to the setup. Subsequently, the steps required to create an agent will be outlined, including the main architecture and essential classes. The process of loading data, training models, running evaluation jobs, and utilizing additional features to enhance productivity will also be explained. After covering these fundamental steps, the chapter will focus on the baseline models. Specifically, three models will be described: the Constant Velocity Agent, the Ego Status MLP[1] Agent, and the TransFuser Agent. These models will be introduced sequentially, as each has a more complex design than the previous. Alongside increasing complexity, these models also demonstrate significantly improved performance, achieving higher PDM scores. Experimenting with these baseline models provides a deeper understanding of the framework and establishes a solid foundation for later sections. Although new models can be created from scratch, building upon an existing baseline model offers a practical starting point. This approach enables the creation of custom solutions more efficiently and helps users become more familiar with the framework. However, the framework is not limited to this approach, any model compatible with NAVSIM's input and output formats can be implemented. A good example of this is the Language Model Agent, which adopts a completely different approach.

5.1 Software and Hardware

Simple models, such as the Constant Velocity Agent, do not involve computationally intensive workloads and can be run without the need for Graphical Processing Units (GPUs), allowing experiments to be completed in a relatively short time. As models become more complex, such as those including Artificial Neural Networks (ANNs), the need for more powerful hardware increases, as seen with the Ego Status MLP Agent. Training ANNs typically requires GPUs, which are optimized for handling heavy mathematical operations, particularly matrix multiplications, and enabling parallel processing. The need for GPUs becomes even more critical for models like

[1] A multilayer perceptron (MLP) is a type of neural network composed of fully connected layers.

the TransFuser agent, which also learns from sensor data, and can contain billions of learnable parameters, depending on the configuration. For larger models, multiple GPUs can be combined to work together (scaling out). A practical solution is to use a cloud provider or a dedicated on-premises GPU server.

The experiments were conducted using the DGX server, a GPU server managed by the Department of Telecommunications and Artificial Intelligence (TMIT) at my university. This server is equipped with four Tesla V100-DGXS-32GB GPUs. However, to minimize resource usage, only one or two GPUs were utilized for the majority of the experiments. Access to a Linux Docker container was provided, and a connection was established via Secure Shell (SSH). Visual Studio Code[2] (VS Code) was used as the editor, although this was a personal preference. The Remote - SSH extension[3] was found to be extremely useful, significantly enhancing productivity. The graphical interface allowed for easy access to the file explorer, the ability to open multiple tabs, visualize evaluation outcomes, and more. Furthermore, Conda[4] was employed for managing Python packages across different virtual environments and projects.

To begin working with NAVSIM, the original GitHub repository was cloned from `https://github.com/autonomousvision/navsim`. The required *maps* and *navtrain* datasets were then downloaded. Although helper scripts were provided for downloading the data, the script for the *navtrain* split was missing. A new script was created to address this, based on those available for other splits. This step was essential, as *navtrain* is the most compact split and the most suitable for the intended use case, given the resource limitations (e.g., disk space). Once the data was downloaded, it was organized into the correct file hierarchy, as described in Section 4.1.1. The installation process beyond these steps was straightforward, following the provided documentation.

It is also worth mentioning that training and evaluation tasks can take several hours or even multiple days to complete. During this time, it is crucial for the session to remain active, otherwise, the task may fail with an error. To ensure this, the Linux `screen` command was used to start a new session, which keeps the session alive even if the SSH connection is closed. Another useful tool is `nvidia-smi`, which allows for monitoring of GPU resources. This command proves especially valuable when combined with the `watch` command, enabling continuous tracking of current GPU usage.

[2]VS Code is a popular code editor, available at `https://code.visualstudio.com/`

[3]Remote - SSH is a VS Code extension to simplify development and troubleshooting on a remote machine.

[4]Conda is an open-source package manager system for Python and R.

5.2 Creating Agents

5.2.1 Main Components

NAVSIM is built on the PyTorch Lightning[5] framework and further abstracts its capabilities. All agents in NAVSIM are required to inherit from the ***AbstractAgent*** abstract class and implement its key methods. This abstraction allows the seamless integration of custom agents into the training and evaluation pipelines, serving as an interface to ensure compatibility. The framework also supports data loading, caching, logging, and custom callbacks, enabling users to focus primarily on the model creation aspect of the whole workflow. The key methods vary depending on whether the agent has learnable parameters and requires training (learning-based agent) or does not require training (non-learning-based agent). Below, the methods are listed, with the main purpose of each being highlighted [12]:

- **___init___()**: Agent constructor.
- **name()**: Name of the agent, that will be used to name generated files.
- **initialize()**: Initializes the agent's state. Each worker will call this method for its instance of the agent before making inferences.
- **get_sensor_config()**: Has to return a SensorConfig object. It defines which sensor modalities should be loaded for the agent in each frame. SensorConfig is a dataclass that stores for each sensor a List of indices of history frames for which the sensor should be loaded. Only the necessary sensors should be loaded, as this has a significant impact on runtime.
- **compute_trajectory()***: Only needs to be implemented for non-learning-based agents. Given the AgentInput, which contains the ego state and sensor modalities, it must compute and return a future trajectory for the agent, represented as an array of BEV poses (with x, y and heading in local coordinates). (The following methods are not relevant for the non-learning-based agent.)
- **get_feature_builders()**: Has to return a List of AbstractFeatureBuilder. Each FeatureBuilder takes the AgentInput object and compute the feature tensors[6] used for agent training and inference. A single FeatureBuilder can compute multiple feature tensors and should return them in a dictionary format of type Dict[str, Tensor].
- **get_target_builders()**: Has to return a List of AbstractTargetBuilder. It is similar to the get_feature_builders() function, but the TargetBuilder has access to the Scene object in addition to the AgentInput. The Scene object contains ground-truth information for training.

[5]PyTorch Lightning is a deep learning framework, that provides a high-level interface for PyTorch. Available at `https://lightning.ai/pytorch-lightning`

[6]A tensor is a generalization of scalars, vectors, and matrices to higher dimensions.

- **forward()**: The forward pass through the model. Features are provided as a dictionary containing all the features generated by the feature builders. All tensors are already batched[7] and located on the same device as the model. The forward pass must output a dictionary, with one entry labeled "trajectory" containing a tensor that represents the future trajectory. This tensor should have the shape $[b, t, 3]$, where b is the batch size, t is the number of future timesteps, and 3 corresponds to x, y, and *heading*.
- **compute_loss()**: Given the features, the targets and the model predictions, the function calculates the loss, and returns it as a single tensor.
- **get_optimizers()**: This function is used to define optimizers for the model. It can return a single Optimizer or even a dictionary, that contains the Optimizer and an LRScheduler (learning-rate scheduler).
- **get_training_callbacks()**: This function is optional, and can be used to return a List of pl.Callback (PyTorch Lightning Callback) to monitor or visualize the training process.

All non-learning-based agents must implement the **compute_trajectory()** function, which contains their core logic. Since these agents lack learnable parameters, the future trajectory output must be directly computed from the AgentInput parameters. They can access all sensor data (if provided by **get_sensor_config()**) and ego status data to predict the output. However, these agents are primarily intended for defining baselines or becoming familiar with the framework and are not designed to compete with learning-based agents. Another advantage is their ability to validate the credibility of NAVSIM. A notable example is the Constant Velocity Agent, which performs poorly compared to more advanced models. As discussed in Section 4.1, this behavior is expected and desirable, as it highlights the distinction between state-of-the-art models and simpler, non-adaptive agents.

Learning-based agents follow the fundamental steps typical of deep-learning frameworks. While the function names and return types may not exactly match those in frameworks like PyTorch or PyTorch Lightning, they are intuitive and descriptive. The **get_feature_builders()** and **get_target_builders()** functions generate features in tensor formats, which are later utilized by data loader classes. These functions also retrieve sensor information to include in the features via the **get_sensor_config()** method, similar to non-learning-based agents. The **forward()** method processes the input data through the model and computes the output tensors during each training step. This method is also used during validation steps without any modifications, and the model weights remain unchanged, as handled by PyTorch Lightning. Then, the **compute_loss()** function is called to calculate the loss between the predictions and the target values. The optimizer, configured in the **get_optimizers()** method, updates the model's weights based on the computed gradients and its own optimization logic. Additionally, the **get_training_callbacks()** method can be used optionally to define callbacks that

[7] A batch is a subset of the training dataset that is processed together in one forward and backward pass during training.

monitor the agent's performance during training. Callbacks can be set for the beginning or end of training, validation, and testing epochs[8]. These callbacks can include functionalities like early stopping, logging information, visualizations, and more complex operations tailored to specific requirements.

Although it is only sufficient to implement a single class to define an agent, for more complex models, it is advisable to split the code into multiple Python modules. For example, while the constant velocity and ego status MLP baselines may be defined in a single file, the TransFuser model is divided into multiple helper modules to maintain a clean and organized design.

5.2.2 Configuring and Running Agents

Once the key functions are implemented, the models can be run. The screen command was always used to open a new session before running models. To initiate the training task, running a single Shell script (e.g., `run_model_training.sh`) is sufficient. Separate scripts are preferred for each agent, as this makes them easier to manage. Inside the script, the path to the actual Python file that contains the training initialization logic is specified, along with other useful parameters. Most of these parameters are optional, but the agent name, experiment name, train-test split, and caching options are typically defined, as they usually remain constant for a single agent. The `run_model_training.sh` script executes the Python file, which also loads additional parameters using Hydra[9]. Specifically, there are default configuration files as well as agent-specific configuration files (which are created for new custom agents), organized in a hierarchical structure. This provides the flexibility to easily change configurations, which is very useful when working with deep-learning models that have many hyperparameters to optimize. Hydra can instantiate classes at runtime, so the names in the configuration files must match the exact class names. Additionally, configuration values can be overridden via the command line, allowing to modify parameters from the `run_model_training.sh` script as well. During training jobs, logs are generated, which can be useful for both troubleshooting and monitoring. These logs are stored under the `\exp\` folder, organized by experiment name and timestamp. The generated logs can be used to visually monitor the training progress on TensorBoard[10]. Logs produced by the (optional) callbacks are also located here and can potentially be viewed on TensorBoard. By default, a checkpoint is saved at the end of each epoch, with each new checkpoint overwriting the previous one. These checkpoints can be used to evaluate the model from a specific point in time by setting the checkpoint path parameter in the evaluation script.

Starting an evaluation task is similar to running training jobs. It involves a Shell script with similar parameters. The primary difference is that the Python path points to the `run_pdm_score.py` file, which is used for evaluating model performance. Additionally, a checkpoint parameter must be provided, indicating the specific agent to be evaluated. For non-learning-based agents, the evaluation process

[8] An epoch is one complete pass through the entire training dataset during the training process.

[9] Hydra is an open-source Python framework that enables dynamic, hierarchical configurations, which can be easily overridden via config files or the command line.

[10] TensorBoard provides visualization and other tools for machine learning experiments.

is the same but does not require a checkpoint path. The results are saved in the \exp\ folder, but without the "training" prefix in the experiment name. The final PDM score is displayed as part of the results, along with a `.csv` file containing a detailed list of subscores for each scene, identified by their *token_id*. Another option allows for creating a submission file that can be uploaded to the leaderboard, which is available on Hugging Face[11]. Although the competition has ended, the test server remained open at the time of writing.

5.3 Baseline Models

Baseline models serve as comparison or starting points for developing new end-to-end driving models. These models range from simple, rule-based approaches to more advanced, learning-based architectures. For example, the Constant Velocity Agent represents a basic, non-learning-based model that assumes constant speed and direction, offering minimal complexity and performance. The Ego Status MLP Agent introduces learning-based capabilities using a simple neural network architecture, enabling it to adapt to different driving scenarios with better accuracy. Finally, the TransFuser model integrates sensor data and employs a sophisticated deep learning approach, significantly improving decision-making and trajectory prediction. Studying these baseline models provides valuable insights into the framework's capabilities and serves as a foundation for developing more advanced, custom solutions.

5.3.1 Constant Velocity Agent

The experiments were started by evaluating the constant velocity agent. This agent maintains a constant speed and heading angle, resulting in a straight-line trajectory. The **compute_trajectory()** function processes the AgentInput, which contains the ego vehicle's status information. Since historical status information is not relevant for this agent, it uses only the most recent data. The latest status includes the ego velocity property, represented as a 2D array, which provides the speed along the x and y coordinates. The function calculates the actual speed as a single scalar value derived from these components. Using this speed, the function iterates through the total interval length of the scene to generate the trajectory path. The interval length and the number of poses are derived from the TrajectorySampling configuration. These values are used to handle interpolation when the output frequency differs from the frequency used during evaluation.

Thanks to the dataset filtering, NAVSIM includes more challenging scenarios and fewer simple, straight-driving cases, which increases the complexity of the tasks. These more difficult situations require agents to handle various types of environments, such as complex turns, dynamic obstacles, and more diverse road conditions. This is a conscious design choice in NAVSIM, aimed at promote the development of more capable and sophisticated autonomous driving models. For example, as shown in Figure 5.1, the constant velocity agent struggles significantly in these complex scenarios. The agent, which follows a simple strategy of maintaining a constant speed

[11] Hugging Face is a platform for sharing, training, and deploying machine learning models.

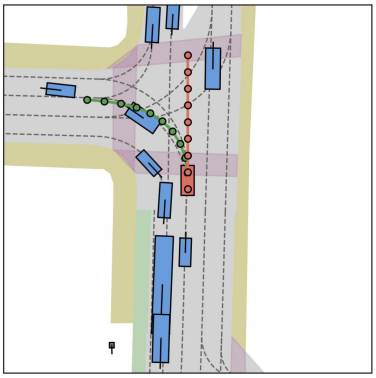


Figure 5.1: The image highlights the limitations of the Constant Velocity Agent. While the ego-vehicle is expected to make a left turn and follow the green trajectory, it continues moving straight along the red path instead.

and heading, fails to navigate turns and continues driving in a straight line, even when the road requires a different course of action. This behavior highlights a critical limitation of such simple models: their inability to handle dynamic environments effectively.

The Constant Velocity Agent's performance aligns with the low PDM score of 20.6517 observed during the evaluation. While this low score might seem disappointing at first, it plays a crucial role in the overall evaluation process. By establishing a baseline for performance, it sets a reference point that allows more sophisticated agents to be measured and compared effectively. Without such baseline models, it would be challenging to track the improvements made by advanced agents, making it harder to assess their true capabilities.

This highlights one of the key strengths of NAVSIM, as it creates a testing environment with enough complexity and variation that simple agents like the constant velocity model are unable to perform well. This, in turn, makes it easier to identify and evaluate more advanced models that can handle such challenging scenarios, thereby emphasizing the strengths of more sophisticated approaches to autonomous driving.

5.3.2 Ego Status MLP Agent

The Ego Status MLP Agent is a learning-based agent, but it is still considered a blind agent since it ignores all sensors that perceive the environment. As the name implies, it is a simple, fully connected neural network that uses only the status information as its input. The **get_sensor_config()** function returns a SensorConfig object, but without any sensor data, as the model does not utilize this information. Including sensor data would create unnecessary overhead and waste resources. The model only uses velocity, acceleration, and driving command status information. The **get_feature_builder()** function returns this information in the form of tensors. To achieve this, an EgoStatusFeatureBuilder helper class (inherited from AbstractFeatureBuilder) was created to compute the features. It extracts the

latest ego status from the AgentInput, which contains the relevant information. The velocity and acceleration values are both defined along the x and y axes, while the driving command can take four possible values: left, forward, right, or unknown. These values are concatenated and converted into a tensor, resulting in a tensor of length eight. The model has an input layer of length 8, matching the size of the input tensor. The network contains three hidden layers, each with a size of 512 by default. Finally, the size of the output layer is determined by $num\_poses \times 3$, corresponding to the representation of the trajectory. The **compute_loss()** function computes the L1 loss between the predicted trajectory obtained from the forward pass and the ground-truth target trajectory. The L1 loss is calculated as the mean absolute error between the two vectors, as defined by Equation 5.1. The ego status MLP agent uses the Adam[12] optimizer for training, with the learning rate set to $lr = 10^{-4}$.

$$\text{L1 Loss} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{5.1}$$

The Ego Status MLP Agent was trained for 50 epochs using the default parameters, and its performance was evaluated afterward. The model achieved a PDM score of 66.8895, representing a significant improvement over the Constant Velocity Agent. The training process was efficient, and the model demonstrated the ability to handle moderately complex scenarios. However, since it does not leverage sensor information, its performance is limited and cannot compete with more advanced models that have richer input data.

After the experiments with these two simple baseline models were finished, the focus was shifted towards the TransFuser Agent, which is by far the most complex and highest-performing model among the baselines.

5.4 TransFuser Agent

The TransFuser Agent is also a baseline model available within the NAVSIM framework. However, it is important to note that it should not be mentioned on the same page as the two previous models. TransFuser is significantly more advanced and achieves state-of-the-art performance in its field. Rather than delving into the complicated implementation details, a high-level overview of its architecture will be provided, highlighting the key ideas behind its design. Additionally, areas of the model that could potentially be improved will be pointed out.

To validate the capabilities of the TransFuser model, it is worth mentioning that team NVIDIA was able to achieve first place in the *Autonomous Grand Challenge* with their Hydra-MDP[36] model. The Hydra-MDP employs a teacher-student knowledge distillation architecture, where the student model learns from a variety of trajectory candidates provided by both human and rule-based teachers. The Hydra-MDP architecture consists of two primary components: the Perception Net-

[12]The Adam optimizer is a widely used adaptive gradient-based optimizer, which is able to adjust learning rates dynamically.

work and the Trajectory Decoder, with the Perception Network being based on the official TransFuser model. This speaks to the robustness and flexibility of TransFuser, which was a key component in the winning solution of the end-to-end driving competition in NAVSIM.

Moreover, the TransFuser model has been heavily relied upon as a foundation for developing custom agents in this work. Its effectiveness made it an ideal starting point for the experiments, which focused on making targeted improvements to this model. Given the significant impact that TransFuser has had both in competitions and in this thesis, more time and effort will be dedicated to explaining this model in the following sections.

5.4.1 Overview

The TransFuser model is a novel approach to sensor fusion for autonomous driving, designed to integrate complementary information from LiDAR and image data. While LiDAR sensors offer precise 3D information, their performance in real-world driving scenarios is often limited by the lack of contextual information, such as traffic lights or HD maps. This information is also typically not available in real time. To overcome these limitations, TransFuser uses the attention mechanism of transformers to fuse features form LiDAR and image data at multiple stages of the model [10, see p. 1–2].

The main idea behind TransFuser is to enhance the representation of the 3D scene by adding global context reasoning into the feature extraction layers. This approach is especially beneficial in complex environments where the ego-vehicle needs to understand interactions between dynamic agents and objects at different distances. For example, traffic lights are located far from the ego-vehicle while it has to navigate through an intersection. Traditional sensor fusion methods often focus on local neighborhood aggregation, which can limit the performance due to the lack of global context [10, see p. 1–2].

The architecture for end-to-end driving consists of two main components: a multimodal fusion transformer designed to integrate information from multiple sensor modalities (images and LiDAR), ensuring comprehensive environment understanding, and an auto-regressive waypoint prediction network, that enables precise trajectory planning. The goal is to complete a given route, while safely reacting to other dynamic agents and following traffic rules [10, see p. 3].

5.4.2 Formulating the Problem

The TransFuser architecture employs the behavior cloning (BC) approach, a supervised learning method within IL, where the agent learns directly from expert demonstrations by minimizing the error between its predictions and the expert's actions [10, see p. 3].

$$\arg\min_{\pi} \mathbb{E}_{(X,W)\sim\mathcal{D}}\Big[L_1(W, \pi(X))\Big] \tag{5.2}$$

Equation 5.2 defines the challenge that the TransFuser architecture addresses through imitation learning. Here, X represents the input features, W corresponds to the expert's human trajectory expressed in BEV space (in ego coordinates), and these together constitute the dataset $D = \{(X^i, W^i)\}_{i=1}^{Z}$ of size Z. In other words, we are looking for a π policy which minimizes the expected value of the L_1 loss function, given the training data X and W. Similar to the Ego Status MLP, TransFuser uses the L1 loss as its primary loss function, as indicated in the formula with the L_1 notation. Additionally, TransFuser applies multiple auxiliary losses to enhance its performance, which will be discussed in greater detail later [10, see p. 3].

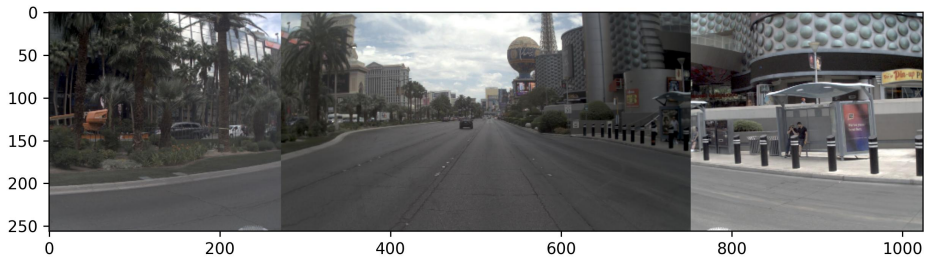


Figure 5.2: The images captured by the three front-facing cameras of the ego-vehicle are stitched together into a single image and resized to a resolution of 1024×256, providing the agent with a wider field of view.

The input feature X consists of three front-facing camera images and a LiDAR point cloud, captured at a single timestamp. As illustrated in Figure 5.2, the camera images are stitched together into a single image, providing an approximately 140° field-of-view (FOV) [15, see p. 8–9]. The LiDAR data is processed to include points within 32m in front of the ego-vehicle and 16m to each side, forming a 32m $\times$ 32m BEV grid. This grid is divided into 0.125m $\times$ 0.125m blocks, resulting in a resolution of 256×256 pixels. The height dimension is discretized into two bins, representing points below and above the ground plane. Additionally, the 2D goal location is transformed into the BEV space and added as a third channel, creating a three-channel pseudo-image of size 256×256 pixels as the LiDAR representation [10, see p. 3].

TransFuser only utilizes data from the latest timestamp, as prior research [48][46][3] has demonstrated that adding historical data does not consistently lead to improved performance.

5.4.3 Multi-Modal Fusion Transformer

The core principle of the transformer's self-attention mechanism lies in its ability to capture the global context between the image and LiDAR modalities. The transformer processes a sequence of features (tokens), represented as vectors, and

enhances them with positional encodings to preserve spatial relationships. Let $\mathbf{F}^{in} \in \mathbb{R}^{N \times D_f}$ represent the sequence of input feature vectors, where N denotes the number of feature vectors and D_f is the feature dimension. The transformer computes queries ($\mathbf{Q}$), keys ($\mathbf{K}$) and values ($\mathbf{V}$) by multiplying $\mathbf{F}^{in}$ with three learnable weight matrices ($\mathbf{M}^q, \mathbf{M}^k, \mathbf{M}^v$). These matrices have the following dimensions: $\mathbf{M}^q \in \mathbb{R}^{D_f \times D_q}, \mathbf{M}^k \in \mathbb{R}^{D_f \times D_k}$ and $\mathbf{M}^v \in \mathbb{R}^{D_f \times D_v}$. Equation 5.3 shows the calculation of the $\mathbf{Q}, \mathbf{K}$ and $\mathbf{V}$ matrices [10, see p. 3–4].

$$\mathbf{Q} = \mathbf{F}^{in}\mathbf{M}^q, \mathbf{K} = \mathbf{F}^{in}\mathbf{M}^k, \mathbf{V} = \mathbf{F}^{in}\mathbf{M}^v \tag{5.3}$$

The attention mechanism computes the attention matrix ($\mathbf{A}$) by first taking the dot product of the query ($\mathbf{Q}$) and key ($\mathbf{K}$) matrices, and dividing the result by the square root of the key dimension D_k. The output is then passed through a softmax function to normalize the values into probabilities. Finally, the result is multiplied by the value ($\mathbf{V}$) matrix to produce the $\mathbf{A}$ matrix. The computation of the attention weights is summarized in Equation 5.4 [10, see p. 4].

$$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{D_k}}\right)\mathbf{V} \tag{5.4}$$

Finally, the transformer applies a non-linear transformation to compute the output features, $\mathbf{F}^{out}$, which maintains the same shape as the input features, $\mathbf{F}^{in}$ [10, see p. 4] (see Equation 5.5).

$$\mathbf{F}^{out} = \text{MLP}(\mathbf{A}) + \mathbf{F}^{in} \tag{5.5}$$

The transformer applies the attention mechanism through multiple layers, each utilizing several attention heads. The convolutional feature extractors for both the image and LiDAR BEV inputs capture different aspects of the scene at various layers. These features are fused at multiple scales within the encoder, where intermediate grid-structured feature maps are represented as 3D tensors for each modality. Positional embeddings are added to these features through element-wise summation, along with velocity embeddings. Each transformer block generates output features of the same size as the inputs [10, see p. 4].

To reduce computational costs, the new features are downsampled using average pooling before being fed back into the transformer. These downsampled features are then upsampled back to the original resolution using bilinear interpolation. This process is repeated across multiple layers with different resolutions, ensuring that important spatial information is preserved throughout [10, see p. 4].

After applying this mechanism, the two feature maps are transformed into feature vectors of length 512 using average pooling followed by a fully connected layer. The resulting 512 dimensional feature vectors from both the image and LiDAR BEV

streams are combined via element-wise summation. This 512-dimensional vector serves as a compact representation of the environment, encoding the global context of the 3D scene. Figure 5.3 illustrates the architectural overview of the TransFuser model, showcasing the fusion transformer component introduced in this section, as well as the waypoint prediction network [10, see p. 4].

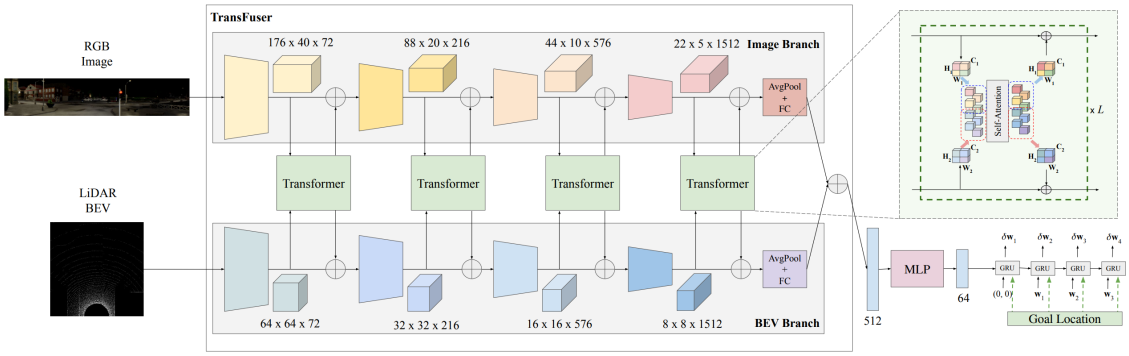


Figure 5.3: Architecture of the TransFuser model [10, see p. 4]. TransFuser takes RGB images and LiDAR BEV features as its inputs, which are then fused together through multiple layers of transformer modules. The combined features are processed with an MLP, followed by an auto-regressive waypoint prediction network.

In the original TransFuser model [10], an auto-regressive waypoint prediction network, based on a Gated Recurrent Unit (GRU), was used to predict the ego-vehicle's future trajectory. The network received the output of the multi-modal fusion transformer, where the 512-dimensional feature vector was reduced to 64 dimensions using an MLP. A single-layer GRU, followed by a linear layer, then predicted the waypoints for the ego-vehicle's future positions.

In NAVSIM, the implementation of the TransFuser baseline has some notable differences. While the fusion transformer backbone operates as previously described to create fused features from image and LiDAR data, the output features are not passed to a GRU model. Instead, a simple MLP, implemented within the TrajectoryHead module, predicts the waypoints. Additionally, auxiliary tasks, including BEV semantic segmentation and Detection Transformers (DETR) style bounding-box detection, are introduced. The trajectory loss is computed using the L1 loss function as previously discussed, whereas the BEV semantic segmentation employs cross-entropy loss, and the bounding-box detection uses Hungarian matching loss. The total loss for the TransFuser agent in NAVSIM is calculated as a weighted summation of these individual losses.

Performing auxiliary tasks has been shown to enhance model interpretability and robustness [10, see p. 5]. In the original implementation, four auxiliary tasks were utilized: depth prediction, semantic segmentation, HD map prediction, and vehicle detection. The results demonstrated that omitting any single auxiliary task did not significantly impact overall scores, however, removing all of them led to a steep performance decline. Interestingly, the most significant performance drop occurred when HD map prediction was excluded, a task that is not implemented for the TransFuser baseline in NAVSIM.

5.4.4 Performance

In the initial experiment with the TransFuser baseline, the model was trained for 100 epochs using the default configuration. The architecture utilizes a ResNet34[23] backbone for both the image and LiDAR branches. Within the backbone, two attention layers are used for feature fusion. The LiDAR sensor data is restricted to the range of $[-32m, 32m]$ along both the x and y axes. For the loss function the following weights were used: $\text{weight}_{trajectory} = 10$, $\text{weight}_{agent\_class} = 10$, $\text{weight}_{agent\_box} = 1$ and $\text{weight}_{bev\_semantic} = 10$. Here, BEV semantic maps were generated for vehicles, pedestrians, roads, walkways, centerlines, and other static objects. The training process was monitored using TensorBoard, where basic metrics such as steps, epochs, training loss, and validation loss were tracked. Additionally, a callback function provided visualizations of the predicted semantic maps and the LiDAR histograms, highlighting the differences from the ground truth values. The visualization of the callback function is displayed in Figure 5.4.

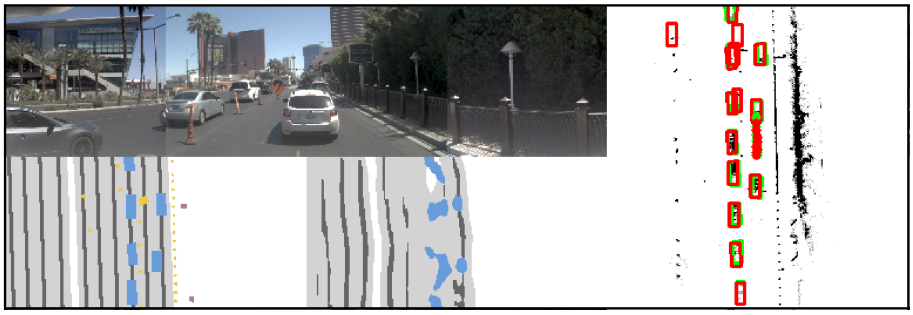


Figure 5.4: Visualization of the TransFuser callback function. It shows the ground-truth and predicted BEV semantic maps in the bottom-left corner. On the right side, green boxes represent the ground-truth, while red boxes indicate the predicted vehicle positions on the LiDAR histogram.

The training process for TransFuser took less than a day using a single GPU, with evaluation completing in under two hours. The final PDM score of 84.8994 highlights the model's effectiveness, demonstrating that TransFuser not only delivers a state-of-the-art solution but also maintains low computational demands compared to other high-performing models. For example, UniAD (Unified Autonomous Driving) [26] is another end-to-end solution, which incorporates a wide range of tasks, such as perception, prediction and planning in semi-sequential architecture. UniAD is used as an example, as ViDAR [51] was previously studied, a visual point cloud forecasting framework, which improves the performance of UniAD in many tasks. However, UniAD requires 80 GPUs and three days of training and still slightly underperforms TransFuser [15, see p. 8].

5.4.5 Key Takeaways

TransFuser is a multi-modal fusion transformer that integrates representations from multiple modalities to capture the global 3D context of a scene more effectively than

previous IL-based models. By leveraging the attention mechanism through multiple transformer layers, it improves at handling complex traffic scenarios. One of its notable advantages is its low computational requirements, enabling fast iterations across different configurations and experiments. Due to these benefits, it was found to be the most valuable baseline for experimentation, particularly in addressing challenges encountered with models like ViDAR. Although TransFuser is already optimized for various configurations, opportunities to enhance its performance still exist. Therefore, key insights about the TransFuser model are highlighted, as they form the foundation for the subsequent ideas.

High Drivable Area Compliance (DAC) often correlates with increased collision rates, as observed when evaluating various models, alongside TransFuser [10, see p. 10]. Failing to drive with no collisions (NC) is heavily penalized in NAVSIM, making it crucial to minimize to achieve a higher PDM score. Collisions primarily occur during unprotected turns and lane changes [10, see p. 10], suggesting that focused analysis of these scenarios offers a promising way for improvement. To maintain high DAC while simultaneously reducing collision rates, strategies targeting safer navigation in these high-risk situations are essential.

It is also important to mention that TransFuser only utilizes the front three cameras for the image input. Experiments have shown that increasing the field of view does not significantly improve the model's performance, while reducing the input to only the middle front-facing camera leads to a decrease in performance. Regarding the LiDAR range, modifying it in either direction from the default settings results in a small performance drop. The optimal number of attention layers lies between 2 and 8, using fewer or more layers leads to lower performance. TransFuser originally used 4 attention layers by default, whereas in NAVSIM, two layers are used. It was emphasized that ensembling plays a crucial role in TransFuser's evaluation, as different initial seeds can lead to significant variations in the scores. In conclusion, due to this behavior, models should be trained and evaluated with multiple seeds to ensure reliable results [10][15].

TransFuser only considers the latest timestamp to generate predictions. However processing temporal inputs is likely necessary to reduce vehicle collisions in intersections by enabling estimation of the acceleration and velocity of other traffic participants [10, see p. 14]. Although previous research [48][46][3] has shown that the use of historical information does not necessarily improve performance, this remains a topic for further discussion. In ViDAR, for example, one of the main ideas was to utilize both spatial and temporal information to predict future states. This pretraining method improved the performance of UniAD in various areas by a range of 4-10%. However, simply adding historical frames may not lead to significant improvements, so more sophisticated preprocessing techniques would likely be needed.

According to the original paper of TransFuser [10], the most impactful architecture choice was the backbone architecture for both the image and LiDAR branches. This means that the optimization of other hyperparameters will probably has less impact on the final score. However experimenting with different backbones looks promising to get better results. In Hydra-MDP [36], team NVIDIA also experimented with multiple backbones, and found that ViT-L [18][50] and V2-99 [32] in combina-

tion with their teacher-student distillation solution leads to significant performance improvements.

Recommended by the TransFuser creators, the model could likely benefit from integrating advanced training techniques to enhance its robustness and adaptability [10, see p. 10–11]. Active Learning [22] could be leveraged to prioritize the most informative data samples, optimizing the learning process by focusing on edge cases and reducing labeling effort. Incorporating DAgger [41] would address the compounding error problem in sequential decision-making by iteratively refining the model through corrective feedback from an expert policy. Adversarial Simulation [47][21] could expose the model to challenging scenarios and unexpected edge cases, thereby improving its resilience to real-world variability. Additionally, Reinforcement Learning (RL) [40] based fine-tuning would allow the model to learn optimal policies through reward-driven exploration, enhancing its decision-making capabilities in complex environments. Finally, teacher-student distillation [36][6][7][8] could be used to transfer knowledge from a larger, more powerful model or ensemble of models to a more compact version, improving efficiency while retaining high performance.

5.5 Custom Models

5.5.1 Multi-Camera TransFuser Agent

The initial approach of improving the TransFuser model focused on expanding the available input sensor data by incorporating rear camera images from the ego-vehicle. This enhancement is particularly important because unprotected turns and lane changes significantly impact collision rates, making these challenging maneuvers a critical starting point for improvement. Currently, TransFuser relies only on front-facing cameras as image inputs, which likely limits its robustness. In contrast, human drivers consistently check side and rear mirrors when changing lanes to avoid collisions with other vehicles. Relying only on forward-facing data while ignoring a part of the surrounding environment introduces significant risk. Therefore, integrating rear-facing images would enable the model to predict future trajectories more accurately and safely. Furthermore, the TransFuser paper itself emphasizes that the existing sensor setup does not capture data from the rear, a crucial limitation in lane change scenarios [10, see p. 14].

In the **get_target_builders()** method, information from the rear camera images was incorporated. The rear images (L2, B0, and R2 cameras) were first cropped to match the resolution of the front-facing camera images (L0, F0, and R0). They were then rearranged in reverse order to align with their visual context and stitched together into a single image. Afterward, the image was horizontally flipped and resized to 1024×256 pixels to match the resolution of the processed front-facing images. Figure 5.5 displays the rear image representation of the scene, similarly to the front view, that was shown in Figure 5.2. As a result, the image feature extraction process produced two feature tensors of identical dimensions, representing the forward and rear visuals. The rear camera tensor representation was added as

a new key-value pair in the result dictionary, while the front camera, LiDAR, and ego-status features remained unchanged.

Figure 5.5: The three rear camera images were combined into a single back-view representation, simulating the perspective a human driver would see in the rear mirror.

The second modification involved updating the TransFuser backbone module to incorporate rear image features, enabling the model to learn a more precise representation of its surroundings. As described in Section 5.4.3, the original design fuses front-facing camera images and LiDAR point clouds in multiple phases to capture the global context of the scene. To enhance this, a new branch was introduced specifically for the rear image features. At each layer, after the fusion of the front image and LiDAR features, the rear image features were also fused with the LiDAR features. Finally, all three features – the front image, rear image, and LiDAR – were combined into a single feature vector, which was returned at the end of the backbone's forward pass.

The BEV feature representation was calculated in a manner consistent with the original TransFuser, but using the modified features from the updated backbone. The rest of the implementation remained unchanged, meaning that the additional rear image features were only utilized in the calculation of the BEV features. The model's forward pass returned the BEV semantic map, along with the agent and trajectory heads, now enhanced with extra visual information from the rear of the model's environment.

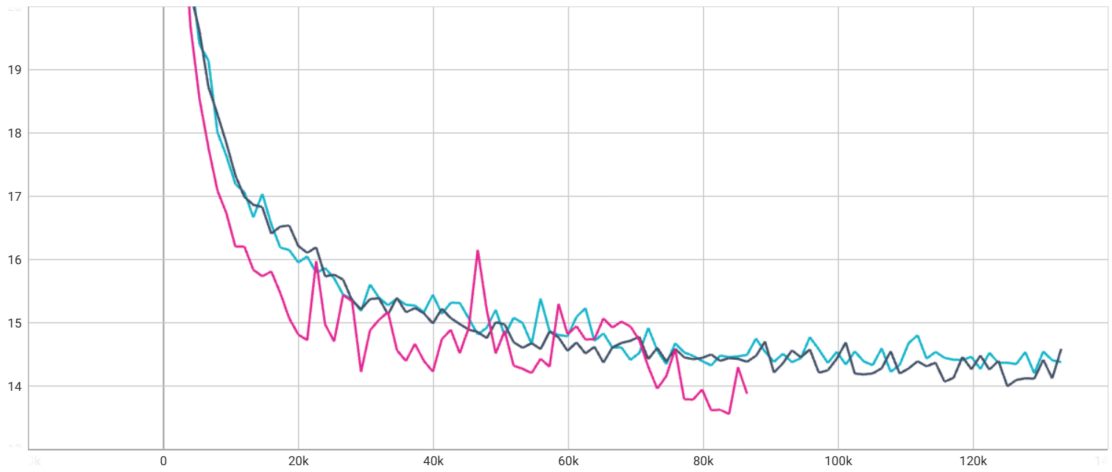
Since the architecture of the Multi-Camera TransFuser Agent closely mirrors the TransFuser baseline, its performance was evaluated under the same configuration. The model was trained for 100 epochs using a single GPU. The extra image processing and feature extraction steps did not significantly affect training time, which was also completed in under a day. After evaluation, the model achieved a PDM score of 84.8313, which is very similar to the baseline score, but unfortunately did not demonstrate an improvement in overall performance. The primary goal of incorporating rear camera features was to reduce collision rates and validate the significance of this added image feature. Although this model did not improve the overall performance, the positive changes in subscores indicate potential for integrating rear camera images effectively. To further validate these findings, experiments will be repeated with different random seeds to confirm the consistency of the results. Additionally, the PDM score is a weighted sum of various subscores, and the results may vary depending on the selected weights. Notably, the NC subscore significantly impacts the PDM score, as it drops to zero if a collision occurs with a tracked vehicle.

Collisions with untracked objects, however, only halve the PDM score. Modifying this value could also change the evaluation result, and probably would be even more realistic for real-world scenarios.

5.5.2 ResNet50 TransFuser Agent

Given that the choice of backbone architecture has a significant impact on model performance, ResNet50 was selected as a larger alternative to the default ResNet34. ResNet50 contains approximately 25.6 million parameters, compared to the 21.8 million in ResNet34, offering improved representational capacity and accuracy. However, this increase in performance comes with higher computational and memory demands, which required additional adjustments to the training setup.

ResNet50 produces deeper feature maps, which required architectural modifications to ensure compatibility with the downstream layers. To manage GPU memory constraints, the batch size was reduced to 16, and gradient accumulation was introduced every 4 batches, effectively mimicking a batch size of 64 while staying within hardware limits. Additionally, training with 32-bit floating point precision proved to be more stable for the deeper network. These adjustments led to significantly longer training times. As the first experiment, training was limited to 65 epochs, taking approximately eight days in total.



Model	Validation Loss	Step	Relative Time
TransFuser	14.58	133,00	21.66 hr
Multi-Camera TransFuser	14.37	133,00	21.36 hr
ResNet50 TransFuser	13.87	86,45	7.98 day

Figure 5.6: The purple curve represents the validation loss of the ResNet50 TransFuser Agent after 65 training epochs, compared to the TransFuser (gray) and Multi-Camera TransFuser (blue) agents.

Figure 5.6 displays the validation loss curve during training of the ResNet50 TransFuser Agent. While the curve indicates lower loss values compared to the previous two models, these improvements were not consistently reflected in downstream evaluation metrics. The model was evaluated on the test set, which is significantly smaller than the training set, allowing evaluation to be completed within a few hours. It achieved a PDM score of 83.0991, representing a slight performance decline relative to earlier models.

To achieve a more fair comparison, training was extended to 100 epochs to validate whether the initial performance drawback was caused by the limited training duration. After 100 epochs, the model achieved a PDM score of 81.6820, indicating an even further decline. These results suggest that despite the deeper architecture's capacity, the model did not generalize better in this setting and may have suffered from overfitting. Adjustments to other hyperparameters could potentially mitigate these issues, but in its current state, the findings suggest that larger models do not necessarily yield better performance. Nevertheless, exploring more capable backbone architectures remains a reasonable direction for achieving further improvements.

5.5.3 Language Model Agent

The power of Large Language Models (LLMs) and Vision-Language Models (VLMs) introduces alternative approaches to traditional autonomous driving agents. These models have been pre-trained on vast amounts of data spanning broad domains, including traffic rules, driving behaviors, and general knowledge about vehicle operation. Leveraging the prior knowledge encoded in such models presents an exciting opportunity for enhancing autonomous driving systems.

A common strategy is to fine-tune these models for specific tasks or stages of the autonomous driving pipeline. VLMs, in particular, demonstrate strong capabilities in understanding visual inputs, allowing them to process camera images and generate natural language descriptions of traffic scenes. These semantic representations can then be used for downstream tasks such as trajectory prediction.

5.5.3.1 Architecture and Implementation Details

Building on this foundation, a Language Model Agent was implemented to explore the applicability of LLMs in autonomous driving. While these models offer impressive capabilities, they also come with significant computational demands (especially for training and fine-tuning) making it challenging to outperform TransFuser-based models given limited resources. Several optimization strategies were required to complete training in a reasonable timeframe, which inevitably introduced trade-offs in performance.

Figure 5.7 illustrates the high-level architecture of the Language Model Agent using a concrete example observed during training. As shown, the model takes two types of inputs: front camera images and ego status data. Unlike previous models, LiDAR data was not included. One of the key design decisions involved selecting the input image resolution, as this significantly impacts computational cost. While

the original resolution (1024×256) provided more detail, it resulted in prohibitively long training times. The resolution was first reduced to 512×128, and eventually further downscaled to 256×64, which proved necessary for efficient training. An example of the downscaled input is shown in the architecture diagram.

The vision-language model processes the input images along with a user prompt to generate natural language descriptions of the traffic scenes. These descriptions are then passed as part of the prompt to the large language model (LLM), along with ego status data such as velocity, acceleration, and the driving command. A mapping function was implemented to convert these numeric values into human-readable language to ensure compatibility with the LLM input. Since the vectors were already centered in the ego vehicle's coordinate system, no additional transformation was required. The driving command was translated into one of four categories: *forward*, *left*, *right*, or *unknown*.

The LLM was then tasked with predicting the trajectory the ego vehicle should follow over the next 4 seconds. Instead of using the LLM's token outputs directly, the final hidden state of the model was fed into a series of multilayer perceptrons (MLPs) to match the dimensional requirements of the predicted trajectory tensor. As with the other models in the thesis, the L1 loss function was used to evaluate performance during training.

After multiple experiments with various vision-language models, including Qwen2.5-VL-3B-Instruct [43] and SmolVLM [39], and several large language models (LLMs), Ovis2-1B [37] was selected as the VLM. It combines the AIMv2 [19] large model for vision and Qwen2.5-0.5B [42] for language. Although Ovis2-1B is a relatively small VLM with moderate performance, it proved to be a good fit under computational constraints. It also offers key advantages such as flexible input resolution and support for batch inference, which was not supported by all tested models.

While Ovis2-1B includes an internal language model, it was found more effective to decouple the language component and use a separate LLM, as this setup provided greater control. The VLM was used solely for inference to generate data for the LLM and was not fine-tuned. This decision saved significant computational resources, as fine-tuning VLMs is especially demanding due to the extra layers required for processing multimodal inputs. However, as a tradeoff, the VLM produced overly general image descriptions, which significantly limited its usefulness for making precise driving decisions in downstream modules.

For the LLM, Qwen2.5-1.5B [42] was selected and fine-tuned using Low-Rank Adaptation (LoRA) [25]. Instead of updating the full weight matrices of the pre-trained model, LoRA introduces smaller trainable matrices that approximate these weights, making the fine-tuning process more efficient. Furthermore, MLP layers were trained alongside the LoRA modules to directly predict trajectory points. Although QLoRA, a quantized variant of LoRA, was also evaluated to further reduce memory usage, it was ultimately excluded due to issues with low-precision floating-point operations when used with the Qwen models.

For both the VLM and LLM components, the maximum number of output tokens was capped at 256 to remain within GPU memory constraints.

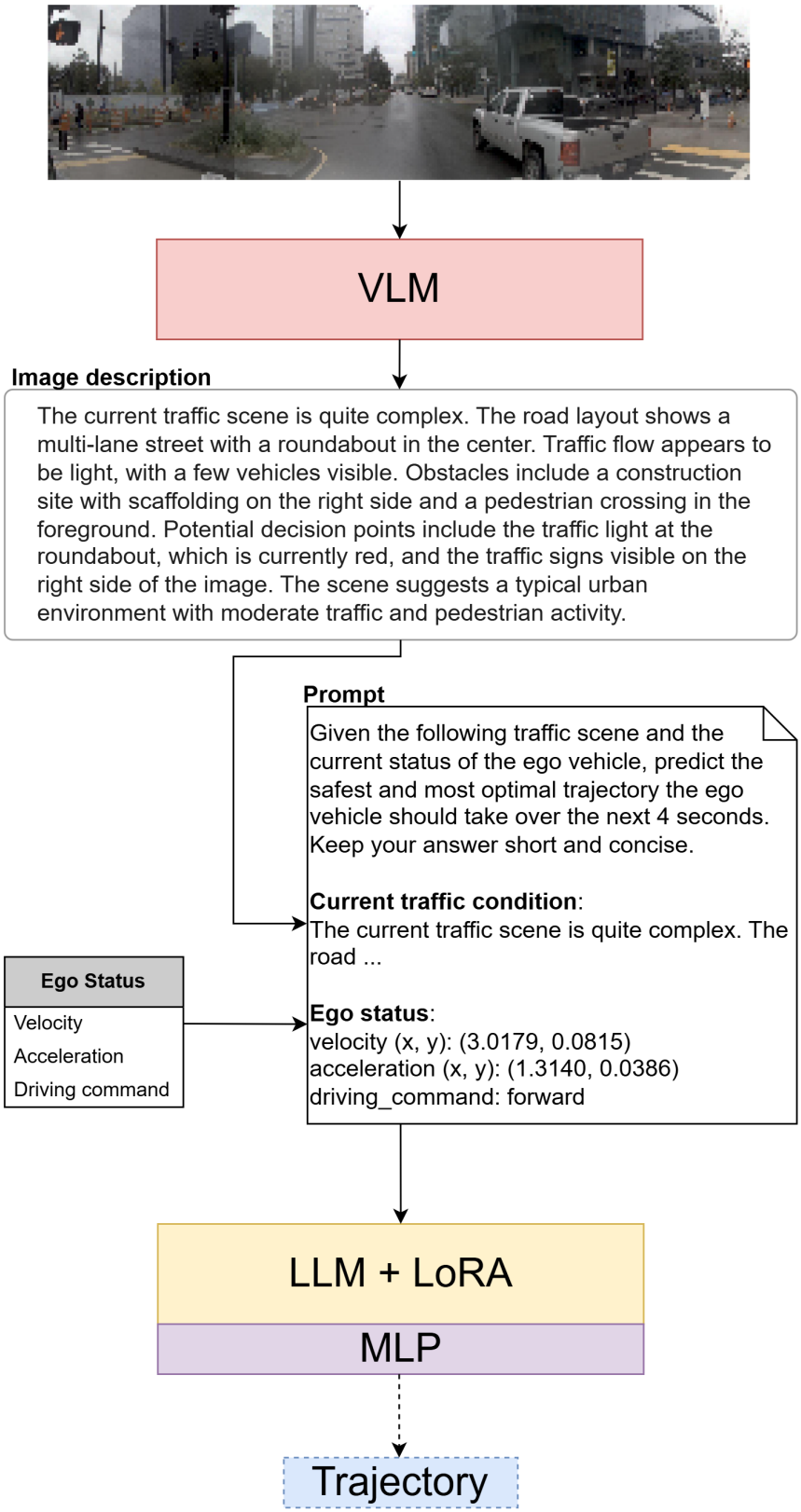


Figure 5.7: Architecture of the Language Model Agent. The values shown were obtained during model training and precisely match the selected example.

The Language Model Agent was trained for 3 epochs with a batch size of 8 and gradient accumulation over 8 steps. Full 32-bit floating point precision was used during training. For the LoRA configuration, a rank of 8 and a LoRA alpha of 16 were selected, with no additional dropout applied. Training was conducted over the course of approximately 3 days using two GPUs. During evaluation, further adjustments were necessary to meet inference latency and memory limits. Ultimately, the model achieved a final PDM score of 53.4692, a notable performance gap compared to TransFuser-based models. However, this result still demonstrates potential, particularly considering the highly constrained and minimalistic training setup.

5.5.3.2 Known Limitations

Computation constraints were considered from the early design phase. This led to the use of low-resolution input images, smaller model architectures, limited output token counts, and shorter prompts. Despite these adaptations, the most significant limitation emerged from the VLM's overly generalized outputs, which may have introduced noise rather than providing helpful guidance to the LLM. This issue could explain why the model underperformed even compared to the Ego Status Agent, which does not utilize any visual input. A viable improvement would be to replace the VLM with one more specialized for the task or to fine-tune it using task-specific supervision, ideally with annotated bounding boxes of relevant traffic participants. However, such an approach would require substantial modifications to the prompt structure, additional preprocessing steps, and increased computational overhead.

Additionally, heavy computational requirements persist even during inference, introducing extra latency. This poses further challenges for deploying such models in real-time, live environments.

Chapter 6

Evaluation Methods and Results

In the **Proposed Work** chapter, evaluation results were primarily presented with a central focus on the PDM score, which was used as the key metric for assessing model performance. While this provided a high-level understanding of the models capabilities, it did not fully explore the nuances of their behavior. This chapter aims to provide a deeper comparative analysis of model performance, with a detailed examination of the subscores that contribute to the overall PDM metric. This granular investigation is crucial for uncovering areas for improvement across the evaluated models. The detailed breakdown allows to pinpoint which components of the models excel in handling NAVSIM's unique challenges. It will also show where the models fall short, providing an opportunity to identify potential areas for enhancement.

Beyond comparative analysis, this chapter will highlight the broader implications of these findings. By analyzing how these models perform under NAVSIM's constraints, their contributions to achieving higher PDM scores can be more clearly understood. Furthermore, this exploration will serve as a foundation for guiding the development of future models, helping to design architectures and strategies that address the identified weaknesses.

Table 6.1 presents the performance of the models discussed in the previous chapter. The first column lists the model names in the same order they were introduced earlier. This is followed by the PDM score, the primary evaluation metric, and then its individual subscore components.

The Constant Velocity Agent demonstrates the poorest performance across all categories, except for the comfort score (C), which remains identical for all models. As expected for an agent based on such a simplistic logic, there is a significant gap in performance compared to the other models. Notably, the ego progress (EP) subscore is exceptionally low, as the agent struggles to follow the designated route. The Constant Velocity Agent successfully avoids collisions in 68% of the tested scenarios and remains within the drivable area (DAC) 57.8% of the time. These metrics offer valuable insights into the dataset's distribution. If these baseline scores were too high, it would suggest that the dataset lacks complexity and does not properly represent challenging traffic scenarios. This, in turn, would call into question the validity of benchmarking more advanced models against such a dataset.

Model	PDMS ↑	NC ↑	DAC ↑	EP ↑	TTC ↑	C ↑
Constant Velocity	20.7	68.0	57.8	19.4	50.0	100
Ego Status MLP	66.9	93.5	78.4	63.1	85.0	100
TransFuser S1	84.9	98.0	93.4	79.6	93.5	100
TransFuser S2	85.0	97.9	93.6	79.6	93.6	100
TransFuser S3	85.0	97.9	93.6	79.5	93.6	100
Multi-camera TransFuser S1	84.8	**98.1**	93.3	79.3	**93.8**	100
Multi-camera TransFuser S2	**85.1**	97.7	93.7	**80.1**	93.2	100
Multi-camera TransFuser S3	85.0	97.8	**93.8**	79.9	93.2	100
ResNet50 TransFuser 65 ep	83.1	97.3	92.2	78.6	91.9	100
ResNet50 TransFuser 100 ep	81.7	97.1	91.4	76.9	91.4	100
Language Model Agent	53.5	86.6	70.4	50.0	75.9	99.2

Table 6.1: Comparison of Model Performance in NAVSIM.

It is also important to note that the comfort score (C) reaches almost its maximum value for all tested models, including the blind baselines. This raises questions about the relevance of this metric in its current form. The method used to calculate the comfort score should be revised to provide more detailed information about the driving experience, or it may even be omitted entirely. Additionally, the time-to-collision (TTS) subscore is always less than or equal to the no collision (NC) value by definition, as collisions with another objects includes a violation of the safety margins to other vehicles. While this is not necessarily problematic, the high correlation between these two metrics suggests that combining them into a single score could be a considerable option.

The Ego Status MLP demonstrates significant improvement across all subscores compared to the constant velocity agent. The most notable improvement is in the ego progress (EP) category, as the model leverages driving commands and acceleration values, enabling it to better navigate in the designated direction. However, this model lacks information about its surrounding environment, and the driving command itself is overly simplistic, represented by a single value (left, forward, right, or unknown). This limited representation fails to differentiate between a sharp turn and a gentle lane change. Including more detailed information, such as the precise angle of a turn, could potentially enhance the performance of this baseline. The limitations significantly impact the DAC score, highlighting the gap between this model and those capable of learning from sensor data. This highlights the importance of incorporating environmental awareness and more descriptive driving commands to achieve further improvements.

Both TransFuser and Multi-Camera TransFuser leverage sensor fusion and outperform the two baseline agents across all evaluation metrics. To ensure the robustness of their performance, both models were additionally trained with different random seeds (denoted by *S1*, *S2* and *S3* on the table). Although their architectures are nearly identical, the primary distinction lies in the use of rear camera images in

Multi-Camera TransFuser, which are incorporated alongside front-facing camera images and LiDAR data.

Overall, their PDM scores are closely matched, making it difficult to declare a clear winner in terms of general performance. When analyzing the subscores, Multi-Camera TransFuser shows a slight advantage in DAC and EP, while TransFuser performs better in NC and TTC categories. Interestingly, DAC and EP are expected to rely more on front-view visual input, raising questions about why the inclusion of rear camera data improves performance in these areas. This suggests that additional visual context may not drastically influence trajectory predictions.

Conversely, the expectation that rear-view awareness would enhance safety-related metrics such as NC and TTC was not supported by the results. TransFuser performed marginally better in these areas, despite lacking rear camera input. Nevertheless, these differences are relatively small, as both models demonstrate strong performance overall, with collisions occurring in fewer than 2% of evaluated scenarios.

Although Multi-Camera TransFuser achieved the highest PDM score in some training runs, it also exhibited greater variability across different seeds. In contrast, the original TransFuser model demonstrated more consistent results with lower performance deviation, indicating greater reliability across training instances.

The ResNet50 TransFuser ranks behind the other two TransFuser-based agents, indicating that a larger model backbone does not necessarily lead to improved performance and may even introduce overfitting. This outcome suggests that simply scaling up the model size is insufficient and other hyperparameters must be carefully adjusted to align with the increased complexity. Rather than merely increasing backbone size, exploring alternative backbone architectures may offer a more promising path for performance gains.

The Language Model Agent achieves moderate results across all subscore categories. Its underperformance compared to the Ego Status Agent appears to stem from the overgeneralized vision inputs and the limited number of training epochs, which introduced additional noise into the model. However, it demonstrates a fundamentally different approach with significant potential for future improvement. Moreover, its use of human-readable representations adds an extra layer of explainability, resulting in a more transparent agent, which is an clear advantage for real-world applications.

Figure 6.1 shows the distribution of the overall PDM scores. For both the TransFuser and Multi-Camera TransFuser agents, the scores were averaged over three runs with the different initial seeds, while for the ResNet50 TransFuser, the version trained for 65 epochs was used, as it achieved better accuracy. The distributions align well with the results presented in the table above.

The most notable observation is the distinct distribution of the ResNet50 TransFuser compared to the other two TransFuser-based agents. It exhibits a higher number of scenarios with scores close to 1, but also more with scores near 0, and fewer in between. This indicates that while the ResNet50 TransFuser can be more accurate in certain cases, it also fails more dramatically in others. This behavior suggests that the model may be overfitting, as performing well on familiar patterns but struggling to generalize to less frequent or more complex scenarios. It highlights

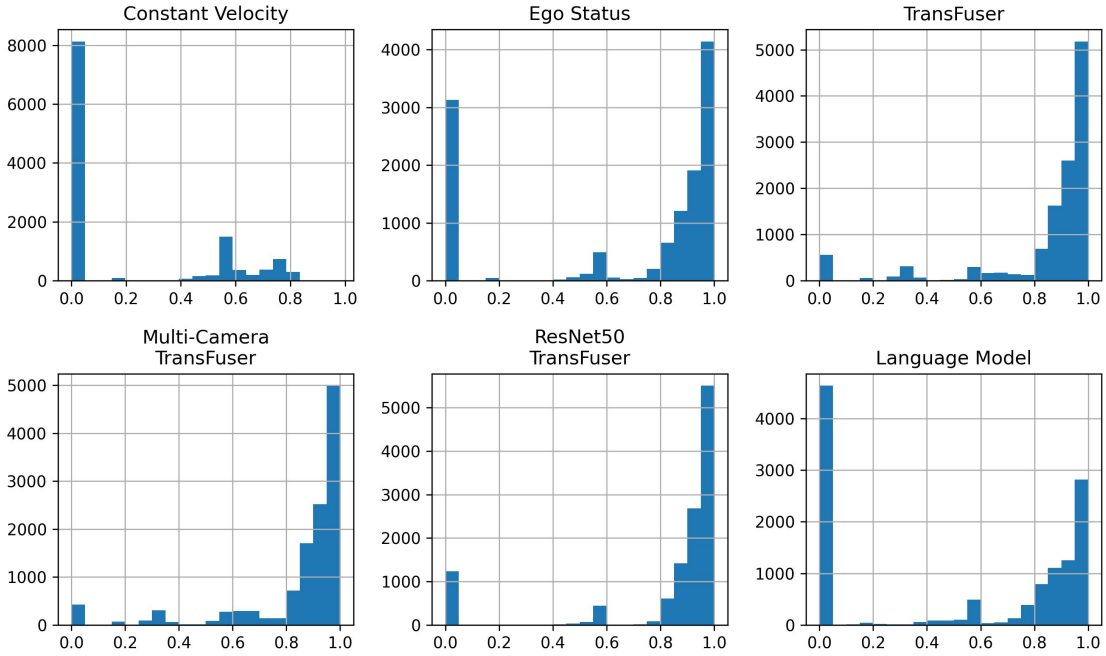


Figure 6.1: Distribution of PDM scores for all tested models.

the importance of balancing model capacity with regularization and appropriate hyperparameter tuning to avoid overfitting.

Finally, Figure 6.2 illustrates the behavior of each agent in randomly selected traffic scenarios across four key situations: taking a turn, performing a lane change, stopping at an intersection, and driving in dense traffic. Overall, agents tend to make less progress compared to the human driver, particularly in more complex scenarios like turning and navigating dense traffic. This behavior could be attributed to the training objective, where collisions are penalized more heavily than reduced ego progress, encouraging models to act conservatively to avoid risk.

The TransFuser-based models show highly consistent behavior across all conditions, reflecting their robustness and ability to generalize. The ResNet50 TransFuser, however, demonstrates greater progress in scenarios involving more straight driving, such as lane changes. This could be because such cases are more frequent in the training dataset.

In turn scenarios, the Ego Progress Agent often deviates between the optimal trajectory and a straight path. This may be due to the limited descriptiveness of the driving command input, which lacks precise angle information, and the absence of any supporting sensor input. The Language Model Agent, on the other hand, typically continues along a straight trajectory with only slight deviations. This behavior is likely caused by the combination of overgeneralized visual descriptions and insufficient training. Unlike the Ego Status Agent, the LLM-based model seems unable to fully utilize the driving command, further suggesting that additional training epochs and refined prompt engineering may be necessary for better performance.

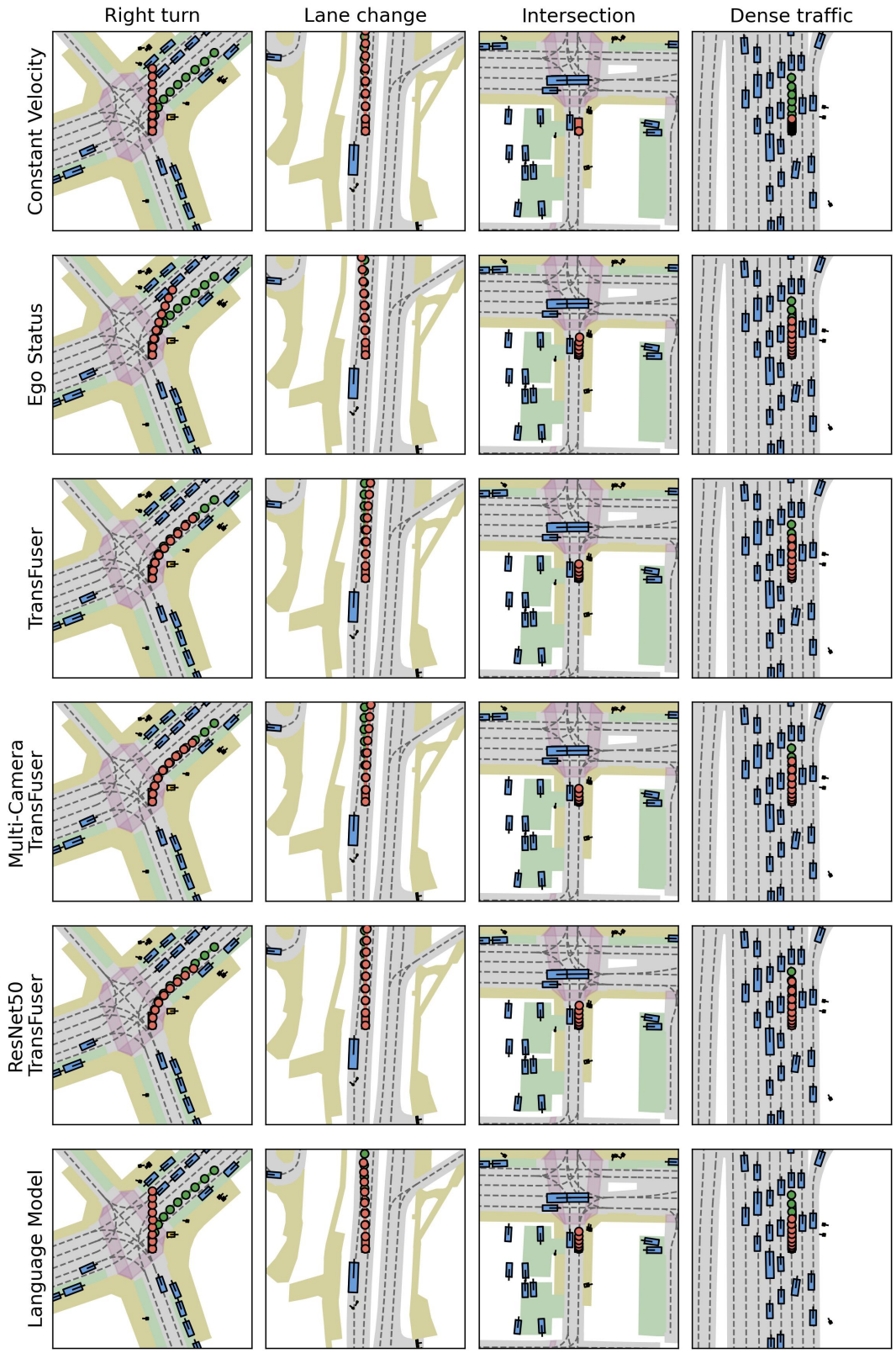


Figure 6.2: Demonstration of each agent's behavior under specific traffic conditions. Red trajectories indicate the paths taken by the models, while green trajectories represent the paths of the human driver.

Chapter 7

Summary

This thesis explored various approaches to autonomous driving agents in simulated environments, with a particular focus on end-to-end solutions, which tend to outperform traditional modular pipelines. A comprehensive comparison was conducted on popular, actively maintained simulation frameworks that are publicly available for research. Among them, NAVSIM, an open-loop simulator, was selected due to its improvements over previous simulators, including more representative datasets and standardized metrics that better reflect model performance.

Different driving agents were benchmarked in NAVSIM, ranging from naive baselines to more advanced models. The architecture of the TransFuser model, a state-of-the-art imitation learning model that leverages transformer networks, was examined in more detail. It achieved the best results in NAVSIM among the provided baselines and served as the foundation for custom models.

One such extension, the Multi-Camera TransFuser was based on the idea that rear camera images could enhance performance, mirroring human reliance on rear-view mirrors during turns and lane changes, which were identified as critical error point and also an important part of the PDM score. This modification led to modest improvements in certain subscores. When evaluated across multiple initial seeds, the Multi-Camera TransFuser maintained a comparable overall PDM score, though with greater variability between runs.

Another variant, the ResNet50 TransFuser, replaced the model backbone with a deeper network. Although the backbone is a critical hyperparameter, the larger model did not lead to improved results, likely due to overfitting. This suggests that scaling up the architecture without adjusting other hyperparameters may not yield better generalization, and that exploring alternative backbone types might be a more promising direction.

A third experimental approach introduced the Language Model Agent, leveraging the capabilities of large language models (LLMs) to represent and interpret driving scenes. This model demonstrated a fundamentally different strategy for autonomous driving. While the implementation faced significant limitations, such as high computational demands and generalization challenges, it revealed strong potential for future research. Notably, its ability to produce human-readable reasoning adds a

valuable layer of transparency, which could be beneficial in driver-assistance applications.

Experiments with custom models highlighted the advantages of NAVSIM, as performance differences were significant compared to simpler baselines. However, for some metrics, performance improvements are difficult to assess due to only minor differences, such as with the comfort (C) metric, which remained consistent across all tested models. Improvements on these metrics could be explored further in future research.

Finally, the thesis examined human-centric aspects of autonomous driving, considering both simulator capabilities and their real-world applicability. One of the major safety concerns remains the simulation-to-reality gap, which continues to be a central focus of research. Broader challenges, including privacy, fairness, and ethical considerations, also need to be addressed through regulation. The AI Act and GDPR were discussed as potential frameworks for managing the risks associated with high-stakes applications like autonomous driving.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Dr. Bálint Gyires-Tóth and András Béres, for their valuable guidance and continuous support throughout the writing of this thesis. I am also thankful to the Department of Telecommunications and Artificial Intelligence at the university for providing the necessary working environment and computational resources. Finally, I would like to thank Continental AG for the opportunity to participate in the Professional Intelligence for Automotive (PIA) program and for the scholarship that supported my work.

Bibliography

[1] Saleh Albeaik, Alexandre Bayen, Maria Teresa Chiri, Xiaoqian Gong, Amaury Hayat Nicolas Kardousk, Alexander Keimer, Sean T. McQuade, Benedetto Piccoli, and Yiling You. Limitations and Improvements of the Intelligent Driver Model (IDM). In *SIAM Journal on Applied Dynamical Systems*, 2022.

[2] Michael Bain and Claude Sammut. A Framework for Behavioural Cloning. In *Machine Intelligence*, 1995.

[3] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst. In *Proc. Robotics: Science and Systems (RSS)*, 2018.

[4] 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 *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[5] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays. Argoverse: 3D Tracking and Forecasting with Rich Maps. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[6] Dian Chen and Philipp Krähenbühl. Learning from All Vehicles. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[7] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Proc. Conf. on Robot Learning (CoRL)*, 2019.

[8] Dian Chen, Vladlen Koltun, and Philipp Krähenbühl. Learning to drive from a world on rails. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2021.

[9] Li Chen, Penghao Wu, Kashyap Chitta, Bernhard Jaeger, Andreas Geiger, and Hongyang Li. End-to-end Autonomous Driving: Challenges and Frontiers. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[10] Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. TransFuser: Imitation with Transformer-Based Sensor Fusion for Autonomous Driving. In *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2022.

[11] NAVSIM Contributors. NAVSIM – Download and installation. URL https://github.com/autonomousvision/navsim/blob/main/docs/install.md, (accessed May 20, 2025), 2024.

[12] NAVSIM Contributors. NAVSIM – Understanding and creating agents. URL https://github.com/autonomousvision/navsim/blob/main/docs/agents.md, (accessed May 20, 2025), 2024.

[13] OpenScene Contributors. The largest up-to-date 3D occupancy prediction benchmark in autonomous driving. URL https://github.com/OpenDriveLab/OpenScene, (accessed May 20, 2025), 2023.

[14] Daniel Dauner, Marcel Hallgarten, Andreas Geiger, and Kashyap Chitta. Parting with Misconceptions about Learning-based Vehicle Motion Planning. In *Proc. Conf. on Robot Learning (CoRL)*, 2023.

[15] Daniel Dauner, Marcel Hallgarten, Tianyu Li, Xinshuo Weng, Zhiyu Huang, Zetong Yang, Hongyang Li, Igor Gilitschenski, Boris Ivanovic, Marco Pavone, Andreas Geiger, and Kashyap Chitta. NAVSIM: Data-Driven Non-Reactive Autonomous Vehicle Simulation and Benchmarking. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[16] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An Open Urban Driving Simulator. In *Proc. Conf. on Robot Learning (CoRL)*, 2017.

[17] Scott Ettinger, Shuyang Cheng, Benjamin Caine, Chenxi Liu, Hang Zhao, Sabeek Pradhan, Yuning Chai, Ben Sapp, Charles Qi, Yin Zhou, Zoey Yang, Aurelien Chouard, Pei Sun, Jiquan Ngiam, Vijay Vasudevan, Alexander McCauley, Jonathon Shlens, and Dragomir Anguelov. Large Scale Interactive Motion Forecasting for Autonomous Driving : The WAYMO OPEN MOTION DATASET. In *Proc. of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[18] Yuxin Fang, Quan Sun, Xinggang Wang, Tiejun Huang, Xinlong Wang, and Yue Cao. EVA-02: A Visual Representation for Neon Genesis. In *Image and Vision Computing*, 2023.

[19] Enrico Fini, Mustafa Shukor, Xiujun Li, Philipp Dufter, Michal Klein, David Haldimann, Sai Aitharaju, Victor G. Turrisi da Costa, Louis Bethune, Zhe Gan, Alexander Toshev, Marcin Eichner, Moin Nabi, Yinfei Yang, Joshua Susskind, and Alaaeldin El-Nouby. Multimodal Autoregressive Pre-training of Large Vision Encoders. 2024. *arXiv:2411.14402v1*.

[20] Cole Gulino, Justin Fu, Wenjie Luo, George Tucker, Eli Bronstein, Yiren Lu, Jean Harb, Xinlei Pan, Yan Wang, Xiangyu Chen, John D Co-Reyes, Rishabh Agarwal, Rebecca Roelofs, Yao Lu, Nico Montali, Paul Mougin, Zoey Yang, Brandyn White, Aleksandra Faust, Rowan McAllister, Dragomir Anguelov, and Benjamin Sapp. Waymax: An Accelerated, Data-Driven Simulator for Large-Scale Autonomous Driving Research. In *Advances in Neural Information Processing Systems*, 2023.

[21] Niklas Hanselmann, Katrin Renz, Kashyap Chitta, Apratim Bhattacharyya, and Andreas Geiger. KING: Generating Safety-Critical Driving Scenari. In *European Conference on Computer Vision*, 2022.

[22] Elmar Haussmann, Michele Fenzi, Kashyap Chitta, Jan Ivanecky, Hanson Xu, Donna Roy, Akshita Mittel, Nicolas Koumchatzky, Clement Farabet, and Jose M. Alvarez. Scalable Active Learning for Object Detection. In *IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[24] Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[25] Edward 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. In *International Conference on Learning Representations*, 2021.

[26] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, Lewei Lu, Xiaosong Jia, Qiang Liu, Jifeng Dai, Yu Qiao, and Hongyang Li. Planning-oriented Autonomous Driving. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[27] Bernhard Jaeger and Andreas Geiger. An Invitation to Deep Reinforcement Learning. 2024. *arXiv:2312.08365v2*.

[28] Aravinda Jatavallabha. Tesla's Autopilot: Ethics and Tragedy. 2024. *arXiv:2409.17380v1*.

[29] Xiaosong Jia, Zhenjie Yang, Qifeng Li, Zhiyuan Zhang, and Junchi Yan. Bench2Drive: Towards Multi-Ability Benchmarking of Closed-Loop End-To-End Autonomous Driving. In *Published in Neural Information Processing Systems (NeurIPS)*, 2024.

[30] Napat Karnchanachari, Dimitris Geromichalos, Kok Seang Tan, Nanxiang Li, Christopher Eriksen, Shakiba Yaghoubi, Noushin Mehdipour, Gianmarco Bernasconi, Whye Kit Fong, Yiluan Guo, and Holger Caesar. Towards learning-based planning: The nuPlan benchmark for real-world autonomous driving. In *Proc. IEEE International Conf. on Robotics and Automation (ICRA)*, 2024.

[31] Puneet Kohli and Anjali Chadha. Enabling Pedestrian Safety using Computer Vision Techniques: A Case Study of the 2018 Uber Inc. Self-driving Car Crash. In *Proc. Future of Information and Communication Conference (FICC)*, 2018.

[32] Youngwan Lee, Joong won Hwang, Sangrok Lee, and Yuseok Bae. An Energy and GPU-Computation Efficient Backbone Network for Real-Time Object Detection. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[33] Norman Lehtomaki, Nils Sandell, and Michael Athans. Robustness results in linear-quadratic gaussian based multivariable control designs. In *IEEE Trans. on Automatic Control (TAC)*, 1981.

[34] Qifeng Li, Xiaosong Jia, Shaobo Wang, and Junchi Yan. Think2Drive: Efficient Reinforcement Learning by Thinking with Latent World Model for Autonomous Driving (in CARLA-v2). In *European Conference on Computer Vision*, 2024.

[35] Quanyi Li, Zhenghao Peng, Lan Feng, Qihang Zhang, Zhenghai Xue, and Bolei Zhou. MetaDrive: Composing Diverse Driving Scenarios for Generalizable Reinforcement Learning. In *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2022.

[36] Zhenxin Li, Kailin Li, Shihao Wang, Shiyi Lan, Zhiding Yu, Yishen Ji, Zhiqi Li, Ziyue Zhu, Jan Kautz, Zuxuan Wu, Yu-Gang Jiang, and Jose M. Alvarez. Hydra-MDP: End-to-end Multimodal Planning with Multi-target Hydra-Distillation. 2024. *arXiv:2406.06978v4*.

[37] Shiyin Lu, Yang Li, Qing-Guo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Han-Jia Ye. Ovis: Structural Embedding Alignment for Multimodal Large Language Model. 2024. *arXiv:2405.20797v1*.

[38] Sivabalan Manivasagam, Ioan Andrei Barsan, Jingkang Wang, Ze Yang, and Raquel Urtasun. Towards Zero Domain Gap: A Comprehensive Study of Realistic LiDAR Simulation for Autonomy Testing. In *Proc. of the IEEE International Conf. on Computer Vision (ICCV)*, 2023.

[39] Andrés Marafioti, Orr Zohar, Miquel Farré, Merve Noyan, Elie Bakouch, Pedro Cuenca, Cyril Zakka, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Vaibhav Srivastav, Joshua Lochner, Hugo Larcher, Mathieu Morlon, Lewis Tunstall, Leandro von Werra, and Thomas Wolf. SmolVLM: Redefining small and efficient multimodal models. 2025. *arXiv:2504.05299v1*.

[40] Eshed Ohn-Bar, Aditya Prakash, Aseem Behl, Kashyap Chitta, and Andreas Geiger. Learning Situational Driving. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[41] 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 *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[42] Qwen Team. Qwen2.5: A party of foundation models, 2024. URL `https://qwenlm.github.io/blog/qwen2.5/`.

[43] Qwen Team and Alibaba Group. Qwen2.5-VL Technical Report. 2025. *arXiv:2502.13923v1*.

[44] Adam Tonderski, Carl Lindström, Georg Hess, William Ljungbergh, Lennart Svensson, and Christoffer Petersson. NeuRAD: Neural Rendering for Autonomous Driving. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[45] European Union. Regulation (EU) 2024/1689 of the european parliament and of the council of 17 may 2024 on laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). In *Official Journal of the European Union*, 2024. URL https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=OJ:L_202401689&qid=1730147197042, (accessed May 20, 2025).

[46] Dequan Wang, Coline Devin, Qi-Zhi Cai, Philipp Krähenbühl, and Trevor Darrell. Monocular Plan View Networks for Autonomous Driving. In *Proc. IEEE International Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[47] Jingkang Wang, Ava Pun, James Tu1, Sivabalan Manivasagam, Abbas Sadat, Sergio Casas, Mengye Ren, and Raquel Urtasun. AdvSim: Generating Safety-Critical Scenarios for Self-Driving Vehicles. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[48] Chuan Wen, Jierui Lin, Jianing Qian, Yang Gao, and Dinesh Jayaraman. Keyframe-Focused Visual Imitation Learning. In *Proc. of the International Conf. on Machine learning (ICML)*, 2021.

[49] Zhenhua Xu, Yujia Zhang, Enze Xie, Zhen Zhao, Yong Guo, Kwan-Yee K. Wong, Zhenguo Li, and Hengshuang Zhao. DriveGPT4: Interpretable End-to-end Autonomous Driving via Large Language Model. In *IEEE Robotics and Automation Letters*, 2024.

[50] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[51] Zetong Yang, Li Chen, Yanan Sun, and Hongyang Li. Visual Point Cloud Forecasting enables Scalable Autonomous Driving. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[52] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. In *IEEE Access*, 2020.

[53] Maryam Zare, Parham M. Kebria, Abbas Khosravi, and Saeid Nahavandi. A Survey of Imitation Learning: Algorithms, Recent Developments, and Challenges. In *IEEE Transactions on Cybernetics*, 2023.

[54] Wenshuai Zhao, Jorge Pena Queralta, and Tomi Westerlund. Regulation (EU) 2016/679 of the European Parlament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation). In *Official Journal of the European Union*, 2016. URL https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679, (accessed May 20, 2025).

[55] Wenshuai Zhao, Jorge Pena Queralta, and Tomi Westerlund. Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey. In *IEEE symposium series on computational intelligence (SSCI)*, 2021.

[56] Brian D. Ziebart, Andrew Maas, J.Andrew Bagnell, and Anind K. Dey. Maximum Entropy Inverse Reinforcement Learning. In *Proc. AAAI Conf. on Artificial Intelligence*, 2008.

Appendix

A.1 Human-Centered AI Self-assessment

Does this activity involve the development, deployment and/or use of Artificial Intelligence-based systems? - **Yes**

1. **Explanation as to how the participants and/or end-users will be informed about**:
    - In this thesis, various autonomous driving agents were evaluated within a simulation environment, live deployment and direct interaction with end-users were not part of the scope. The work explicitly details the role of AI across different stages of the development lifecycle, thereby ensuring that readers are clearly informed about the integration and function of AI throughout the system.
    - The performance of multiple autonomous driving agents was benchmarked, revealing both their strengths and limitations. In addition to individual evaluations, broader challenges such as the sim-to-real gap and the inherent lack of transparency in end-to-end agents were also discussed.
    - The decision-making process was described in detail, including the selection of an appropriate simulation environment and the design of custom models informed by prior findings.
2. **Details on the measures taken to avoid bias in input data and algorithm design**: The dataset was collected across various geographical locations and by multiple drivers to reduce potential bias in the data collection process. Furthermore, a carefully selected data split (navtrain) was used for training, which contains complex and challenging traffic scenarios.
3. **Explanation as to how the respect to fundamental human rights and freedoms (e.g. human autonomy, privacy and data protection) will be ensured**: The dataset used in this thesis is publicly available and contains only the minimal information necessary for training autonomous driving agents, without any additional personal identifiers such as names or addresses. However, it includes visible faces and license plates, which are considered personal data under the GDPR. While this information may not be essential for model performance, blurring such identifiable features would better align the

dataset with GDPR requirements and privacy best practices, especially for any future commercial applications.

4. **Detailed explanation on the potential ethics risks and the risk mitigation measures**: Within the scope of this thesis, the primary ethical concerns addressed were safety risks, potential biases in the dataset, and privacy issues related to data collection. These challenges and their respective mitigation strategies were discussed in detail. However, before any real-world deployment of these models, further ethical considerations must be explored, including decision-making in moral dilemmas and liability in the event of failures or accidents.

Could the AI based system/technique potentially stigmatise or discriminate against people (e.g. based on sex, race, ethnic or social origin, age, genetic features, disability, sexual orientation, language, religion or belief, membership to a political group, or membership to a national minority)? - **Yes**

Although data collection was conducted across multiple locations to improve generalization and reduce bias, fully eliminating bias remains an open challenge that requires further effort. A more comprehensive approach would involve including a broader range of cultures, driving behaviors, traffic regulations, and vehicle types. Additionally, it is important to ensure representation of diverse demographic groups, including different races, ethnicities, and individuals with disabilities, who are currently underrepresented in the dataset.

Does the AI system/technique interact, replace or influence human decision-making processes (e.g. issues affecting human life, health, well-being or human rights, or economic, social or political decisions)? - **Yes**

As long as autonomous driving agents are tested solely within simulators, they do not influence human decision-making. However, the primary objective of such systems is to assist or eventually replace human drivers by providing more consistent and reliable decision-making. This is based on the idea that autonomous agents, when properly trained and validated, can outperform the average human driver in terms of safety, reaction time, and adherence to traffic rules.

Does the AI system/technique have the potential to lead to negative social (e.g. on democracy, media, labour market, freedoms, educational choices, mass surveillance) and/or environmental impacts either through intended applications or plausible alternative uses? - **Yes**

Autonomous driving systems may impact the labor market by reducing demand for driving-related jobs and could raise concerns about mass surveillance due to extensive data collection. Potential misuse or insufficient oversight could affect personal freedoms, privacy, or contribute to environmental damage.

Does this activity involve the use of AI in a weapon system? - **No**

The thesis focused exclusively on end-to-end autonomous driving systems evaluated within simulated environments, without the use of any kind of weapons.

Does the AI to be developed/used in the project raise any other ethical issues not covered by the questions above (e.g., subliminal, covert or deceptive AI, AI that is used to stimulate addictive behaviours, lifelike humanoid robots, etc.)? - **No**

The major potential issues were covered by the questions above. However, given the rapid development of autonomous driving technologies, new ethical challenges may emerge in the future and will require continuous reassessment.

A.2 Hyperparameters

```
train_test_split:
  scene_filter:
    num_history_frames: 4
    num_future_frames: 10
    frame_interval: 1
    has_route: true
    max_scenes: null
  data_split: trainval
agent:
  config:
    trajectory_sampling:
      time_horizon: 4
      interval_length: 0.5
    latent: false
  lr: 0.0001
split: trainval
cache_path: /root/workdir/NAVSIM/exp/training_cache
use_cache_without_dataset: true
force_cache_computation: false
seed: 42
dataloader:
  params:
    batch_size: 64
    num_workers: 4
    pin_memory: true
    prefetch_factor: 2
trainer:
  params:
    max_epochs: 100
    check_val_every_n_epoch: 1
    val_check_interval: 1.0
    limit_train_batches: 1.0
    limit_val_batches: 1.0
    accelerator: gpu
    strategy: ddp
    precision: 16-mixed
    num_nodes: 1
    num_sanity_val_steps: 0
    fast_dev_run: false
    accumulate_grad_batches: 1
    gradient_clip_val: 0.0
    gradient_clip_algorithm: norm
```

Listing A.2.1: Hyperparameters of the Multi-Camera TransFuser Agent, which achieved the best results among custom models in NAVSIM.