



UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO: Autonomous Vehicle Training using a Simulation Framework

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniero en Tecnologías de la Información

Autor/a:

Pedro Estefano Cajilima Cardenaz

Tutor/a:

Israel Pineda, Ph.D.

Urcuquí, mayo 2022

SECRETARÍA GENERAL
ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
ACTA DE DEFENSA No. UITEY-ITE-2022-00020-AD

En la ciudad de San Miguel de Urcuquí, Provincia de Imbabura, a los 9 días del mes de agosto de 2022, a las 13:00 horas, en el Aula S_CAN de la Universidad de Investigación de Tecnología Experimental Yachay y ante el Tribunal Calificador, integrado por los docentes:

Presidente Tribunal de Defensa	Dr. ANTON CASTRO , FRANCESC , Ph.D.
Miembro No Tutor	Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.
Tutor	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.

Se presenta el(la) señor(ita) estudiante **CAJILIMA CARDENAZ, PEDRO ESTEFANO**, con cédula de identidad No. **0104954177**, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, con el objeto de rendir la sustentación de su trabajo de titulación denominado: **Autonomous Vehicle Training using a Simulation Framework**, previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN**.

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

Tutor	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.
--------------	--


Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:

Tipo	Docente	Calificación
Presidente Tribunal De Defensa	Dr. ANTON CASTRO , FRANCESC , Ph.D.	6,0
Tutor	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.	10,0
Miembro Tribunal De Defensa	Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.	10,0


Lo que da un promedio de: **8.7 (Ocho punto Siete)**, sobre 10 (diez), equivalente a: **APROBADO**

Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.


CAJILIMA CARDENAZ, PEDRO ESTEFANO,
Estudiante


Dr. ANTON CASTRO , FRANCESC , Ph.D.
Presidente Tribunal de Defensa


Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.
Tutor


Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.
Miembro No Tutor


MEDINA BRITO, DAYSY MARGARITA
Secretario Ad-hoc

Autoría

Yo, **PEDRO ESTEFANO CAJILIMA CARDENAZ**, con cédula de identidad 0104954177, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autor/a del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urququí, mayo 2022.



Pedro Estefano Cajilima Cardenaz

CI: 0104954177

Autorización de publicación

Yo, **PEDRO ESTEFANO CAJILIMA CARDENAZ**, con cédula de identidad 0104954177, cedo a la Universidad de Investigación de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación

Urcuquí, mayo 2022.



Pedro Estefano Cajilima Cardenaz

CI: 0104954177

Dedication

I first dedicate a fraction of this to the present, past, and future me.

I want to thank me for believing in me.

I want to thank me for doing all this hard work.

I want to thank me for having no days off.

I want to thank me for never quitting.

The remaining 99% I dedicate entirely to my mom and dad for their constant and unwavering support that I will forever cherish.

Pedro Estefano Cajilima Cardenaz

Acknowledgment

Throughout the writing of this dissertation I have received a great deal of support and assistance.

I would first like to thank my advisor, Dr. Israel Pineda, for his guidance in finalizing this project and for being a constant pillar of support throughout this work, both as a professor and as a friend.

I would also like to thank Dr. Eugenio Morocho who helped pushed me through the initial stages of this work and provided useful insights on many of the topics covered in this work.

I extend a special thanks to the team behind the TransFuser project. Thanks to their work, their documentation, and the community behind their open GitHub repository, this project has turned into a fun and interesting dive into autonomous driving research through simulation.

Finally, on behalf of all those who were once undergrad students, I thank Alexandra Elbakyan, the person behind Sci-Hub. Your fight for providing the world with tools for the open pursuit of knowledge is inspiring as is the belief that knowledge must be open to all.

Pedro Estefano Cajilima Cardenaz

Resumen

La conducción autónoma (AD) es un campo de estudio candente ya que los investigadores de diversos campos de estudio están trabajando para contribuir en este campo y avanzar hacia nuevos modelos, métodos y herramientas de última generación, y hacia vehículos autónomos (AV) con automatización completa de nivel cinco. Una forma de categorizar las tareas de AD necesarias para la conducción es definiendo tareas de localización, percepción, planificación y control, cada una de las cuales se compone de subtareas aún más pequeñas y específicas. Para cada tipo de tarea que necesita ser resuelta por un sistema AD, existen múltiples enfoques para manejarla. Debido a este amplio espectro de tareas cubiertas por AD, los campos de investigación que se superponen con AD incluyen, entre otros, visión artificial, teoría de control, aprendizaje profundo y computación de alto rendimiento (HPC).

Un método de aprendizaje comúnmente implementado en trabajos recientes es el aprendizaje por imitación (IL). En el caso de AD, este método de aprendizaje consiste en aprender a imitar una política de conducción experta que se puede representar a través de un conjunto de datos de conducción. Estos conjuntos de datos suelen consistir en información sensorial secuencial percibida a través de sensores desde la perspectiva de un vehículo. Los tipos de datos más comunes que se utilizan en los conjuntos de datos son imágenes percibidas a través de cámaras RGB y nubes de puntos 3D percibidos a través de sensores de detección y alcance de luz (LiDAR).

Otro aspecto importante de AD es el concepto de simulación. Más recientemente, el simulador CARLA se ha convertido en uno de los simuladores de código abierto más populares creados específicamente para el desarrollo de sistemas AD. Además de simular, el proyecto CARLA proporciona herramientas suficientes y accesibles para una variedad de investigaciones relacionadas con la AD. Una de las características clave de CARLA que

ayudan a democratizar la investigación de AD es que se puede utilizar para generar datos de conducción simulados. Esto es crucial para el desarrollo de sistemas AV, ya que generalmente requieren cantidades significativas de datos para entrenar. Esto es especialmente cierto para el desarrollo de sistemas AV de nivel cinco capaces de conducir de manera segura en cualquier tipo de entorno, ya que deberán poder manejar cualquier tipo de situación y escenario que pueda surgir de manera segura y confiable.

Aparte de que la conducción es una tarea compleja (incluso para los humanos), otra de las principales dificultades de la AD que suele acompañarla es que requiere cálculos costosos que no siempre son posibles de implementar en una computadora personal (PC) de uso general. Una solución a esto es hacer uso de sistemas informáticos de alto rendimiento (HPC), como clústeres, que están diseñados para manejar tareas computacionalmente costosas, como las que involucran big data, aprendizaje profundo y optimización de hiperparámetros (HPO) a gran escala.

Teniendo en cuenta el panorama en el que se encuentran actualmente las investigaciones y capacidades de AD, este trabajo se centra en la optimización de un novedoso modelo de última generación llamado TransFuser. El modelo TransFuser tiene componentes de sistema que permiten AD de extremo a extremo, incluyendo entradas sensoriales, un codificador de fusión multimodal, una red basada en GRU para predecir puntos de ruta y un controlador PID para traducir esos puntos de ruta a movimientos de vehículos. La parte novedosa de este modelo se encuentra dentro del codificador de fusión multimodal, ya que utiliza una arquitectura llamada transformador que usa mecanismos de auto-atención para combinar las modalidades de entrada de cámara RGB y LiDAR. Para ello, trabajamos con el marco de simulación CARLA y un sistema HPC para realizar tres experimentos de barrido de HPO destinados a optimizar el modelo TransFuser y comprender la relación entre el rendimiento de conducción y cuatro hiper-parámetros haciendo uso del optimizador AdamW. Los hiper-parámetros de enfoque son la cantidad de capas y cabezas de atención utilizadas por los bloques de transformadores, la tasa de aprendizaje y los tamaños batch. Además, utilizamos el CARLA Leaderboard benchmark para evaluar nuestros modelos entrenados en escenarios de conducción urbanos complejos.

Palabras Clave:

Simulador CARLA, Conducción autónoma, Transformadores, Optimización de hiper-parámetros, Fusión multi-modal.

Abstract

Autonomous driving (AD) is a hot field of study as researchers from varying fields of study are working towards contributing in this field and pushing towards new state-of-the-art models, methods, and tools, and towards autonomous vehicles (AVs) with full level five automation. A way of categorizing the AD tasks necessary for driving are by defining localization, perception, planning, and control tasks, each of which are made up of even smaller, more specific, sub-tasks. For each type of task that needs to be solved by an AD system, there exists multiple approaches for handling it. Due to this ample spectrum of tasks covered by AD, the fields of research that overlap with AD include, but are not limited to, computer vision, control theory, deep learning, and high-performance computing (HPC).

One popular learning approach commonly implemented in recent works is imitation learning (IL). In the case of AD, this learning method involves learning to imitate an expert driving policy that can be represented through a driving dataset. These datasets typically consist of sequential sensorial information perceived through sensors from the perspective of a vehicle. The most common types of data used in driving datasets are image frames perceived through RGB cameras and 3D cloud-point data perceived through light detection and ranging (LiDAR) sensors.

Another important aspect of AD is the concept of simulation. Most recently, the CARLA simulator has become one of the most popular open-source simulation frameworks specifically made for developing AD systems. It is a versatile framework that provides sufficient and accessible tools for a variety of research related to AD. One of the key features of CARLA that help democratize AD research is that it can be used to generate simulated driving data. This is crucial for developing AV systems since they usually require significant amounts of data to train. This is especially true for developing level five AV

systems capable of safely driving in any type of unseen environment as they will need to be able to handle any type of situation and scenario that may arise in a safe and reliable manner.

Aside from driving being a complex task (even for humans), another of the main difficulties behind AD that usually accompanies it is that they require expensive computations that are not always possible to implement on a general-use personal computer (PC). One solution to this is to make use of HPC systems, such as clusters, that are build to handle computationally expensive tasks such as those involving big data, deep learning, and large-scale hyper-parameter optimization (HPO).

Taking into consideration the panorama in which current AD research and capabilities fall in, this work focuses on optimizing a novel state-of-the-art AD model called TransFuser. The TransFuser model has system components that allow for end-to-end AD including sensorial inputs, a multi-modal fusion encoder, a GRU-based network for predicting waypoints, and a PID controller for translating those waypoints into vehicle motions. The novel portion of this model lies within the multi-modal fusion encoder as it uses an architecture called a transformer that uses self-attention mechanisms for combining RGB camera and LiDAR input modalities. For this, we work with the CARLA simulation framework and an HPC system to perform three HPO sweep experiments aimed at optimizing the TransFuser model and understanding the relationship between driving performance and four hyper-parameters, namely the number of attention layers and attention heads used by the transformer blocks, and the learning rate and batch sizes that determine the optimization process with the AdamW optimizer. Furthermore, we use the CARLA Leaderboard benchmark to evaluate our trained models in complex urban driving scenarios.

Keywords:

CARLA simulator, Autonomous driving, Transformers, Hyper-parameter optimization, Multi-modal fusion.

Contents

Dedication	v
Acknowledgment	vii
Resumen	ix
Abstract	xiii
Contents	xv
List of Tables	xix
List of Figures	xxi
1 Introduction	1
1.1 Background	1
1.1.1 Crash Statistics	2
1.1.2 Human Error	3
1.1.3 Autonomous Driving Roadmap	4
1.1.4 Motivation	5
1.2 Problem Statement	6
1.3 Objectives	8
1.3.1 General Objective	8
1.3.2 Specific Objectives	9
2 Theoretical Framework	11
2.1 Autonomous Driving History	11
2.1.1 Historical Context	12

2.1.2	Challenges and Benchmarks	13
2.1.3	The Need for Simulation in Autonomous Driving	14
2.2	Driving Tasks and Learning Approaches	16
2.2.1	Main Driving Tasks	16
2.2.2	Multi-Modal Sensor Fusion	18
2.2.3	Learning Approaches	19
2.2.4	Imitation Learning	20
2.3	CARLA Simulation Framework	22
2.3.1	Client-Server Architecture	23
2.3.2	Town Environment Characteristics	24
2.3.3	Sensor Suite	27
2.3.4	Additional Key Features	30
2.4	Dataset	32
2.4.1	14 Weathers Minimal Dataset	33
2.4.2	Sensor Configuration	35
2.4.3	Expert Policy	35
2.4.4	Routes	36
2.4.5	Scenarios	36
2.5	CARLA Leaderboard Benchmark	38
2.5.1	Evaluation Environment	40
2.5.2	Sensor Availability	41
2.5.3	Score and Infraction Performance Metrics	42
3	State of the Art	47
3.1	Baseline Training Models	48
3.1.1	CILRS	48
3.1.2	AIM	49
3.1.3	Late Fusion	51
3.1.4	Geometric Fusion	53
3.2	Attention and Transformer Architectures	56
3.2.1	Self-attention	56

3.2.2	Attention in Autonomous Driving	57
3.3	TransFuser Model	58
3.3.1	Performances on the CARLA Leaderboard	60
3.3.2	Transformer Module	61
3.3.3	Self-Attention Module	63
3.3.4	Ablation Study	64
3.4	PID Controller	66
3.5	Model Hyper-Parameters	68
3.5.1	Hyper-Parameter Optimization	70
3.6	High Performance Computing	70
3.6.1	Clusters	71
3.6.2	Simulations and Deep Learning in HPC	72
4	Methodology	75
4.1	Phases of Problem Solving	75
4.1.1	Description of the Problem	76
4.1.2	Analysis of the Problem	77
4.1.3	Experimental Design	78
4.2	Model Proposal	81
4.2.1	Self-Attention System Design	82
4.3	Experimental Setup	82
4.3.1	Setting Up the Simulation Environment	83
4.3.2	Parallel Computing Environment	85
4.3.3	Evaluation Environment	86
4.4	Implementation	87
4.4.1	Testing	87
4.4.2	Training Autonomous Vehicles	88
4.4.3	Evaluating Autonomous Vehicles	88
5	Results and Discussion	91
5.1	Baseline Experiment	91
5.2	Transformer HPO Sweeps (Experiment #1)	93

5.3	TransFuser HPO Sweeps (Experiment #2)	96
5.4	Sweep Metrics	101
5.4.1	Training and Validation Losses	101
5.4.2	HPC Usage	103
6	Conclusions	109
6.1	Future Works	111
	Bibliography	113

List of Tables

2.1	CARLA town descriptions.	25
2.2	Type and amount of sensors used for generating datasets.	33
2.3	14 weathers minimal dataset size per town.	34
2.4	Winning teams of the SENSORS track in the CARLA AD Challenge 2020.	40
2.5	Infraction metrics accounted for during each route driven.	43
3.1	Top public submissions to the CARLA AD Leaderboard platform in 2021.	61
3.2	Top public submissions to the CARLA AD Leaderboard platform in 2022.	61
4.1	IDs for the transformer configurations of the first sweep.	79
4.2	IDs for the TransFuser configurations of the second set of sweeps.	80
4.3	Software used for running the CARLA simulator.	83
4.4	Hardware specifications of an A100-SXM4-40GB GPU.	85
4.5	Fixed ports used for running CARLA servers/clients in parallel.	89
5.1	Driving scores of baseline models.	92
5.2	Infraction rates of baseline models per kilometer driven.	92
5.3	Driving scores of transformer configurations.	94
5.4	Infraction rates of transformer configurations per kilometer driven.	95
5.5	Performance averages for each attention layer value tested.	95
5.6	Performance averages for each attention head value tested.	96
5.7	Driving scores of Config-3.	97
5.8	Infraction rates of Config-3.	98
5.9	Driving scores of Config-4.	99
5.10	Infraction rates of Config-4.	100

5.11	Driving scores of Config-6.	101
5.12	Infraction rates of Config-6.	102
5.13	Summary of the best performing models from the TransFuser HPO sweeps.	103
5.14	Average infraction rates for the best performing TransFuser models.	103
5.15	Average training times for baseline and transformer sweep models.	107
5.16	Average training times for TransFuser configurations.	107

List of Figures

1.1	Fatal crashes in the U.S., 1994-2019. Source: [1].	3
1.2	Common taxonomy for levels of driving automation.	4
2.1	CARLA simulator client-server architecture.	23
2.2	14 weather variations in CARLA simulator.	27
2.3	RGB and LiDAR data samples corresponding to one driving frame.	34
2.4	Scenarios being simulated in the dataset. Source: [2].	39
2.5	General structure of the CARLA Leaderboard benchmark.	40
2.6	Examples of a trained autonomous agent being evaluated in Town05 routes.	41
3.1	CILRS model architecture. Source: [3].	48
3.2	AIM model architecture. Source: [3].	49
3.3	Late fusion model architecture. Source: [3].	52
3.4	Geometric fusion model architecture. Source: [3].	53
3.5	Attention function. Source: [4].	57
3.6	Transformer architecture. Source: [4].	58
3.7	GPT model architecture. Source: [5].	59
3.8	Attention maps in AD. Source: [3].	59
3.9	TransFuser model architecture. Source: [3].	60
3.10	Transformer module architecture. Source: [3].	62
3.11	Self-attention mechanism of a TransFuser transformer.	63
3.12	Complete end-to-end TransFuser AD model. Source: [3].	66
4.1	Transformer systems model of the TransFuser model.	81
5.1	Config-4 Parallel Coordinates Chart.	104

5.2	Training Losses vs. Epochs in TransFuser HPO Experiment.	104
5.3	Validation Losses vs Epochs in TransFuser HPO Experiment.	105
5.4	Validation Losses vs Epochs in Baseline and Transformer HPO Experiments.	105
5.5	GPU Usage Percentage vs. Training Hours.	106

Chapter 1

Introduction

The idea of partial or fully autonomous vehicles (AV) capable of analyzing the environment through sensors and assisting human drivers in the decision-making process involved in driving has become of greater interest in recent decades to industrial, robotics, and academic research communities. A more specific line of AV research studies end-to-end autonomous driving (AD) models that input data from multiple sensor modalities and output driving controls through a neural network (NN).

In this chapter, we introduce a general background to the field of AD and the motivation behind this work. We also describe the specific problems this work focuses on, some ongoing challenges, and a justification for the need for high-performance computing (HPC) systems for model optimization.

1.1 Background

Hussain and Zeadally [6] define AVs as computer-controlled vehicles capable of perceiving essential features in the surrounding environment and making driving decisions without the need for any human interaction. Despite state-of-the-art advances in AV technology, traffic congestion in urban environments continues to be a problem as thousands of vehicle accidents continue to occur each day throughout the world, resulting in the direct and indirect reduction of quality of life for many people worldwide. Rather than any specific cause of the accidents that occur, studies have shown that 94% of fatal accidents are a consequence of human errors described as poor decision making by human drivers [7].

In other words, the driving forces for AD research come from the need for modes of

transportation that are safer than human drivers and that optimize time and resources more efficiently [6]. These needs have partly led to a race for mass-producing the first commercial vehicle with full automation. However, current demonstrations from academia and industry have shown that we have yet to reach this level of automation and that further research over an array of AD sub-tasks is required.

In this section, we provide background information on the field of AD, describing motivations pushing toward creating and implementing full AVs as commercial vehicles or public modes of transportation. This information includes crash statistics, the human error aspect of driving, and a roadmap to full AD with levels of automation.

1.1.1 Crash Statistics

A study from the U.S. Department of Transportation Fatality Analysis Reporting System [1] shows that in 2019 there were more than 33,000 police-reported fatal crashes in the U.S. which led to more than 36,000 deaths, including vehicle passengers, motorcyclists, bicyclists, and pedestrians. This report showed that the most common events to take place previous to fatal crashes were collisions with other vehicles and collisions with static objects. The 5.2 trillion kilometers driven in the U.S. in 2019 have resulted in approximately 11 deaths per 100,000 people and 0.69 deaths per 100 million kilometers traveled, which averages to 99 deaths per day. These statistics are far more surprising when realizing that in the past 25 years, the annual number of fatal crashes has continuously fluctuated and has only managed to decrease from about 36,000 to 33,000, as can be seen in Figure 1.1. The number of fatal crashes in 2016 reached a recent apex of almost 35,000 fatal crashes, showing that despite efforts made within this 25-year time frame to improve road safety, there is still a relatively high rate of accidents taking place every day.

The World Health Organization reported in 2021 [8] that approximately 1.3 million people in the world die per year from traffic accidents. Road crash injuries are the leading cause of death for children and young adults between five and 29 years. Furthermore, between 20 and 50 million more people suffer from non-fatal injuries caused by road crashes, directly and indirectly leading to economic losses, disabilities, and loss of productivity, all of which ultimately reduce the quality of life for many people.

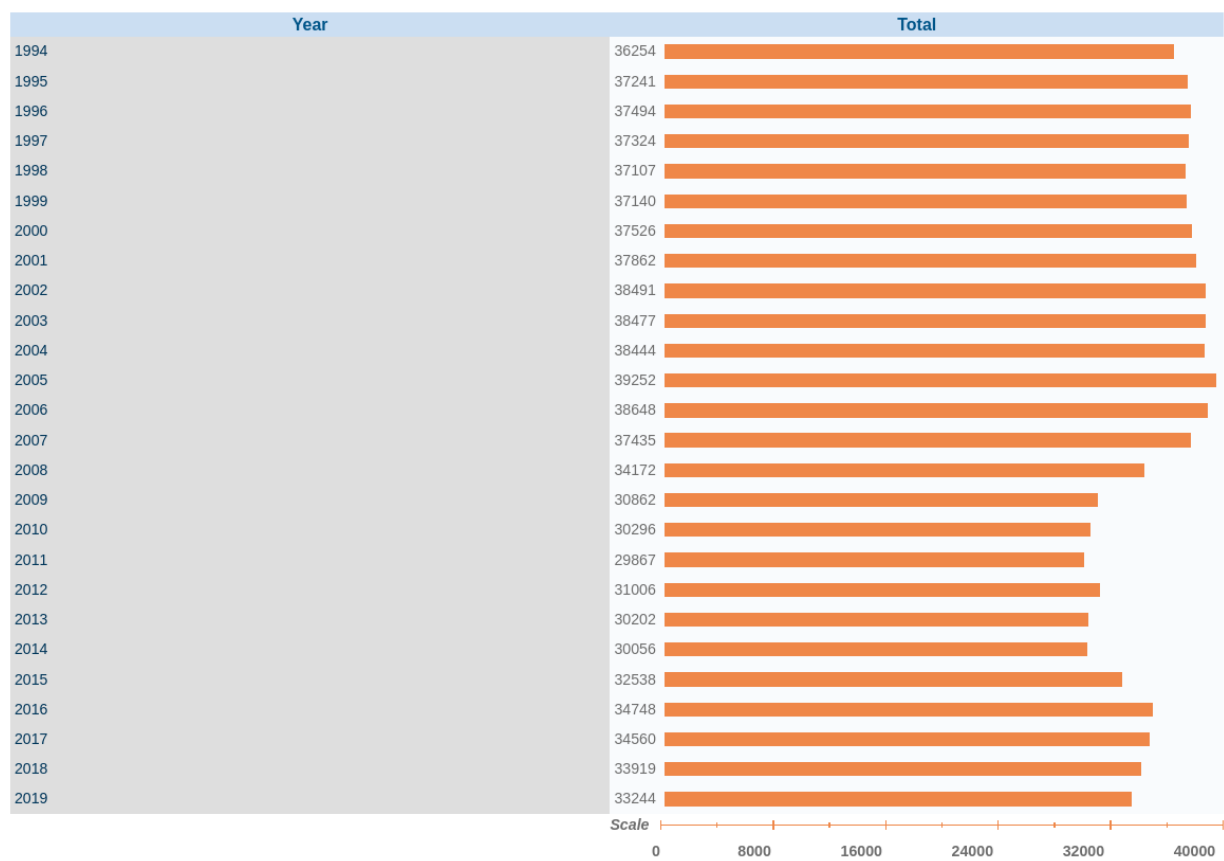


Figure 1.1: Fatal crashes in the U.S., 1994-2019. Source: [1].

1.1.2 Human Error

Human errors can be a person falling asleep behind the wheel, failing to see an oncoming obstacle, not understanding a street sign, or driving while intoxicated from alcohol. In environments with multiple driving agents, these errors are an additional factor of uncertainty that leads to more human errors resulting from poor reactions to these unpredictable events. Understandably, neither humans nor machines can make the best decisions 100% of the time in uncertain and continuously evolving driving environments. Still, we can apply improvements to both the human and machine aspects of driving.

At a sociopolitical level, laws and road regulations concerning the human aspect of driving are high-level solutions for creating safer road environments. Still, not all humans abide by said laws and regulations. The precautions taken at an industrial level have done little to none to remove the human error factor involved in driving. On the other hand, AD is a solution that seeks to mitigate, or better yet remove, the human error from the

task of driving.

1.1.3 Autonomous Driving Roadmap

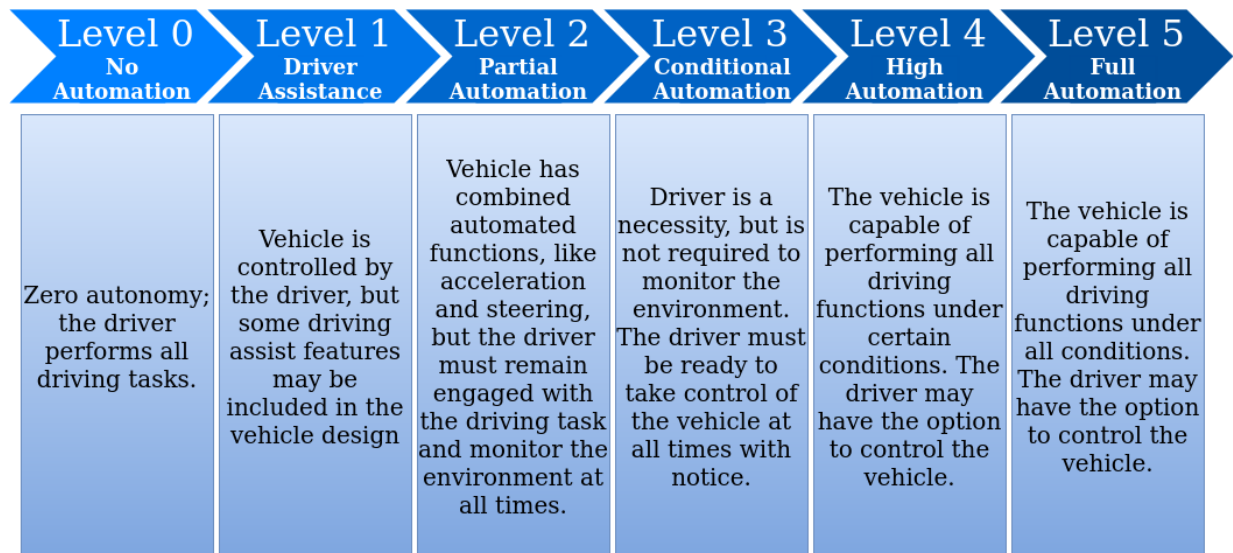


Figure 1.2: Common taxonomy for levels of driving automation.

Although the AV idea did not necessarily arise from a need to reduce the dangers and risks accompanying human drivers, it serves as a low-level solution to minimize most road dangers and accidents deriving from poor human decision-making. This low-level solution can be divided and categorized into the levels of automation seen in Figure 1.2. This taxonomy standardized by the Society of Automotive Engineers (SAE) [9] serves as a roadmap for AVs to follow before fully autonomous systems capable of safely driving in any environment and scenario without any human interaction enter the world.

From an industrial standpoint, automotive tech giants such as Tesla and Waymo are leading the race for creating level five AVs, having developed commercial vehicles with level two automation and taxi-service vehicles with level four automation. Even though both companies implement some level of automation in their vehicle designs, there are still many barriers to overcome before reaching an era where full AVs prove themselves to be a far safer alternative to human drivers. Some studies forecast that vehicles with level five automation systems will not arrive until at least 2030 [10]. However, forecasts on the arrival of AVs have proven to be wrong in the past, so it is plausible to believe that they will not arrive until years after the predicted date. In the meantime, research in the field

will continue to push forward advances that will eventually make vehicles with level five automation a reality.

1.1.4 Motivation

Intelligent cities apply innovative technologies to tackle the economic, social, and ecological challenges of future cities. Recent studies have planned for the development of smart cities [11], whose main objectives are to improve the quality of urban services and the overall quality of life. These studies also suggest that one of the cornerstones needed to create a sustainable smart city is the concept of smart mobility, which constitutes a range of alternative modes of transportation, including AVs. As cities continue to grow in density, so will the number of traffic congestions caused by current inefficient transportation networks. Thus, smart mobility and AD will become essential components for managing these swelling urban populations by reducing the human error factor involved in driving.

Further proof of the growing spotlight around AD research is seen in recent competitions and benchmarks centered on developing and ranking the top-performing AD models, such as the CARLA Challenge [2]. Naturally, public interest in AD will grow while AV systems improve and expand to new limits, eventually revolutionizing modes of transportation. With both academia and industry especially interested in improvements being made in AD research, it is only a matter of time before large-scale plans to implement level five AVs on public roads come to fruition.

With this background in mind, the primary motivations for this research work are summarized as creating efficient and safer modes of transportation with less fatalities, and motivating for further AD research to be carried out. There are numerous AD topics and regions which require further study, and this work may also serve as a stepping stone for future studies. By focusing on specific driving sub-tasks and driving environments and implementing the right set of tools, we can work towards pushing current level four or five AD capabilities.

1.2 Problem Statement

The emergence of full AVs appears to be a sure thing while it remains an active line of research. However, AD technology is typically made to operate in select regions prepared for handling that technology. In the case of less developed regions where AD research is close to non-existent, such as South America, it will likely take more time for that technology to catch up. Part of this drawback stems from AD models being dependent on their training data. Since it is difficult to represent the entire complexity of driving in every region of the world in a single driving dataset, level four AV systems are generally made considering the climate and driving norms of hand-picked regions where they will function. Therefore, level four AVs will remain limited to thriving in regions whose driving data was collected for training. This is especially true if the goal is to someday implement AV technology in less-studied driving regions, such as in South America, that are not amongst the principal actors in AV research and would require a greater deal of work to implement in local streets and cities.

AD systems constitutes of multiple tasks with different types of methods that can be applied to solve each task. Training a system to drive autonomously involves enabling a system to handle tasks stemming from computer vision, planning, and control theory problems. Due to this complexity, full level five AD is a difficult task to teach to a machine system for a number of reasons. In this section, we describe the problems related to training AVs with a special focus on multi-modal perception, computational costs, and hyper-parameter optimization (HPO).

First, there is a high degree of uncertainty in real-world driving environments due to the high level of scene variability and unpredictability. Outcomes of early AD challenges [12, 13] showed that participating vehicles faced multiple sources of errors stemming from sensor noise, environment complexity, and system failures, all of which add to the overall level of uncertainty that each vehicle handles. The uncertain dynamics of driving environments has endured as an AD challenge as dynamic road scenarios are commonly associated with lower performance accuracy than their less dynamic counterparts [14].

The ego-vehicle, the vehicle of primary interest during development, must be capable of maneuvering safely in countless driving environments, scenarios, and uncertainties that

could present themselves at any instance during a drive in order to classify for level five automation. However, despite the advances that large companies such as Tesla and Waymo have applied to some of their most advanced AV models, state-of-the-art AVs still require human interaction during a real-world driving episode or are limited to driving in specific environments. Therefore, they do not classify for level 5 automation neither.

Testing for full level five automation is a task additional to the training of AD systems that adds a level of complexity to the development process. Naturally, the computational cost required for evaluation increases as more driving scenarios are tested. However, since there is no efficient or standardized way for testing an AD system in every possible driving scenario, an alternative is to evaluate an AD system in a fixed number of settings and conditions. Ideally, the more scenarios used for development the better. Nevertheless, time and computational resources are two key factors that limit the amount of models that can be trained and evaluated. The convergence between HPC and artificial intelligence then becomes more crucial as they can be used in unison for developing intelligent AD agents [15]. The challenge then becomes in making use of HPC systems and methods to solve expensive optimization problems related to AD.

Considering this computational limitation, the main problem that this work focuses on is the HPO of an imitation learning (IL) AD model for a sub-task known as multi-modal fusion. This sub-task is a perception task that applies to AVs with the peculiarity of using different types of sensors. The most commonly studied sensor types for AD are RGB cameras that produce images in \mathbb{R}^2 and light detection and ranging (LiDAR) sensors that produce cloud-point data in \mathbb{R}^3 . Multi-modal fusion deals with combining the information from different sensor modality representations by implementing one of three paradigms of fusion commonly referred to as early fusion, deep fusion, late fusion, or combinations of those paradigms [16]. Regarding HPO, we are especially interested in the problem of how the number of attention layers, attention heads, batch size, and learning rate hyper-parameters relate to the problem of multi-modal fusion and perception in terms of compared benchmarked performance.

Real-world deployment of full AVs has taken longer than expected, especially when considering the importance of operational safety to general driving standards. Despite advances, multi-modal fusion models still tend to experience performance degradation when

in the presence of challenging driving scenarios which can sometimes be brought by simple changes in scenery [17, 18]. In other words, there is a need to improve the software-learning aspect of current AV models so that they can drive further distances while committing fewer road infractions.

In 2021, a multi-modal fusion model called TransFuser was presented as a novel approach for using transformers to encode the global context of a 3D scene perceived through multiple sensor modalities [3]. It also introduces attention-based hyper-parameters tunable through HPO for the task of multi-modal fusion in AD. Since then, TransFuser implementations have shown improving results in recent submissions to the official CARLA Leaderboard platform [2]. However, literature outlining how the tuning of certain hyper-parameters affects the capacity of driving models to handle challenging driving scenarios is still relatively limited.

This lack of research hampers the driving potential of this state-of-the-art model and is partly attributed to AVs commonly requiring data-intensive computations to train. Most personal computer (PC) systems are not made for efficiently handling the strain of data-intensive computations that usually accompany AD models, limiting the amount of testing that can be done on a PC. Considering the data-intensive nature of AD models, this limitation makes HPC systems and methods necessary for large-scale AD development and HPO.

Against this backdrop, this thesis project aims to highlight the relationships between novel attention model components and the AD task of fusing sensor modalities to drive in complex urban scenarios. We are also concerned with the overall model performance and training efficiency of the tested models and variations. Performance can be measured through benchmarks, while training efficiency can be measured through system metrics provided by the HPC system.

1.3 Objectives

1.3.1 General Objective

- Optimize the hyper-parameters of the state-of-the-art TransFuser model using a CARLA simulation framework running on an HPC system.

1.3.2 Specific Objectives

- Perform large-scale HPO experiments on AD models by implementing the CARLA simulation framework on an HPC system.
- Train three instances of four AV baseline models and the TransFuser model with default configurations.
- Perform three grid-searches to hyper-tune the transformer architecture of the TransFuser model resulting in nine trained model variations for each grid-search.
- Perform three grid-searches to hyper-tune the learning rates and batch sizes of the three best-performing transformer variations resulting in 54 trained variations of the TransFuser model for each grid-search.
- Monitor the hyper-parameters and system metrics of the second grid-search through a machine learning tool called WandB.
- Obtain performance metrics for all trained models using the CARLA Leaderboard benchmark.

Chapter 2

Theoretical Framework

As significant developments have been made in the past decades regarding computer vision, machine learning, and simulations, researchers have commonly begun integrating newer machine learning methods into other computer vision-related tasks such as AD. Convolutional NNs (CNNs) are an example of this as they are components frequently implemented in AD models and learning approaches due to their revolutionary impact in pattern recognition [19]. Now, thanks to open-source simulation frameworks, experimenting with AD models and agents is more straightforward and more accessible than it was less than a decade ago. For example, the CARLA framework allows the development of autonomous agents, generation of driving data, and evaluation of driving performances.

This chapter describes the theoretical baselines needed to understand better how we train and evaluate AV models using the CARLA simulation framework. We first provide a historical context of AD, leading up to AD tasks, and common learning approaches to end-to-end AD. Then, we describe the CARLA simulation framework consisting of a driving simulator [20], an open-source dataset of simulated driving in 14 weather conditions [3, 21], and a benchmark for evaluating driving performance [2]. Chapter 3 then complements this chapter by describing the AD models studied in this work.

2.1 Autonomous Driving History

With both industry and academia focused on developing reliable full AVs, we can imagine reaching a point where steering wheels are no longer a necessary part of a car. Whether or not this happens, any possible future with AVs will only be thanks to the historical progress

and milestones that have paved the way for current AV capabilities and innovation. Since the emergence of the first AV systems, models have gone from using NNs in AV systems to implementing the latest technologies and system models such as HPC and transformers. For a better context of how AD has evolved, this section presents a general panorama of AD development in the past 70 years. We highlight historical challenges and benchmarks and justify the need for simulations in AD research.

2.1.1 Historical Context

1930s - 1990s Science fiction literature of the early 20th century gave birth to the idea of a vehicle capable of driving itself. It was not until the 1930s that Norman Bel Geddes materialized this idea by depicting an autonomous car at the 1939 World's Fair in New York. This exhibit outlined his vision of a world where humans are removed from the driving process. In the 1950s, a joint initiative by General Motors and RCA Laboratory became the first of many to begin a phase of AV research and development.

In the years to follow, numerous technological research and development programs throughout the U.S., Europe, and Asia took place under governmental and academic institutions to work on smart vehicle systems, and driving scene recognition through image processing [22]. In an essay from 1969 [23], John McCarthy described the concept of an AV as a vehicle capable of navigating itself based on two inputs, the same visual input that is available to a human driver and the destination input of the user. This definition of AV would become a standard for future research and development.

The rest of the 20th century was filled with pioneering AD research resulting in key milestones such as the cruise control feature, the use of LiDAR technology for distance control, and prototypes that made use of NNs to input raw road images and output steering controls in real-time [24]. Here, the introduction of artificial intelligence to AV systems would come to be a crucial factor that revolutionized the field of AD and further propelled autonomous capabilities to parking, maneuvering a moving vehicle, and rationalization of a driving scenario [25].

2000s - 2022 (Present day) From 2004 to 2007, the Defense Advanced Research Projects Agency (DARPA) held three AV Challenge Programs in the U.S. that helped

to motivate and accelerate research in AVs, increasing interest in academic and industrial environments. It was in this decade that large automobile and tech companies such as Volvo and Google began to undertake their journeys into the race for developing full AVs [22]. Since then, many other car companies have joined in the race by developing their own AV technologies and releasing car models with advanced driver support systems ranging at a level one automation for tasks such as collision avoidance through advisory and warning features, parking, and lane-keeping [26].

In 2016, after the SAE taxonomy for autonomous levels of driving (Figure 1.2) became widely accepted, the world saw its first fatal accident involving an AV, followed by the first pedestrian fatality in 2018. Naturally, concerns regarding the legal status of AVs continued to grow, meaning that cities had to adapt to this rapidly evolving technology, both on a technical and a legal scale. An inventory of cities around the world shows that as of 2019, 56 cities have been hosting AV tests or have committed to doing so in the near future, while 40 cities have been undertaking extensive surveys to regulate, govern, and plan for issues raised by AVs [27].

By this point, cars with level two automation have already reached the market. Some Tesla vehicle models include an autopilot mode capable of adaptive cruise control, acceleration control, and emergency braking. An enhanced version of the Tesla autopilot mode also supports lane changing and autonomous parking. In 2021, some car companies began leasing car models in certain areas with government-approved level three automation technology. Legally, the drivers of these vehicles are allowed to take their eyes off the road but must be ready to take manual control if needed. Examples of this are the Honda Traffic Jam Pilot driving technology in Japan and Mercedes-Benz Automated Lane-Keeping System in Germany.

2.1.2 Challenges and Benchmarks

Challenges and benchmarks are methods for measuring the performance of a machine learning system. In the case of AD, they measure the driving capabilities of an AV system given a specific task and driving scenarios. Here we present some of the most historically relevant AV challenges and benchmarks in the past two decades to demonstrate the evolving tendencies of AD research and development.

- **DARPA Grand Challenge (2004)** The task was to construct a vehicle that could autonomously drive 241 km across the Nevada desert where no team completed the task. Only five vehicles traveled more than 1.6 km, with the farthest-traveling vehicle traveling just over 11 km. Since no winner was declared, the challenge was rescheduled for the following year [12].
- **DARPA Grand Challenge (2005)** This task was to complete a 212 km driving course. Five out of 23 finalists completed the course with a mean mileage between occurrences of significant errors or failures equal to 193 km [12].
- **DARPA Urban Challenge Event (2007)** Six out of 11 finalists successfully finished the 97 km urban course with a mean mileage between occurrences of significant errors or failures equal to 161 km [13]. Despite this being a significant event at the time, the course lacked real-world urban scenes such as pedestrians and cyclists.
- **VisLab Intercontinental Autonomous Challenge (2010)** A challenge based on GPS waypoints tested four AVs in real traffic conditions on a 13,000 km intercontinental trip over three months. The goal was to check and improve the performance of AVs developed by VisLabs and recollect driving data for future research (about 50 TB was recollect) [28].
- **The KITTI Vision Benchmark Suite (2012)** A benchmark suite that used an AD platform to develop benchmarks for computer vision perception tasks such as object detection [29]. It was the first time that deep learning was used to solve an AD task.
- **CARLA Challenge (2019)** An online driving challenge based on the CARLA simulator that evaluates AD models under the same set of simulated driving scenarios and tasks [2]. This challenge is held yearly and has seen improvements in the performance of AVs as well as growing interest amongst the research community.

2.1.3 The Need for Simulation in Autonomous Driving

Driving environments are a must-have for developing and testing AD systems where safety is crucial. However, some have argued that hundreds of millions of driven test miles are

needed in order for an AV to be statistically more reliable than a human driver at handling complex traffic situations [30]. This requirement becomes challenging and expensive with real-world driving when replicating traffic situations over a long period of time. Considering the vast number of situations that may arise when driving, the development of reliable level five AV systems depending solely on real-world settings is slowed by the expensive and time-consuming task of physically collecting driving data [31].

Driving simulations are alternative solutions that have become popular research tools in recent years. They also serve as a sandbox for developing, training, and testing AD systems. In doing so, they bring the benefit of improving upon the development, quality, and confidence of AD models [32]. They are synthetic virtual environments typically generated by computer graphics, physics-based modeling, and robot motion planning techniques. The simulation process helps separate the algorithmic aspect of AD from the hardware, allowing researchers to focus solely on improving the AD system models that will eventually make AVs a safer mode of transportation than human drivers. The use of simulated environments for AD serves two primary purposes. First, to test and validate the capability of AD vehicle systems in the context of perception, navigation, and control. Second, to generate large amounts of labeled driving data valid for training machine learning models [33].

Waymo, the company owning a fleet of some of the most advanced AV systems globally, has admitted to having driven more than 32 million kilometers since the origins of the company [34]. With half of that total distance driven in just over a year since the company started providing autonomous taxi services to the public, they have also driven approximately 32 billion kilometers through simulations. In this manner, Waymo has obtained sufficient driving data to test and implement their fleet of autonomous taxis. This example indicates a clear benefit for accelerating AD research through simulations. This way, the pursuit of developing systems capable of driving further distances while committing fewer road infractions is also accelerated.

Even though the hardware related to AVs has become cheaper and more accessible over the years, the software aspect still needs improvement to make safe and reliable AVs. This challenge is directly related to making a practical AV programmed to handle multiple scenarios and challenges that can arise while driving. Simulations provide an

alternative and more scalable method for designing complex environments for AVs to drive in. Depending on the necessities of the user, the simulated environments can be made to include complex social interactions, traffic congestion, crashes, infrastructure conditions, poorly marked roads, and rough surfaces, among other elements that make up countless possible driving scenarios. Therefore, simulations are necessary tools for the large-scale implementation of AVs and for making AD research more accessible to people.

2.2 Driving Tasks and Learning Approaches

AD is a complex task consisting of various smaller sub-tasks that need to be solved for end-to-end driving. AD systems are capable of handling a wide range of sub-tasks, including but not limited to motion planning, vehicle localization, pedestrian, traffic sign and road-marking detection, automated parking, vehicle cybersecurity, and system fault diagnosis [35]. When considering the role of AVs in the concept of smart mobility, these sub-tasks extend to fuel efficiency, adaptive cruise control with queue assist, platooning, and safe maneuvering through lanes and intersections [36]. The list of sub-tasks becomes even more extensive when considering connected AVs that need to manage communication with any entities that can affect or be affected by the vehicle. This communication is known as vehicle-to-everything, where communication may extend to other vehicles, infrastructures, networks, pedestrians, and devices [36, 11].

We generalize the complex task of driving into four main sub-tasks: localization, perception, planning, and control. This section provides a comprehensive overview of these main tasks and the algorithms and approaches for handling most AD tasks and learning desired driving behaviors. Specifically, the model architectures implemented in this work focus on solving the perception task of multi-modal sensor fusion for end-to-end AD.

2.2.1 Main Driving Tasks

For the scope of this work, we consider the essential operational functions that an independent AV must be able to perform to navigate through a given scenario. With this in mind, this subsection classifies AD capabilities into four main task categories: localization, perception, planning, and control. We also present examples of algorithms and methods

used for each of these four tasks.

Localization Localization is the task of a vehicle recognizing itself and approximating its location and position in an environment. Usually, this information is presented through x - y coordinates and obtained through global navigation satellite system (GNSS) and inertial measurement unit (IMU) sensors. However, since these measurements can be prone to inaccuracies due to nearby structures, localization approximations can be improved by using additional sensors and map-matching. One technique is the Adaptive Monte-Carlo Localization algorithm that uses a dynamically-adjusted particle filter to track the position of a vehicle with respect to a map [35]. There also exists a family of algorithms commonly used for environment mapping and vehicle localization known as simultaneous localization and mapping (SLAM) algorithms [37].

Perception Perception is the task of perceiving the environment surrounding a vehicle and deriving some understanding from it. Sensors are indispensable for this task as they allow vehicles to obtain different data types from their environments, such as 2D RGB images and 3D point clouds. Being one of the more complex AD tasks to solve, perception can be split into a series of computer vision sub-tasks, including traffic lane recognition, semantic image segmentation, object detection, and sensor fusion.

The perception algorithms that can be applied can vary depending on the specific sub-task that needs solving. However, these are typically classified into three groups of algorithms. There are general machine learning algorithms such as AdaBoost and SVM methods combined with feature extraction algorithms, artificial NNs (ANNs), and deep learning methods including CNNs. SLAM algorithms can also help gain perception about an environment, especially when combined with another family of algorithms called the detection and tracking of moving objects (DATMO) algorithms [37].

Planning Planning is the task of deciding how a vehicle should react, given a perceived scenario or a current state. Planning involves deciding all the low-level actions that the vehicle would need to take, such as steering, braking, and accelerating, and higher-level decisions, including route planning and lane-changing. The methods for planning and making decisions can be classified into four groups. There are route planning algorithms

that can be used assuming a map layout is known a priori, methods for planning the movement and control of vehicles such as the holistic end-to-end planning approach, and decision-making mechanisms based on deep reinforcement learning, decision trees, and Markov decision processes [35]. Of these methods, the holistic end-to-end method is among the most commonly used approach as it simplifies the number of algorithms needed by combining the tasks of perception, planning, and control into one, helping to reduce the average execution time of the planner [37, 35].

Control Control is a control theory task of implementing the necessary low-level vehicle motion controls to carry out the decision results from the planning task. Here the low-level controls depend on the output from the previous planning phase. If, for example, a sequence of waypoints were output, then a method for converting those waypoints into actions would be needed. If, on the other hand, vehicle controls were output, then the task would consist of applying those controls to the vehicle.

Several algorithms can carry out the control task depending on the desired motions and the driving goals. For example, some algorithms prioritize safety while others may prioritize low computational cost and fuel efficiency. One approach is to divide the vehicle controls into different controllers for different tasks, such as lateral and longitudinal controls and steering controls. Then, for each control, different algorithms can be applied, such as a tunable proportional-integral-derivative (PID) controller for carrying the necessary motions for each control of the ego-vehicle [35].

2.2.2 Multi-Modal Sensor Fusion

One sub-task for AD perception is the fusion of data from different sensor modality representations. The three paradigms for fusion are early fusion, deep fusion, and late fusion [16]. Here early fusion can refer to fusion methods that combine LiDAR data at the raw data level and camera data at the data or feature level. Late fusion methods fuse the output results from different modalities. Meanwhile, deep fusion fuses cross-modal LiDAR data at the feature level and camera data at the data and feature level. In this context, finding optimal fusion methodologies become a research challenge for AD.

The first models to implement fusion methods on multi-sensor vehicles included end-to-

end architectures that exploit continuous convolutional blocks to fuse image and LiDAR representations for the AD tasks of 3D object detection, ground estimation, and depth completion [38, 39]. This fusion process commonly involves encoding continuous streams of geometric information and applying some fusion method. Since driving data is sequential information, other early approaches used recurrent NN (RNN) architectures such as long short-term memory that allowed for modeling the input and output sequences of data [40]. Furthermore, ever since the integration of attention mechanisms and CNNs grew in computer vision tasks such as image classification [41] and object localization [42], the use of attention-based methods has spread to AD tasks such as object detection [43, 44], motion prediction [44, 45], lane changing [46], and sensor fusion [3].

2.2.3 Learning Approaches

For a system to have learned some aspect of driving, it must be capable of receiving sensory inputs and outputting a driving action or decision. Studies show that computer vision is an essential part of improving sensorimotor systems since models that use explicit intermediate representations train faster, achieve better performance, and generalize better to new environments [47]. However, the learning methods they use to learn from inputs can differ. There are two main approaches used to create complete AD systems, the modular pipeline approach and the end-to-end learning approach. This subsection provides examples for both.

Modular Pipelines Modular pipeline systems [48, 49] are the traditional approaches for creating AD systems. Also known as the mediated perception approach by some works [50], modular systems divide the general task of driving into various sub-tasks. These sub-tasks generally involve localization, perception, planning, and control, each of which links sensory inputs to motion outputs and can be further divided into more specific tasks such as object detection and road planning [51].

This modular way of designing systems is advantageous as reliable architectures can be built by placing specific algorithms for handling each sub-task problem that requires solving. Since there is more accumulated research on the specific sub-tasks related to AD than on the general task of AD, it is also beneficial to use modular components that have

been researched and tested more extensively. However, the major flaws of modular systems are that they are prone to error-propagation and over-complexity [51], causing researchers to look for more optimal systems capable of performing just as well as, if not better than, modular systems.

End-to-end Learning End-to-end driving, also known as the behavior reflex approach by some works [50] combines the general task of driving into a single learnable system architecture by directly mapping input sensory data to motion outputs. While end-to-end models essentially handle the same sub-tasks as modular pipelines, they differ in how interconnected an end-to-end learning pipeline is versus a modular pipeline. End-to-end models date back to the ALVINN [24] vehicle in 1988 and have become more feasible solutions since then, thanks to advances in ANNs and deep learning [51].

In practice, end-to-end architectures are developed mainly with three approaches for end-to-end learning. These approaches include direct supervised learning that requires ground-truth knowledge [52, 53], reinforcement learning that learns the optimum way of driving by maximizing future rewards [54, 55], and neuroevolution that uses evolutionary algorithms to train ANNs without the need of backpropagation [56].

Each approach represents three fundamentally distinct forms of learning and is capable of performing AD tasks. Naturally, each approach functions distinctly and has some advantages and disadvantages. For example, while the direct supervised learning approach can imitate an expert driver with off-line training, these methods tend to have trouble with environment generalization. Furthermore, while deep reinforcement learning and neuroevolution offer novel solutions to AD, they both require training in an online environment through repeated trial and error until obtaining the desired behavior. Also, urban driving and real-world driving are still in the early stages of development with the deep reinforcement learning and neuroevolution approaches, respectively [51].

2.2.4 Imitation Learning

IL is an area of machine learning where an agent attempts to learn an optimal policy that best imitates the desired behavior from a set of demonstrations provided by an expert. In the context of end-to-end AD, IL has served for mapping perceptual inputs to control

commands in lane following [57] and obstacle avoidance [58] tasks. Considering that AD handles perceptually sequential data, it is common for IL system models to implement a CNN, an RNN, or a multi-layer perceptron (MLP) in their system designs.

Behavioral Cloning Behavioral cloning is one approach to end-to-end IL based on supervised learning. Given a set of training trajectories obtained off-line by an expert driving policy where each trajectory consists of a sequence of observations and expert actions, the goal of this method is to train a target policy to mimic the actions of the expert driving policy based on the collected observations [59].

Limitations Limitations that made it challenging to apply behavioral cloning to the general task of AD were noted in early studies of IL applied to AD. For example, when the ALVINN vehicle [24] reached a forked road, the network outputted two discrepant travel directions. This output indicated that the optimal action cannot always be inferred from perceptual input alone when driving, meaning that an AV will always have trouble deciding which direction to take at an intersection. Also, if the capabilities of an AV are limited to lane following, it is unable to turn from one road to another without the interaction of a human driver [57].

Another one of the main known limitations to this approach is the distributional shift between training and testing distributions resulting from a vehicle handling unseen complex driving scenarios. Other problems beyond AD that limit the generalization capabilities of an AD model are high variances, dataset biases, and causal confusion. In IL models, there tend to be high variances in performance resulting from sensitivity to random network initialization methods, and data sampling order [17]. They have also shown to be sensitive to dataset biases as the learned policies tend to be dominated by the training datasets [60]. Causal confusion is another problem related to dataset bias sensitivity where spurious correlations are not distinguishable from real causes in observed training demonstration patterns [61]. These models have also shown to suffer from a sensitivity to network initialization, and sampling order [62].

Conditional IL Due to the limitations of not being able to control trained policies during test time, conditional IL (CIL) models were proposed as improvements to IL models

[63]. Unlike typical IL methods, CIL methods resulted in models capable of responding to navigational commands during test time. During training, models are conditioned on high-level navigational commands such as "turn right" or "follow lane." Given an expert driving policy π^* and a dataset $\mathcal{D} = \{ \langle o_i, c_i, a_i \rangle \}_{i=1}^N$ of size N where o_i are the sensory data observations, c_i the high-level navigational commands, and a_i the resulting vehicle actions, the goal of CIL is to learn a policy π parametrized by θ to mimic the behavior of π^* based on o_i and c_i . In order to achieve this, the best parameters θ^* can be found by minimizing a loss function \mathcal{L} . This IL process is generalized in the following equation,

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_i \mathcal{L}(\pi(o_i, c_i; \theta), a_i) \quad (2.1)$$

Initial studies have shown that CIL combined with deeper residual networks can improve over other state-of-the-art architectures in terms of the generalization capabilities of a trained model. They have also shown that learning can be scaled to large demonstration datasets. However, dataset bias is still a problem as causal confusion can hinder the performance of a model, especially with large-scale datasets. Also, these models are still prone to high variances and require multiple runs to find the best possible policies. Finally, as with all policy learning methods, environments with more multi-agent dynamics can negatively affect CIL models as they are not capable of capturing and generalizing these types of dynamics [17].

2.3 CARLA Simulation Framework

Although several AD frameworks have been proposed [64, 65] and compared [51, 66] in other works, amongst the most popular and state-of-the-art include the CARLA driving simulator [20]. CARLA (Car Learning to Act) is a high-fidelity open-source driving simulation framework built to support the training, prototyping, and evaluation of AD systems in urban environments. The free and open-source nature of CARLA makes it a flexible and accessible framework that helps democratize AD research and development, allowing for a great deal of customization and control over environmental factors such as the weather and time of day.

Another benefit of CARLA is that it supports detailed benchmarking of driving poli-

cies by providing feedback through detailed descriptions of committed infractions such as collisions and other traffic violations. Furthermore, it provides flexible support for setting up experimental suites that can be configured and used to train any modular pipeline, IL, or reinforcement learning approach and generate large amounts of labeled driving data.

This section describes the CARLA simulation framework used in this work, following the ideas stated in the original CARLA paper [20] and in the respective online documentation of the simulator. This includes descriptions of the client-server architecture implemented by the CARLA simulator, the eight simulated town environments available for training and testing, including the actors populating those worlds, and the available sensor suite for the ego-vehicle. Other aspects of this framework that require more definition for evaluating purposes are further explored in Section 2.4.

2.3.1 Client-Server Architecture

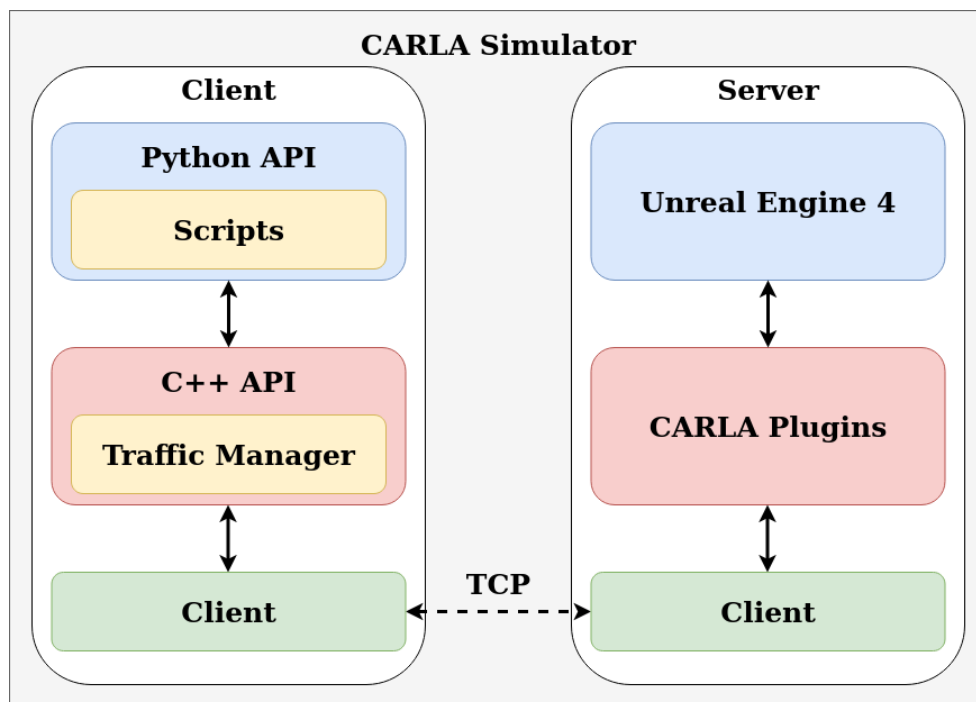


Figure 2.1: CARLA simulator client-server architecture.

CARLA is built using Unreal Engine 4, a tool for creating high-quality 3D environments with realistic vehicle physics and various plugins. Also, CARLA uses Python and C++ APIs to grant control over the simulation. The CARLA simulator counts with two driving

modes: standalone and server modes. The standalone mode is a video-game-style mode where a vehicle can be controlled manually and driven freely. This mode serves to explore the layout of the available town maps, weather conditions, vehicles, and sensors. On the other hand, the server mode implements the client-server architecture seen in Figure 2.1 which essentially allows the simulation world to be run by a CARLA server and controlled by a client application. This architecture allows for multiple clients to be created in the same or different nodes depending on how communication is set up through transmission control ports (TCP) [67]. In doing so, CARLA serves as an interface through which the simulated world and the agent can interact.

The modules that make up the client-side control the interaction between the AD agents and the server by managing the logical aspect of the agents as it drives through different routes and scenarios. This task includes spawning actors, getting the current state of the world, and changing the environment weather. It does this by sending commands to the server that dictate the steering, braking, and accelerating actions that it should take. It also sends meta-commands to the server, which controls the sensor suite utilized by the ego-vehicle, environmental aspects of the simulation such as the weather and illumination conditions, and the population density of non-player characters (NPC) to spawn, which includes vehicles and pedestrians. In conjunction with the client, the server-side is responsible for running the simulated world and rendering the pre-defined scenarios. In response to incoming commands from the client, the server sends data and meta-data concerning the sensors of the ego-vehicle. In this mode, the client application is essentially collecting data and sending instructions to the server running the simulation. An essential idea behind this architecture is that it adds a scalability factor to the simulation process, a valuable trait for research and development as multiple scenarios and agents may be tested simultaneously.

2.3.2 Town Environment Characteristics

CARLA simulations offer great flexibility through highly customizable driving environments and actors. In the context of CARLA, actors are considered to be anything that plays a role within the simulated world, including vehicles, pedestrians, sensors, traffic signs, and traffic lights, all of which have physical attributes and action states. When

running a CARLA server, certain aspects of the simulated world, such as the weather type and world-port, can be specified through environment variables. In contrast, other aspects, such as vehicle sensor configurations, defined routes, and scenarios, require defining Python, XML, and JSON scripts. It is also possible to build custom-made maps and vehicles. However, the process becomes more complex as it requires additional tools such as Unreal Engine Blueprint. A more straightforward way of working is to use the CARLA Blueprint library containing pre-made actor layouts in the form of Blueprints necessary for spawning actors.

Table 2.1: CARLA town descriptions.

Town ID	Description
Town01	A town with three-way junctions.
Town02	A town similar to Town01 but smaller.
Town03	Includes a five-lane junction, a roundabout, unevenness, and a tunnel.
Town04	Includes an infinity loop and a highway that connects to a small town.
Town05	A grid-like town with cross junctions, a bridge, and multi-lane streets.
Town06	Includes long highways with multiple entrances and exits.
Town07	A rural environment with narrower roads, barns, and few traffic lights.
Town10	Includes different city environments such as an avenue or promenade, and more realistic textures.

The simulation framework that this work focuses on consists of the eight urban-like towns described in Table 2.1. Each of these town environments is made up of digital assets, which are categorized into two groups: static objects such as vegetation, traffic signs, buildings, and other basic infrastructure, and dynamic objects such as pedestrians and other vehicles.

All assets are combined via a layered approach to build the eight CARLA towns. This layered approach consists of three parts: (1) designing the town roads and sidewalks, (2) placing static object assets down on the town terrain, and (3) defining the locations where the dynamic object assets will appear. Following a similar process, further maps can also be created by the user using the OpenDRIVE standard to describe the roads. Also, with a given map, environmental factors such as the time of day and weather conditions can vary to one of the 14 available kinds of weather options seen in Figure 2.2.



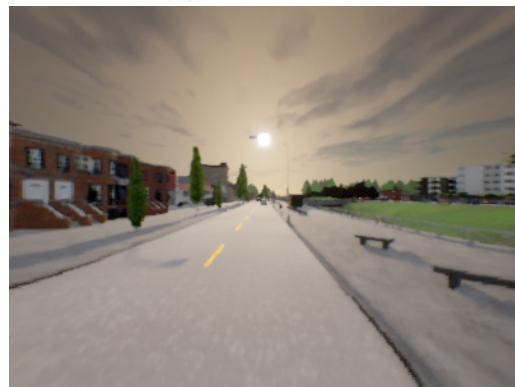
(a) Clear noon



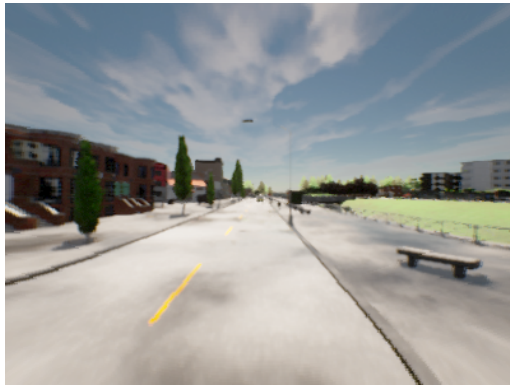
(b) Clear sunset



(c) Cloudy noon



(d) Cloudy sunset



(e) Wet noon



(f) Wet sunset



(g) Wet cloudy noon



(h) Wet cloudy sunset

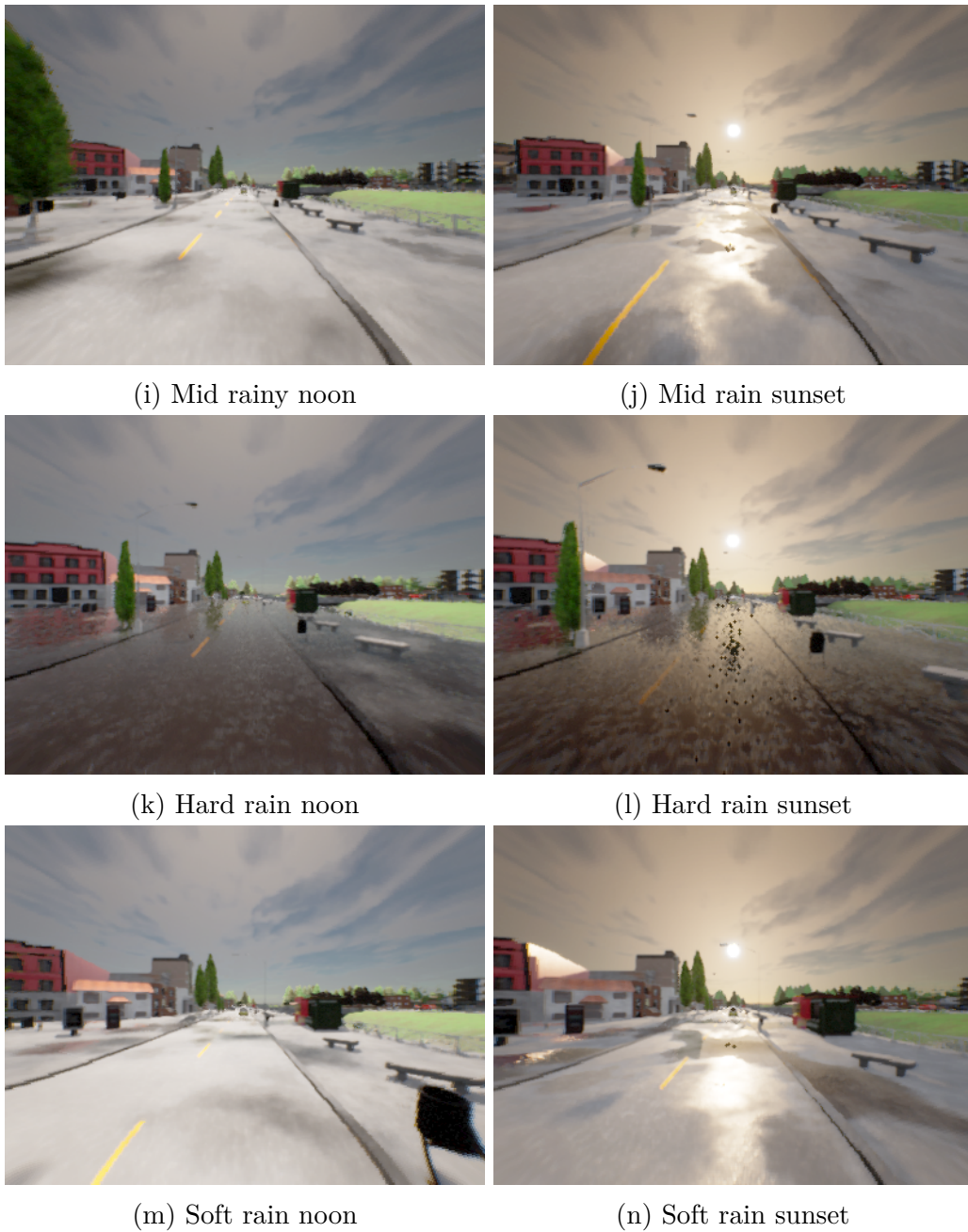


Figure 2.2: 14 weather variations in CARLA simulator.

2.3.3 Sensor Suite

As mentioned in the previous subsection, the definition of an actor in the context of CARLA extends to sensors as well. These sensors are connected to a parent vehicle and follow it throughout its drive, recollecting sensory data throughout the process. The number, type, and placement of each sensor on the vehicle, as well as the way they recollect data depend on how the sensors are configured within the Python scripts on the client-side of the

simulation. By default, the sensors which must continuously provide some output (such as RGB cameras or LiDAR) do so at every simulation step unless configured otherwise. Meanwhile, the sensors which do not meet this criterion obtain data each time a specific event occurs (such as collisions or lane invasions). The sensor suite provided by CARLA includes five camera types, three range sensors, three detector types, three sensors that help abstract the current state of the parent vehicle, and a sensor that provides integration with RSS (Responsible-Sensitive Safety). The complete suite of sensors is listed and described below:

- **RGB camera:** a regular camera that captures images from the scene. This sensor includes a set of post-process effects that help give a better sense of realism by allowing the intensity of light to affect the captured images, adding noise or blurs to the images, and simulating the reflection of bright objects, among other effects.
- **Depth camera:** a camera that provides raw data from a scene and creates a depth map by codifying the distance of each pixel in an image using the three RGB channels. These codified distances stored in RGB channels can then be transformed into a grayscale depth map with millimetric precision.
- **Semantic segmentation camera:** classifies every object in sight and displays it in a different color according to its tag. Every element spawned in the world is created with a unique tag depending on what type of object it is. This sensor provides access to 12 semantic classes, namely buildings, fences, pedestrians, piles, road lines, roads, sidewalks, vegetation, vehicles, walls, traffic signs, and another for any object that does not fall in any of these categories.
- **DVS camera:** also known as an event camera, it measures and captures changes of pixel intensities and brightness between consecutive frames, outputting a stream of asynchronous events instead of intensity frames like a typical camera does. In other words, the camera will not return an image where there are no pixel differences between consecutive frames. Because of this, they possess properties that give them the advantage of producing high-quality visual information with no motion blur even in challenging high-speed scenarios and high dynamic range environments.

- **Optical Flow camera:** measures and captures the motion flow perceived from the point of view of the camera, where every pixel recorded by the sensor encodes the velocity of that point projected to the image plane.
- **Radar sensor:** creates a 2D point map of the elements in the parent vehicles field of view and their relative speed regarding the sensor. Radar helps evaluate the movement and the direction of elements, determining whether the objects are static, moving towards or away from the parent vehicle.
- **LiDAR sensor:** a rotating LiDAR sensor that casts laser rays to compute a cloud of points that represent a static picture of the current scene. The output is a list of 4D LiDAR measurement points where each point includes information concerning their xyz coordinates and the intensity loss during the travel. This sensor also includes a drop-off attribute and a noise attribute to simulate the loss of cloud points due to external perturbations and unexpected deviations, giving the sensors a better sense of realism.
- **Semantic LiDAR sensor:** a rotating LiDAR sensor that casts laser rays to compute a cloud of points that represent a static picture of the current scene. It is similar to the regular LiDAR sensor with two exceptions: (1) this sensor does not include intensity, drop-off or noise attributes, (2) in addition to the coordinates of each cloud point, it also includes the cosine between the angle of incidence and the normal of the surface hit, and the index and semantic tag of the object hit by the ray.
- **Collision detector:** registers an event each time its parent actor collides against any kind of object (static or dynamic) in the simulated world. Several collisions may be detected during one simulation step. The registered information includes the type of object, an ID, and the collision coordinates.
- **Lane invasion detector:** registers an event each time the parent actor crosses a lane marking by considering the space between wheels and road data provided by the OpenDRIVE description of the map. The registered information consists of a list of crossed lane markings.

- **Obstacle detector:** registers an event each time the parent actor has an obstacle ahead by creating a capsular shape ahead of the parent actor that anticipates obstacles. This sensor helps check for collisions by ensuring that collisions with any kind of objects are detected.
- **GNSS sensor:** calculates and reports the current position of its parent actor in the simulated world through x , y , and z coordinates.
- **IMU sensor:** extracts the accelerometer, gyroscope and compass readings of the parent actors current state which measure the linear acceleration, angular velocity, and orientation in radians.
- **Speedometer:** a pseudo-sensor that approximates the linear velocity of the ego-vehicle.
- **RSS sensor:** RSS is a model that defines the notion of safe driving. CARLA provides integration with a C++ library for RSS to modify the trajectory of a vehicle using safety checks when necessary. It does this by using RSS sensors to calculate the RSS state of a parent vehicle and to output the current RSS response as sensor data.

2.3.4 Additional Key Features

Aside from the layout of the simulation environments, its actors, and the sensors that help these actors interact with the environment, other key features allow for more scenarios and AD approaches to be explored. One of those features is the *Traffic Manager* module that is built on the client-side of the simulator on top of the C++ API, as seen in Figure 2.1. This module aims to populate the urban environment with realistic traffic flows by controlling all NPC vehicles set to autopilot mode. The dynamic traffic flow of NPCs roaming throughout the simulation can be controlled through kinematic parameters which allow, force, or encourage some traffic behaviors such as sudden lane changes or cars that exceed the speed limit.

Another key feature is the *Scenario Runner* module that makes use of the CARLA API. A scenario is a complex choreography of actors that results in a specific traffic situation. Their purpose is to force the ego-vehicle being controlled to react to a specific situation.

Scenarios are generally divided into four categories: (1) vehicle control loss, (2) dealing with lane changes, (3) detecting obstacles in the road, and (4) handling intersections. Therefore, the *Scenario Runner* is the module used for defining scenarios for the ego-vehicle to drive in. One way of generating multiple scenarios in a single simulation is by defining a route file that uses waypoints to specify the path that the ego-vehicle agent must follow. The scenario events can then be activated once the ego-vehicle passes near defined trigger positions. This feature is important since defining traffic scenarios allows AV systems to be trained and evaluated under specific circumstances.

Although CARLA includes a wider range of features that, for the most part, go beyond the scope of this work, three features are worth mentioning as they are helpful for research and development. First, there is a *Recorder* feature, which logs snapshots of all simulation and actor states, allowing for a precise reenactment of all simulation events. In addition to this, a similar function allows for recording the sensory data of a driving episode. This feature helps recollect simulated driving data and visualizes the environment through the perspective of the ego-vehicle. Second, one can configure the elapsed simulation time between two simulation steps. This feature is known as *Time-step* and is measured in frames per second (fps). The minimum value is 10 fps, but it is recommended to use a higher time-step as this allows for more extended periods to be simulated in less time. Finally, there are various rendering options for running a simulation. Depending on the version of CARLA, either OpenGL or Vulkan graphic APIs can be used for rendering either high or low-quality graphics. Also, there is a no-rendering mode and an off-screen rendering mode, aside from the normal graphics-rendering mode. In the no-rendering mode, Unreal Engine skips all computations related to graphics. It returns empty data on graphics processing unit (GPU)-based sensors, preventing rendering overheads and facilitating traffic simulations and road behaviors at high frequencies. Meanwhile, in the off-screen rendering mode, all rendering and sensory data is computed as usual, except for there being no display available, making it useful when working on systems that do not count with GUI displays.

2.4 Dataset

For this work, we use the minimal version of an open-source dataset [3, 21] generated by an expert driving policy that made use of the 16 sensors seen in Table 2.2 to drive within the eight urban-like CARLA towns described in Table 2.1. The complete version of the dataset that results from running the expert agent through a series of routes and scenarios weighs approximately 432 GB and is divided into two datasets, a clear weather dataset (229 GB) that only contains clear noon weather (Figure 2.2a) and a 14 weathers dataset (203 GB) that contains data spanning all 14 preset weather conditions (Figure 2.2). In both cases, the recollected data is structured in the following manner:

TownX_{tiny, short, long}: corresponding to different towns and routes files

- **routesX_**: contains data for an individual route
 - **rgb_{front, left, right, rear}**: multi-view RGB camera images
 - **seg_{front, left, right, rear}**: corresponding segmentation images
 - **depth_{front, left, right, rear}**: corresponding depth images
 - **topdown**: segmentation images from a birds eye view (BEV) perspective
 - **2d bbs {front, left, right, rear}**: 2D bounding boxes for different agents in the corresponding camera view in a numpy array format
 - **3d bbs**: 3D bounding boxes for different agents in a numpy array format
 - **lidar**: 3D point clouds in a numpy array format
 - **affordances**: different types of affordances in a numpy array format
 - **measurements**: contains position, velocity and other metadata of the ego-vehicle in a JSON format

With the necessary elements defined, an expert driving policy generated the driving data of this dataset through simulation. This section provides descriptions and characteristics of the 14 weathers minimal dataset used for training and testing, including dataset size, structure, examples, and content. We also describe other aspects relevant to this

Table 2.2: Type and amount of sensors used for generating datasets.

Sensor Types	Complete Dataset	Minimal Dataset
RGB Camera	4	1
Semantic Segmentation Camera	4	0
Depth Camera	4	0
LiDAR Sensor	1	1
GNSS Sensor	1	1
IMU Sensor	1	1
Speedometer	1	1

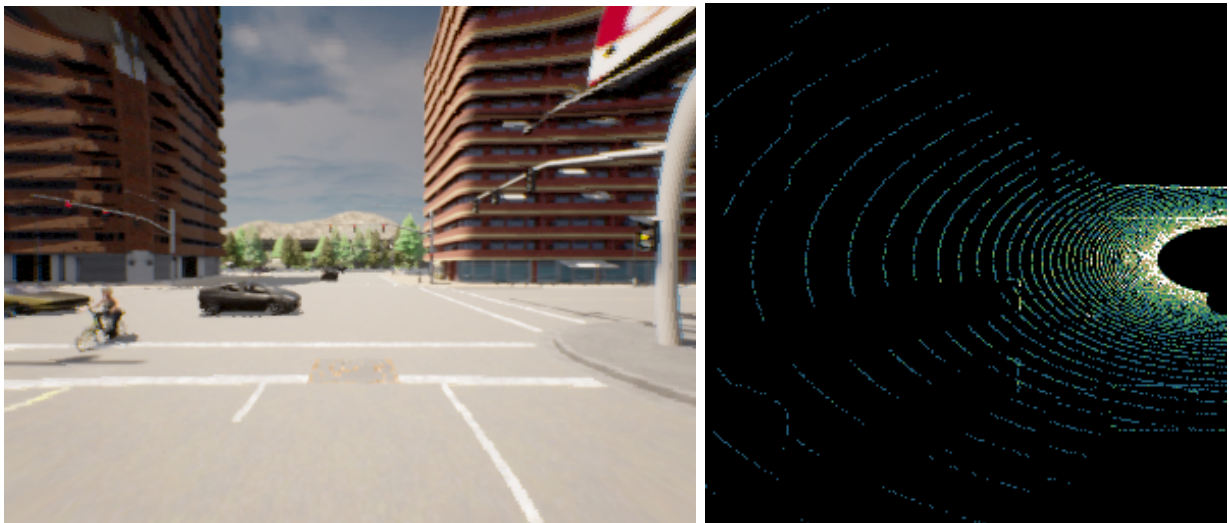
dataset, such as the configurations of the sensors on the ego-vehicle, the expert driving policy, the travel routes, and the driving scenarios.

2.4.1 14 Weathers Minimal Dataset

In addition to the complete large-scale dataset, the authors provide a minimal version of the 14 weathers dataset that weighs about 65 GB. This is the dataset utilized for training all the AD models presented in this work. It is essentially the same as the complete 14 weathers version, with the only difference being that it has been trimmed down to hold only the information needed to train the models.

Each of the eight towns used to recollect data contain between 100 and 200 dynamic agents and 14 weather conditions that sequentially varied for each frame recollected, as can be seen in Figure 2.2. In other words, the weather condition is varied after every 0.5 seconds (in-game time) to provide a uniform distribution of all weathers within the dataset. Also, data is generated at a rate of 2 FPS. More details on the amount of data on each town can be seen in Table 2.3. This data consists of three components: RGB images, LiDAR point cloud data, and vehicle driving measurements such as vehicle position, driving speed, steering angle, brake, and throttle values.

In total, this dataset contains about 144,000 driving frames, of which about 121,000 frames representing seven towns are reserved for training and 23,000 frames representing Town05 for validation. Figure 2.3 shows an example of the resulting RGB image (Figure 2.3a) and LiDAR BEV data (Figure 2.3b) that correspond to a single driving frame. In this example, the RGB image remains unprocessed while the 3D LiDAR cloud data (Figure 2.3b) is presented in BEV representation. The recollected data within the minimal dataset



(a) RGB image sample

(b) LiDAR point cloud

Figure 2.3: RGB and LiDAR data samples corresponding to one driving frame.

is structured in the following manner:

TownX {**tiny, short, long**}: corresponding to different towns and routes files

- **routesX**: contains data for an individual route
 - **rgb**: front-view RGB camera images
 - **lidar**: 3D point clouds in a numpy array format
 - **measurements**: contains position, velocity and other metadata of the ego-agent in a JSON format

Table 2.3: 14 weathers minimal dataset size per town.

Town ID	Size (GB)
Town01	8.6
Town02	6.1
Town03	11.5
Town04	12.4
Town05	10.8
Town06	8.7
Town07	3.7
Town10	3.4
Total	65.2

2.4.2 Sensor Configuration

In Table 2.2 we can see that the minimal dataset relies only on a GNSS sensor for localization, an IMU sensor for orientation, a speedometer for measuring speed (all of which do not require much configuration but are readily easily accessible through the Python API), a front-view RGB camera, a LiDAR sensor (both of which require configuration to use). The RGB camera has a field of view (FOV) of 100° and is mounted 2.3m from the ground level and 1.3m in front of the centroid of the ego-vehicle to not include the hood itself in the rendered images. Also, this camera is set to have a resolution of 400×300 pixels which is eventually processed to a resolution of 256×256 pixels for training. The LiDAR sensor is set to have an upper FOV of 10° and a lower FOV of -30° . It is mounted 0.2m above the RGB camera, has a range of 85m, and has a rotation frequency of 10 FPS.

2.4.3 Expert Policy

An expert policy is an auto-pilot agent that takes advantage of privileged information from the CARLA simulator, such as the global position of other actors, in order to avoid collisions, traffic violations and drive with exceptional overall performance. The primary purpose of this agent is to generate driving data that other models without privileged information can use for training. The expert policy in this framework consists of a path planner and two PID controllers, one for lateral control and one for longitudinal control. Given a route as a sequence of sparse waypoints, the path planner interpolates the route as a sequence of dense waypoints 1m apart. It then uses the nearest waypoints after 4m and 7m of its current position to calculate the aim direction of the ego-vehicle. This direction is input into the lateral PID controller, which outputs the necessary steering controls for reaching said waypoints.

The longitudinal PID control focuses on matching the speed of the ego-vehicle to a target speed from a discrete set of speed values, $\{0, 4, 7\}$ m/s, depending on the circumstance. If the ego-vehicle is approaching a stop sign, the target speed is set to 0 m/s. If the ego-vehicle is approaching a turn, the target speed is set to 4 m/s. Under a clear straight road, the target speed is set to 7 m/s. In addition to these heuristics, the performance of the expert policy also depends on three hyper-parameters that

tune each PID controller. For the later PID controller, these hyper-parameters are set to $K_p = 1.25, K_i = 0.75, K_d = 0.3$. For the longitudinal PID controller, these hyper-parameters are set to $K_p = 5.0, K_i = 0.5, K_d = 1.0$. In this manner, the expert policy is designed to drive cautiously, producing datasets that reflect safe driving behaviors.

2.4.4 Routes

The CARLA Leaderboard repository provides a set of 76 routes throughout six towns, each route being defined by a sequence of waypoints that the ego-vehicle should ideally follow to complete a route and be considered on-road. In order to incorporate more turnings and intersections into the training dataset, different routes spanning a total of eight towns are defined by sampling the existing routes. The resulting routes can be divided into three categories, long routes where each route represents 1000-2000m of road, short routes of 100-500m that represent three intersections, and tiny routes of 25-50m that represent single intersections. Data is generated for training and validation using only the short and tiny routes, using Town05 routes for generating the validation dataset, and all other town routes for generating the training dataset. The long routes defined for Town05 are left for the evaluation process since they have a greater driving diversity than the routes in the other towns. Also, only short and tiny routes are used for training since training models on datasets generated by long routes would cause the ego-vehicle to primarily learn to follow a straightforward path rather than the driving skills needed to handle more complex traffic scenarios seen at intersections.

2.4.5 Scenarios

Along with a set of routes, the CARLA Leaderboard repository also provides a set of three scenarios for each town. For this implementation, more scenarios are defined for training through the *Scenario Runner* feature by defining two things, trigger transforms that indicate the spawn location and orientation scenarios, and information on additional actors present in that scenario. Through this method, additional adversarial scenarios that focus on safety-critical situations are included in the dataset. Specifically, the seven types of scenarios depicted in Figure 2.4 are included in this dataset. They are defined by the

official CARLA Leaderboard platform [2] that considers the following 10 types of scenarios for evaluation,

Scenario 1: *Control loss without previous action* The ego-vehicle loses control due to bad road conditions and must recover, coming back to its original lane.

Scenario 2: *Longitudinal control after leading vehicle's brake* The leading vehicle decelerates suddenly due to an obstacle, and the ego-vehicle must perform an emergency brake or an avoidance maneuver.

Scenario 3: *Obstacle avoidance without prior action* The ego-vehicle encounters an obstacle/unexpected entity on the road and must perform an emergency brake or an avoidance maneuver.

Scenario 4: *Obstacle avoidance with prior action* The ego-vehicle finds an obstacle on the road while performing a maneuver and must perform an emergency brake or an avoidance maneuver.

Scenario 5: *Lane changing to evade slow leading vehicle* The ego-vehicle performs a lane changing to evade a leading vehicle, which is moving too slowly.

Scenario 6: *Vehicle passing dealing with oncoming traffic* The ego-vehicle must go around a blocking object using the opposite lane, yielding to oncoming traffic.

Scenario 7: *Crossing traffic running a red light at an intersection* The ego-vehicle is going straight at an intersection, but a crossing vehicle runs a red light, forcing the ego-vehicle to avoid the collision.

Scenario 8: *Unprotected left turn at intersection with oncoming traffic* The ego-vehicle is performing an unprotected left turn at an intersection, yielding to oncoming traffic.

Scenario 9: *Right turn at an intersection with crossing traffic* The ego-vehicle is performing a right turn at an intersection, yielding to crossing traffic.

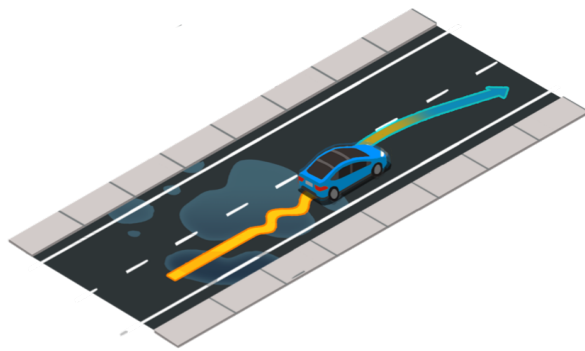
Scenario 10: *Crossing negotiation at a non-signalized intersection* The ego-vehicle needs to negotiate with other vehicles to cross a non-signalized intersection. In this situation, it is assumed that the first to enter the intersection has priority.

2.5 CARLA Leaderboard Benchmark

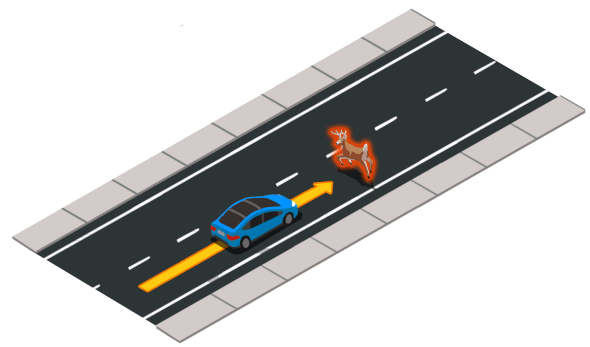
The Leaderboard benchmark [2] is the benchmark developed by CARLA for evaluating AD systems in numerous traffic scenarios through the CARLA simulator. Thus, it is an essential framework for obtaining driving agent performance metrics. The benchmark works by inputting an experiment suite that consists of a set of sensors, pre-defined routes, and driving environments through which an autonomous agent must drive, along with the files on the autonomous agent itself. The experiment suite used in this work is the one implemented by Prakash *et al.* [3, 21]. The Leaderboard benchmark then runs the CARLA simulation [20] using all of these elements to output a set of 12 performance metrics that quantify the overall performance of the evaluated agent through a series of scores and infraction rates along with an optional recording of the evaluation using the *Recorder* feature. This description is the general structure of the Leaderboard benchmark and is visualized in Figure 2.5.

In order to run the Leaderboard benchmark with the CARLA simulation, five key parameters must be set: the routes file containing all of the routes the agent is to follow, the scenario file containing all the trigger transforms that, when reached by ego-vehicle, triggers a specific scenario to occur, the number of times the ego-vehicle should traverse each route, the Python agent file and weights of the autonomous agent to be evaluated. With these conditions met, agents can be evaluated and compared under the same conditions. Table 2.4 shows examples of the benchmark results of the top-performing agents that were submitted to the SENSORS track of the CARLA AD Leaderboard platform in 2020.

Despite the CARLA framework including many towns and sensors, only a limited number of sensors are available for a select few evaluation towns. Given a superficial view of the



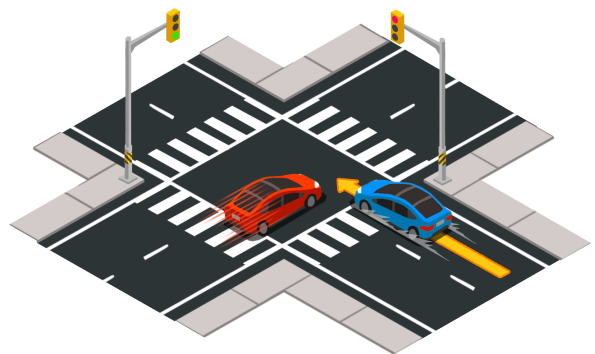
(a) Scenario 1: Control loss



(b) Scenario 3: Object crossing



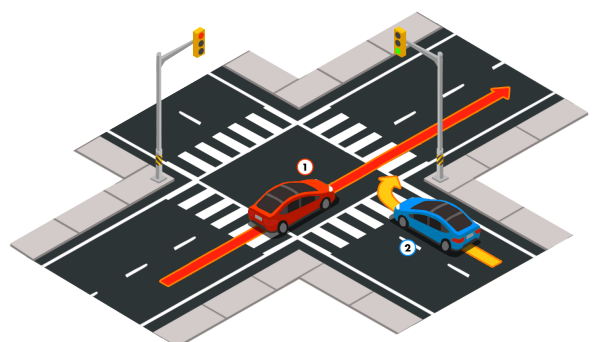
(c) Scenario 4: Object crossing during turn



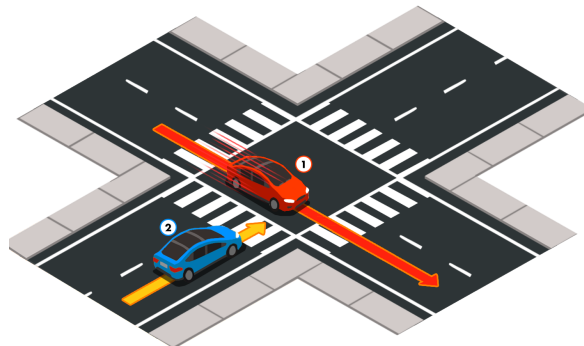
(d) Scenario 7: Vehicle running a red light



(e) Scenario 8: Unprotected left turn



(f) Scenario 9: Right turn with crossing traffic



(g) Scenario 10: Non-signalized intersection

Figure 2.4: Scenarios being simulated in the dataset. Source: [2].

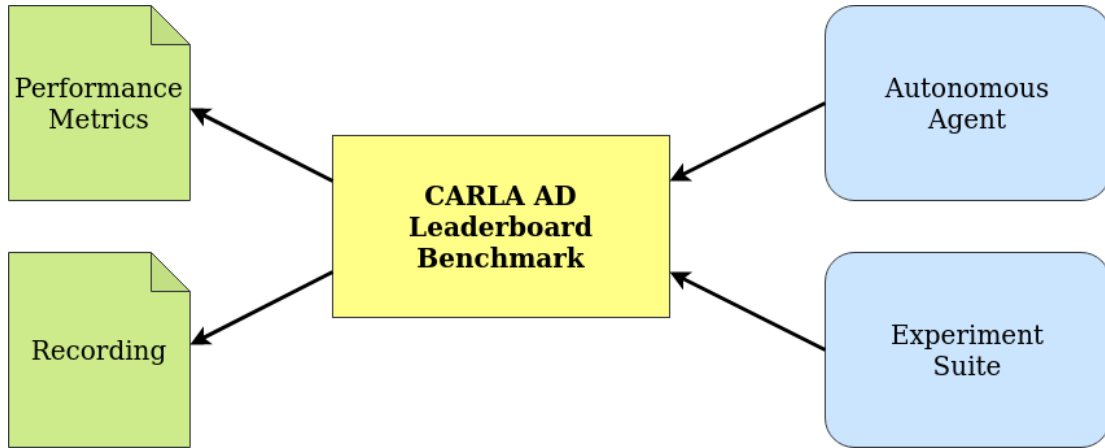


Figure 2.5: General structure of the CARLA Leaderboard benchmark.

Table 2.4: Winning teams of the SENSORS track in the CARLA AD Challenge 2020.

Submission	DS	RC	IP
MaRLn [54]	24.98	46.97	0.52
Anonymous	13.49	23.2	n/a
LBC [68]	8.94	17.54	0.73
Cadre v1	2.77	65.66	0.07

Leaderboard benchmark, the rest of this section further describes three important aspects of a driving benchmark. First, we describe the driving environment chosen for evaluation, the group of sensors available to the ego-vehicle, and finally, the resulting performance metrics.

2.5.1 Evaluation Environment

Each route consists of a starting point, a destination point, and a series of sparse waypoints through which the agent must travel through. However, the exact routes and conditions used by the official CARLA Leaderboard platform are maintained a secret for providing fair and consistent evaluations of all submitted autonomous agents. The only thing known is that multiple instances of 10 types of scenarios occur throughout routes spanning free-ways, urban scenes, and residential districts in multiple weather conditions. Although the official CARLA Leaderboard encourages research and development of level five AD systems capable of safely driving in any unseen condition, it does not allow for a deeper analysis of the evaluation methods or criteria used.

An alternative to the official CARLA Leaderboard benchmark is to use a local imple-

mentation of the benchmark based on the examples provided in their open github repository, and other similar works [69]. The implementation in study includes a set of 10 pre-defined long routes belonging to Town05, which are used strictly for benchmark evaluation (see Section 2.4.4 for more information on how routes are defined and categorized in this implementation). These long routes contain between 16 and 57 sparse waypoints and represent about 1000-2000m of simulated driving road. In addition, this implementation evaluates the ego-vehicle through scenarios that deal with control loss (scenario 01), object crossing (scenarios 03 and 04), signalized intersection negotiation (scenarios 07, 08, and 09), and non-signalized intersection negotiation (scenario 10). Examples of these scenarios can be better visualized in Figure 2.4.

As described in Table 2.1, the Town05 environment through which these routes and scenarios take place is a grid-like town that contains cross junctions, a bridge, and multi-lane streets, all of which can be seen in the examples provided in Figure 2.6. Since seven of the eight towns provided by CARLA are used for generating training data, Town05 is set aside as the environment in which the trained models have not had exposure during training. Furthermore, the environmental weather conditions for all benchmark evaluations are set to clear noon weather (Figure 2.2a).



(a) An agent turning at a cross junction

(b) An agent driving across a bridge

Figure 2.6: Examples of a trained autonomous agent being evaluated in Town05 routes.

2.5.2 Sensor Availability

According to the norms established by the Leaderboard benchmark [2], sensor availability depends on the type of track the autonomous agent is evaluated in. The *SENSORS* track

consists of the following: GNSS, IMU, LiDAR, radar, RGB camera, and speedometer. There is a second evaluation modality called the *MAP* track that consists of the same type of sensors with the addition of an HD BEV OpenDRIVE map. However, the models trained in this work only depend on the sensors included in the first modality excluding radar which is not utilized. Therefore, the sensors used in this implementation of the CARLA Leaderboard include LiDAR and RGB camera for perception, and GNSS, IMU, and speedometer for obtaining vehicle measurements. In other words, these are the sensors that the autonomous agent has access to and uses to input environment data and output vehicle controls.

During an evaluation, there are other sensors being utilized by the benchmark that help calculate the performance metrics of the driving agent. These sensors are the collision detector, the lane invasion detector, and the obstacle detector. Through these sensors, along with the ground-truth information that the benchmark has access to, the benchmark can keep track of all of the infractions committed by the agent as well as specify the exact location of the infractions along with any other actor that might have taken part in that infraction.

2.5.3 Score and Infraction Performance Metrics

The CARLA Leaderboard site [2] defines the benchmark outputs as 12 performance metrics that help understand different driving aspects of an autonomous agent being evaluated. These performance metrics are categorized into scores and infraction metrics. For each route that the autonomous agent is evaluated against, all of the infractions committed are tallied and used to calculate the score metrics. After the benchmark has gone through the entire set of routes defined in the experiment suite, a set of global performance metrics are calculated and output. For this work, the global driving scores are presented as averages of all individual route scores, and the global infraction metrics are presented as infraction rates per kilometer driven.

Driving Score The three performance metrics that summarize the driving performance of an autonomous agent through scores are the driving score (DS), the route completion score (RC), and the infraction penalty score (IP). Of these three scores, the DS is the

primary metric since it encompasses the other two scores in its calculation and is the main metric used for comparison with other autonomous agents. It is a percentage metric that begins as zero and ranges from $[0-100]$ whose value is calculated after obtaining the IP and RC of each route. In other words, a score of zero means that the agent did not complete any percentage of any of the routes while a score of 100 means that the agent completed every single route without committing any infractions. The DS of a single route and the global DS are calculated through the following two equations,

$$DS_i = RC_i \cdot IP_i, \quad (2.2)$$

$$DS = \frac{1}{n} \sum_{i=1}^n (DS_i) \quad (2.3)$$

where n = the total number of routes, RC_i = the RC of a single route, and IP_i = the IP of a single route. In order to calculate the remaining two scores, the remaining nine infraction metrics shown in Table 2.5 must first be introduced. These performance metrics can be further categorized into three types depending on how they affect the simulation and the score metrics.

Table 2.5: Infraction metrics accounted for during each route driven.

Type	Infraction	Penalty Coefficient	Description
1	Pedestrian Collision (PC)	0.50	The agent collides with a pedestrian.
	Vehicle Collision (VC)	0.60	The agent collides with a vehicle.
	Layout Collision (LC)	0.65	The agent collides with any other object.
	Red Light Violation (RL)	0.70	The agent runs a red light.
	Stop Sign Violation (SS)	0.80	The agent runs a stop sign.
2	Outside Route Lanes (OR)	-	If an agent drives off-road.
	Route Deviation (RD)	-	If an agent deviates more than 30m from the assigned route.
3	Agent Blocked (AB)	-	If an agent does not take any actions for 180 in-game seconds.
	Route Timeout (RT)	-	If the simulation of a route takes too long to finish.

Infraction Penalty Score Type one infractions consists of collisions and stopping violations. These infractions are organized from most severe to least severe and all count with a penalty coefficient that is applied to the IP every time that infraction occurs. With this in mind, the IP is a metric ranging from $(0, 1]$ that starts off as one and is decreased with every occurrence of a type one infraction. The penalty coefficient comes into play through the following equations that calculate the IP score of a single route and the global IP score,

$$IP_i = \prod_{j \in \mathfrak{F}} (p^j)^{v_i^j} \quad (2.4)$$

$$IP = \frac{1}{n} \sum_{i=1}^n (IP_i) \quad (2.5)$$

where n = the total number of routes, \mathfrak{F} represents the set of type one infractions, p^j = the penalty coefficient for infraction of type j , and v_i^j = the number of infractions of type j committed in route i .

Route Completion Score Type two infraction consists of a single infraction metric that keeps track of all the instances and distances that the agent went outside the route lanes it is supposed to follow. This metric has no penalty coefficient since rather than affecting the IP score, it directly affects the RC score. By keeping track of all of the distances that the agent correctly traveled and the distances that the agent goes off-road, the benchmark can calculate the percentage of the route traversed on-road and off-road. Thus, RC represents the route completion percentage of an agent before it either completes the route or is unable to continue. The RC score of a single route and the global RC score can be calculated through the following equations,

$$RC_i = D_{ON}(1 - D_{OFF}) \quad (2.6)$$

$$RC = \frac{1}{n} \sum_{i=1}^n (RC_i) \quad (2.7)$$

where n = the total number of routes, D_{ON} = the percentage of a route traveled on-road, and D_{OFF} = the percentage of a route traveled off-road.

Simulation Infractions Type three infractions consist of infractions that affect the simulation process rather than affecting any of the score metrics. These infractions are events that, when detected, prevent the agent from continuing their route and cause the simulation of that route to be shut down, thus, forcing the agent to move on to the next route. If any of these infractions take place, the status of the evaluated route is set as *failed*. Otherwise, the status is set as *completed*. In both cases, score metrics for that route are still computed and considered when calculating the global score metrics.

In addition to the three type three infractions listed in Table 2.5, there is one more event that is not listed because it is not related to the driving behavior of an agent. However, it does affect the simulation process. This event is known as the *Simulation timeout* and occurs if no client-server communication can be established in 60 seconds. Like the other type three infractions, this event also causes the benchmark to shut down the current simulated drive and move on to the next road. However, if this event takes place, no route will likely be evaluated since a lack of client-server communication can indicate a problem with how the client-server modules of the simulation were set up.

Chapter 3

State of the Art

This chapter describes the five IL models used for training and testing AD vehicles in this work. This description consists of the CILRS, AIM, late fusion, and geometric fusion baseline models, as well as the state-of-the-art TransFuser model proposed by Prakash *et al.* [3, 21] which is the main model of focus for this work. All of the information that follows is based mainly on this original TransFuser research which includes code repositories, model configurations, and supplementary details, unless otherwise specified. In addition to the model architectures, this chapter introduces the notion of self-attention, which is a critical component of the TransFuser model, the PID controller used for navigation control, the model hyper-parameters, HPO, and HPC.

The models are presented chronologically since the later models can be viewed as an evolution of the previous models through improvements. In other words, some of the models share certain aspects which are re-utilized in later models, such as the input sensor modalities, the ResNet encoders for feature extraction, the multi-scale sensor fusion, the waypoint prediction network, and the PID controller. For this reason, it is necessary to understand the baseline models to understand the totality of the TransFuser model.

More specifically, the CILRS model contributes by introducing the use of ResNet encoders [70] that serve as the perception backbones of all models. The AIM model provides a GRU-based network for conditioning the model on sparse goal locations rather than navigational commands. The late fusion model introduces a LiDAR sensor modality in addition to the RGB image modality along with a novel sensor-fusion method. The geometric fusion model increases the complexity of the sensor-fusion process by adding four instances

of geometric feature projections throughout the feature extraction process. Finally, the TransFuser model replaces the geometric feature projection modules with transformers that use self-attention mechanisms for fusing the two sensor modalities.

3.1 Baseline Training Models

3.1.1 CILRS

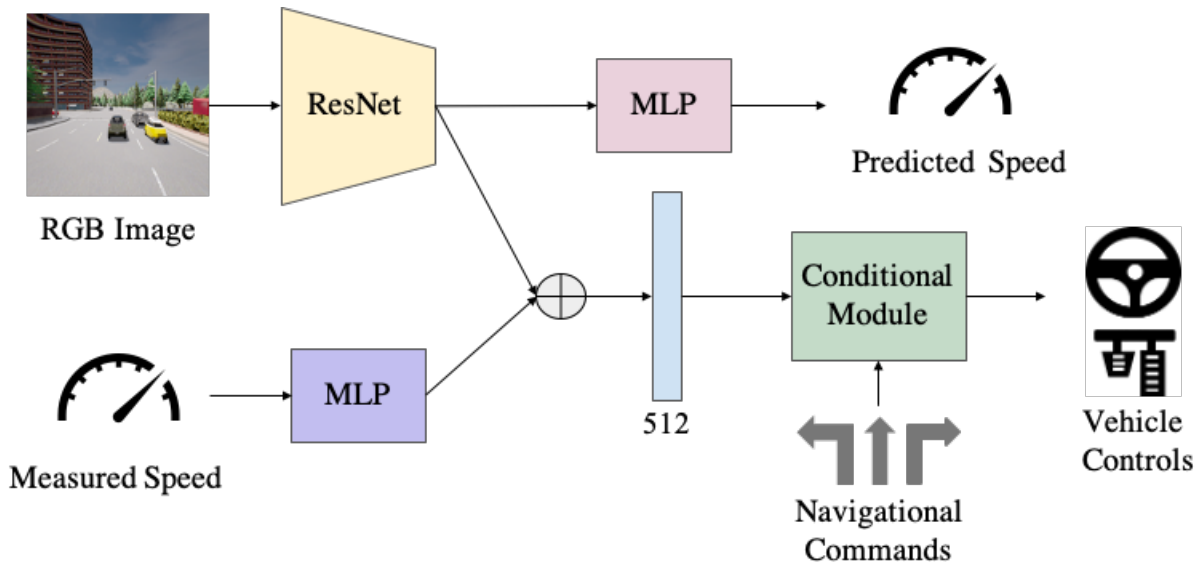


Figure 3.1: CILRS model architecture. Source: [3].

The CILRS (conditional IL ResNet) model seen in Figure 3.1 is an implementation of a CIL architecture [17]. This model takes as inputs images from a single RGB front-view camera, processes them in a ResNet-34 encoder, and outputs a 512-dimension feature vector. At the same time, the model also inputs measured speeds from a speedometer and processes them in an MLP with one hidden layer that outputs a feature vector with a dimension of 512, thus adding velocity information to the encoding. Afterward, the output of the ResNet module is passed to an MLP with two hidden layers that projects the image features to a predicted speed. The outputs of the MLP and ResNet module are also combined via element-wise summation and passed to a conditional module which is essentially a list of six MLPs with two hidden layers that represents the six conditional command branches that this CIL method uses to condition the model on the following six

discrete high-level navigational commands: follow lane, turn left, turn right, go straight at intersection, left lane change, and right lane change.

The conditional module works by selecting one of six conditional command branches depending on the input command and directly outputting the control values for steer, throttle, and brake in a prediction variable tensor. Furthermore, all hidden layers of the MLPs used in this model are of size 256. The CILRS model uses only RGB images and measured speed to predict vehicle controls while being conditioned on navigational commands.

This model has outperformed previous state-of-the-art mediated perception models such as CAL [71], MT [72], and CIRL [73] in empty, regular, and dense traffic conditions [17]. Unlike these other models, CILRS does not need additional supervision through extra information such as affordances or segmented images. This is mainly due to the use of deep residual networks and extra focus on speed prediction. Instead, it only requires a large-scale demonstration dataset.

Nevertheless, this model still has limitations that prohibit it from handling the driving complexities of urban environments. One of the problems that CILRS presents is performance degradation when in the presence of dynamic objects, showing limited generalization capabilities as the model fails to capture the complexities of environments with increased multi-agent dynamics. This issue reflects one of the things that subsequent models described in this section aim to improve.

3.1.2 AIM

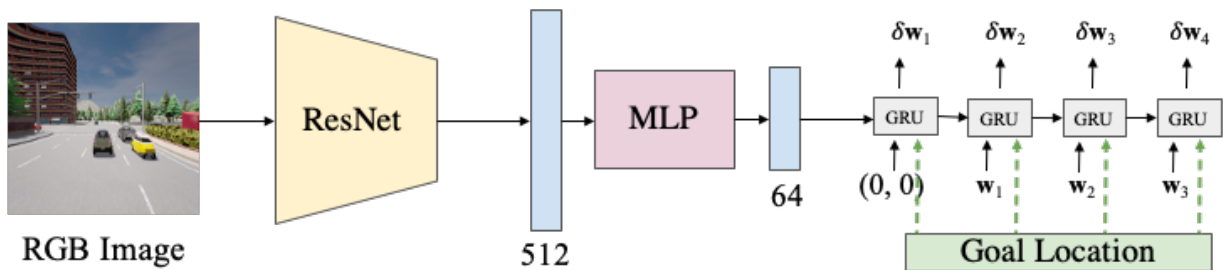


Figure 3.2: AIM model architecture. Source: [3].

The AIM (auto-regressive image-based waypoint prediction) model seen in Figure 3.2 is an improvement upon the CILRS model, with the main differences being that AIM is

conditioned on sparse goal locations rather than navigational commands and, consequently, AIM predicts waypoints indicating the future path of the agent rather than navigational commands. Like CILRS, AIM also inputs images taken from an RGB camera, processes them in a ResNet-34 encoder, and outputs a 512-dimension feature vector. This feature vector is passed to an MLP with two hidden layers of size 256 and 128, reduced to a 64-dimension feature vector, and passed to the auto-regressive waypoint network implemented with four sequentially-aligned gated recurrent unit (GRU) cells.

The GRU-based auto-regressive waypoint is based on an RNN encoder-decoder model architecture that uses an input sequence to maximize the conditional probability of a target sequence applied to machine translation [74]. This method aims to learn a conditional distribution over a variable-length sequence conditioned on another variable-length sequence. The GRU network represents the decoder component in the AIM model that inputs a feature vector from the ResNet encoder and goal locations provided as GPS coordinates for waypoint prediction. The goal locations are input to the decoder since being in BEV space correlates the waypoints more with the GRU network than the image encoder that processes images from a front-view perspective. In other words, when applied to AD, the GRU network conditions the model on sparse goal locations to predict a sequence of four dense future 2D waypoints represented as $\{w_t = (x_t, y_t)\}_{t=1}^T$ where $T = 4$.

The four GRU cells represent a single GRU layer that consists of a hidden gate that is initialized in the first GRU cell by the input feature vector. It also consists of an update gate that controls the information flow encoded in the hidden gate by inputting the current position of the ego-vehicle along with the goal location as GPS coordinates and updating the encoded information so that the network focuses on the context relevant for predicting the next waypoint. Finally, a linear layer is used to input the predicted waypoint information encoded in a 64-dimensional feature vector and output the predicted waypoint coordinates through a linear transformation. This process is repeated four times throughout the GRU network for each GRU cell, inputting the previously predicted waypoint and outputting the next predicted waypoint. The following linear transformation is used to calculate future waypoints:

$$\{w_t = w_{t-1} + \delta w_t\}_{t=1}^T \quad (3.1)$$

where T = the future time-steps in the current coordinate frame of the ego-vehicle, w_t = the predicted waypoint, w_{t-1} = the previous predicted waypoint, and δw_t = the calculated change in coordinates. Keep in mind that $w_0 = (0, 0)$ since the ego-vehicle is always centered from a BEV space.

Experiments with AIM show an improvement of up to 22.82% on the driving score of this model when evaluated in long Town05 CARLA routes and compared to CILRS [3]. These results show that AIM is a stronger image-based baseline than CILRS. It also shows that representing predicted waypoints and goal locations in the same BEV space is better than representing them in different spaces, such as is the case with LBC (learning by cheating) [69], a previous state-of-the-art model that represents goal locations as heatmaps in a front-view perspective and BEV semantic maps for waypoint prediction. Furthermore, previous results of the AIM model suggest that processing goal locations at the near end of the network could help the learning of behaviors needed to follow goal locations and help the image encoder prioritize the higher-level semantics of a driving scene such as traffic light states and multi-agent dynamics over lower-level features.

Despite improvements, the GRU network applied in AIM is still prone to collisions and other road infractions penalized by the Leaderboard benchmark. One problem is that it is limited to generalizing the environment by only using 2D RGB images and goal locations. Thus, the lack of 3D context prevents it from better navigating through 3D urban environments. Still, this network is essential for all subsequent models since it is used for waypoint prediction by taking in a feature vector and goal locations. Therefore, for all subsequent models, only the modules of the feature extractors will be described since they only differ in the fusion techniques implemented in the encoder for processing the two streams of data.

3.1.3 Late Fusion

Studies have shown that AD models that incorporate computer vision-related tasks such as semantic segmentation and depth estimation into their models experience higher performances in sensorimotor tasks than models that are only image-based [47]. Therefore, to improve image-based AD models, many works have experimented with sensor fusion techniques to create multi-modal AD models. These models utilize information from different

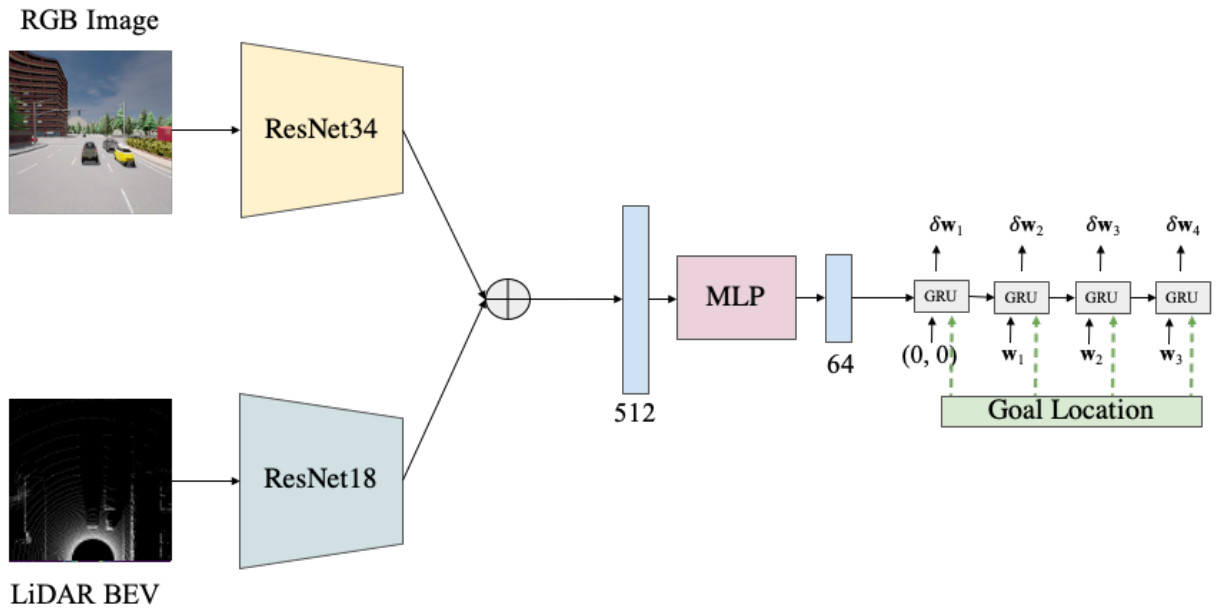


Figure 3.3: Late fusion model architecture. Source: [3].

sensors to complement one another, providing the resulting models with a greater global context of the environment. Prior studies have applied early, mid, and late fusion techniques to camera and depth modalities [75], shown the effectiveness of segmentation-based abstractions in models [76], and combined camera and LiDAR modalities in late fusion architectures. Thus, demonstrating that sensor fusion is essential for the deployment of AD systems in complex urban areas [52].

Mainly inspired by the latter of these examples, the late fusion model seen in Figure 3.3 is one of three multi-modal fusion architectures studied in this work that process both RGB images and LiDAR BEV images in independent streams of ResNet encoders. All three sensor-fusion models pre-process RGB images in the same way described in previous models. LiDAR point cloud data is also pre-processed into a two-bin histogram over a 2D grid with a resolution of 256×256 pixels. Here, the grid space represents a $32\text{m} \times 32\text{m}$ space corresponding to the cloud points 32m in front of the ego-vehicle and 16m to each side. The two-bin histogram results from dividing the height dimension of the LiDAR point cloud data into two bins. One bin contains the points above the sensor plane and the other on/below the sensor plane.

Like the AIM model, the image stream data is processed through a ResNet-34 encoder, while the LiDAR stream data is processed through a ResNet-18 encoder. Both streams

output a 512-dimensional feature vector representing the encoding of each stream individually. The late fusion comes into play here by combining both feature vectors at the end of the ResNet modules via an element-wise summation, thus, fusing the two sensor modalities into one. The resulting output is processed through an MLP and a GRU network decoder in the same way that it is processed in the AIM model to output a sequence of four waypoint coordinates.

Despite late fusion being a one-step fusion method, prior results show an improvement in performance with late fusion of up to 44.09% when compared to the DS of the CILRS model and up to 4.80% when compared to the DS of the AIM model [3]. This improvement suggests that even through an element-wise summation, the fusion of LiDAR cloud data and RGB image data provides the model with additional 3D context that it lacks in image-only models, helping it better navigate 3D urban environments. The late fusion method of this implementation was developed to serve as a baseline for the geometric fusion and TransFuser models. Evaluations of these models show that more profound fusion methods can be means for further improvements in sensor-fusion techniques to provide models with even better global contexts of their environments.

3.1.4 Geometric Fusion

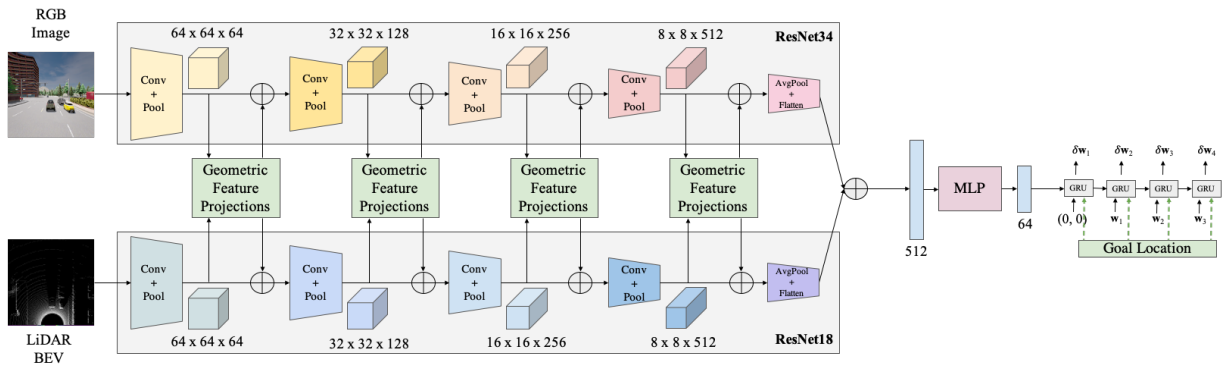


Figure 3.4: Geometric fusion model architecture. Source: [3].

Another sensor-fusion technique implemented in AD research is based on a geometric fusion approach. For example, studies have designed end-to-end architectures that exploit continuous convolutional blocks to model geometric relationships by fusing image and LiDAR data for the AD tasks of 3D object detection [38, 39].

The architecture that most closely resembles the TransFuser model is the ContFuse model [38]. This model architecture creates a multi-modal 3D object detector that uses continuous convolutional blocks to fuse camera image features with point cloud data features by projecting the camera into BEV space. Inspired by the ContFuse architecture, the geometric fusion model seen in Figure 3.4 is the second multi-modal fusion architecture that utilizes both RGB image and LiDAR data. Sensor fusion is done both during and after the encoding modules. Late fusion is performed after the feature extraction process via an element-wise summation of the 512-dimensional output feature vectors from both encoder streams, followed by an MLP block and a GRU network for waypoint prediction. Meanwhile, deep fusion is performed throughout the encoder streams.

The methods for pre-processing data are equivalent to those used in the late fusion model with the addition of dividing the $32\text{m} \times 32\text{m}$ LiDAR BEV grid into blocks of $0.125\text{m} \times 0.125\text{m}$ that are represented by one pixel. As with previous models, RGB and LiDAR BEV images are passed through a ResNet-34 encoder and a ResNet-18 encoder for extracting image and LiDAR features into 3D tensors of dimensions $H \times W \times C$ where H = height, W = width, and C = channels. Four convolutional blocks containing 2D convolutions and average pooling layers are first set up in both encoder streams. These convolutional blocks produce intermediate feature maps throughout the encoder of dimensions $64 \times 64 \times 64$, $32 \times 32 \times 128$, $16 \times 16 \times 256$, and $8 \times 8 \times 512$, which are then downsampled through pooling to produce feature maps of a fixed resolution of 8×8 , resulting in feature maps of dimensions $8 \times 8 \times 64$, $8 \times 8 \times 128$, $8 \times 8 \times 256$, and $8 \times 8 \times 512$. Then, 1×1 convolutions with a stride of one are used to match the embedding dimension of the feature maps to the set value of 512. With these dimensional configurations, the feature maps are ready to enter the geometric feature projection modules with a dimension of $8 \times 8 \times 512$.

After inputting the feature maps, the geometric feature projection module begins working in both directions, from image-to-LiDAR and LiDAR-to-image. For LiDAR-to-image projection, the model unprojects each $0.125\text{m} \times 0.125\text{m}$ space in LiDAR BEV into a 3D space, randomly selects five LiDAR cloud points, and projects them into the image space. The corresponding image features of these five points are added via element-wise summation and are passed to an MLP with three layers of 512 units each. This output is then combined with the LiDAR BEV features corresponding to the $0.125\text{m} \times 0.125\text{m}$ block.

Projection works similarly for image-to-LiDAR fusion. Five LiDAR cloud points that project to that pixel are randomly selected and projected into the LiDAR BEV space for each pixel in the image space. The corresponding LiDAR BEV features are added via element-wise summation and are passed to an MLP with three layers of 512 units each. This output is then combined with the image features corresponding to that pixel.

After LiDAR and image features are combined, the fusion module outputs a feature map of dimension $8 \times 8 \times 512$ that is upsampled with bilinear interpolation to match the original resolution and then processed through a 1×1 convolution to match the original embedding dimension of the feature map. With the $H \times W \times C$ dimensions of this output now matching those of the pre-downsampled feature maps, the output is fed back into the original modality branch from where it came from and is combined with the pre-downsampled feature map via element-wise summation. The geometric feature projection process described up until now constitutes that of one projection module. This process is repeated throughout the feature extractor at four different resolutions of 64×64 , 32×32 , 16×16 , and 8×8 , in that order. After completing the four stages of geometric feature projection, average pooling is performed on the feature maps of both streams, followed by a flattening function. The resulting feature vectors are ready to pass through late fusion, the MLP, and the GRU decoder network for waypoint prediction.

Results on geometric fusion show that the approach outperforms all the image-based baseline models and outperforms the late fusion approach by 2.76% on the DS of short routes [3]. It also has the highest RC score of all the models compared in that work, meaning that it correctly drove farther than all other models. However, these results also show that the infractions rates of geometric fusion are nearly equivalent to those of the previous models, indicating that the geometric fusion approach causes the model to prioritize navigating to a goal destination over avoiding obstacles that lead to infractions such as collisions. For this reason, the DS of this model tends to decrease when evaluating longer routes, causing this approach to obtain a DS 1.20% lower than the AIM approach and 6% lower than the late fusion approach when evaluated on long routes.

3.2 Attention and Transformer Architectures

RNNs or CNNs with encoder-decoder architectures have been the primary basis for most deep machine learning model systems. More recently, studies have demonstrated the advantages of attention mechanisms through a state-of-the-art machine learning architecture called the transformer. Transformers rely solely on attention mechanisms and offer an effective way of mapping input-output dependencies on par with other state-of-the-art models [4]. The concept of attention has been applied to natural language processing tasks such as reading comprehension [77] as well as computer vision-related tasks such classification [78], semantic segmentation [79], and multi-task end-to-end learning for AD models [80]. In all cases, transformers have managed to show improved results over other methods.

Attention Attention has been an influential concept in the area of deep learning introduced in 2015 [77] that allows for the modeling of spatial dependencies independently of their distance in input and output sequences. The key idea behind attention is to allow a decoder to pay attention to specific parts of a data sequence, relieving the encoder of the burden of encoding all of the input information while allowing it to focus more on the information relevant for generating the next sequence of data. For this, it maps inputs to tokens and outputs to the dimensions of the original input data.

3.2.1 Self-attention

Self-attention is an attention mechanism that introduces the notion of relating different positions of a single sequence in order to produce an output representation of the sequence [81]. Inspired by this idea, Google introduced the first transduction model, known as the transformer, that relies solely on self-attention for computing input and output attention token representations without the use of RNNs or CNNs [4]. In this work, an attention function is defined as the process of computing a set of queries, keys, and values (Q, K, and V) which are initialized randomly and mapped to a vector output. Also, the weight assigned to each value is calculated with a compatibility function of the query with the corresponding key. One method proposed by Vaswani *et al.* [4] implements a scaled-dot

product function as seen in Figure 3.5a and summarized in the following equation,

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.2}$$

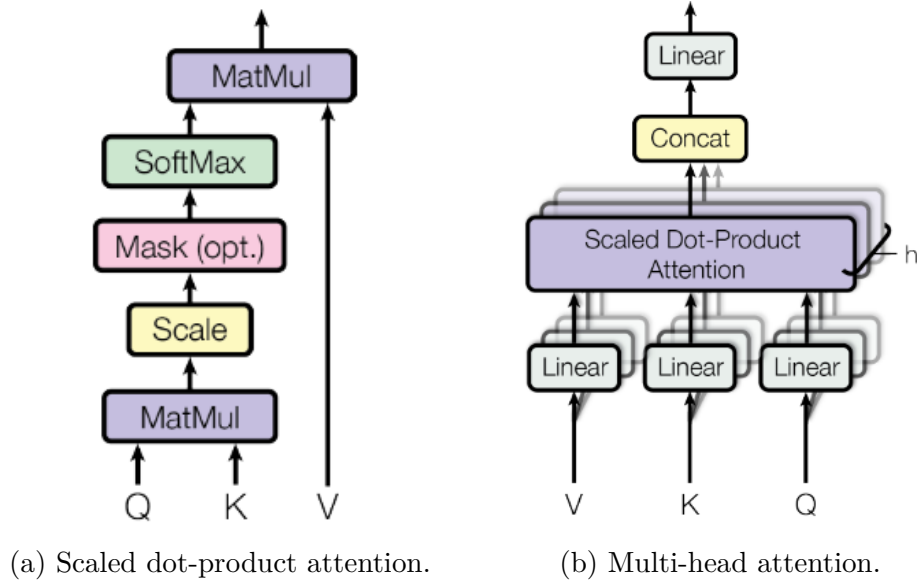


Figure 3.5: Attention function. Source: [4].

Meanwhile, Figure 3.5b depicts the notion of multi-head attention where h attention layers run in parallel computing scaled-dot products before being combined as a weighted sum of the values. This is the multi-head attention mechanism applied in the proposed transformer architecture seen in Figure 3.6 that follows an encoder-decoder style architecture.

GPT Another class of transformer architectures are known as language models. Examples of these models are generative pre-trained (GPT) models. These models apply generative pre-training on a language model using the transformer architecture seen in Figure 3.7 [5]. This multi-head approach defines 12 attention layers through which tokens are processed with self-attention mechanisms and normalization layers.

3.2.2 Attention in Autonomous Driving

Attention mechanisms are commonly applied in conjunction with RNNs in more complex transformer-based model architectures [3, 80, 79]. One application of self-attention in

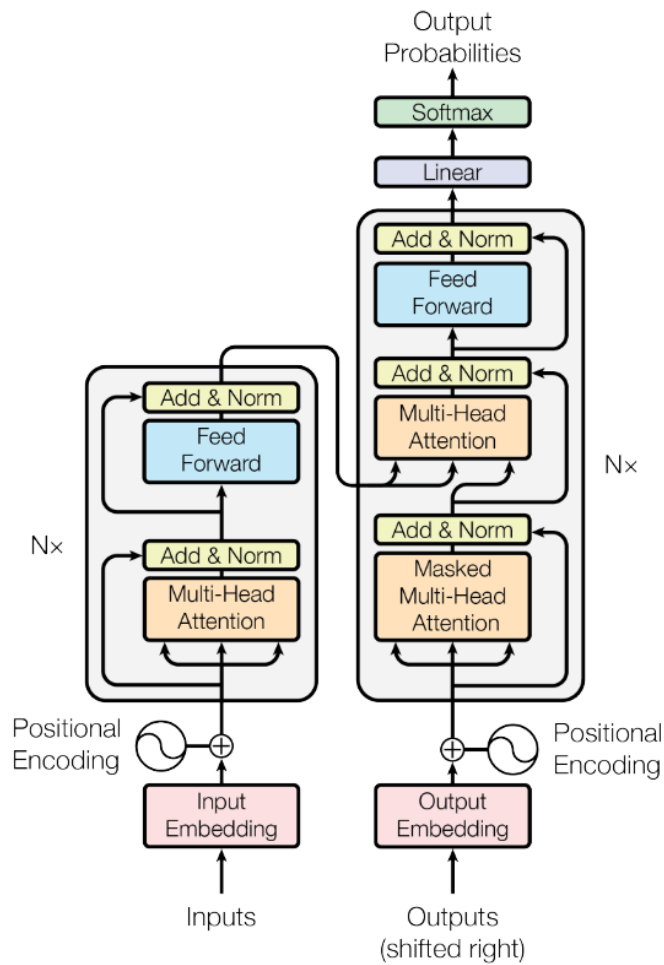


Figure 3.6: Transformer architecture. Source: [4].

computer vision is for the task of full AD [80]. In AD, transformers can also be used to combine data from RGB image and LiDAR sensor modalities. Figure 3.8 shows examples of the top-five attended tokens (green), queries (yellow), and the presence of vehicles in LiDAR images (red) through attention map visualizations. The visualized tokens are evidence of an AD model using self-attention to successfully map dependencies between traffic lights and passing vehicles at an intersection using data of different modalities.

3.3 TransFuser Model

The TransFuser model can be viewed as the equivalent of replacing the geometric feature projection modules from the geometric fusion model with transformer modules, as seen in Figure 3.9. The main difference lies within the four transformer modules of the feature

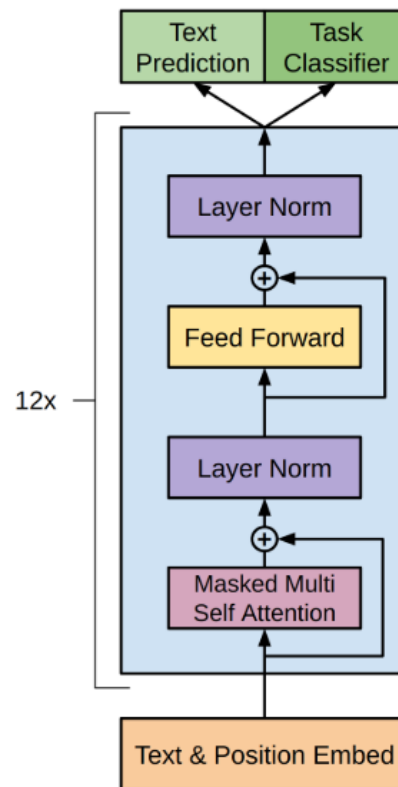


Figure 3.7: GPT model architecture. Source: [5].

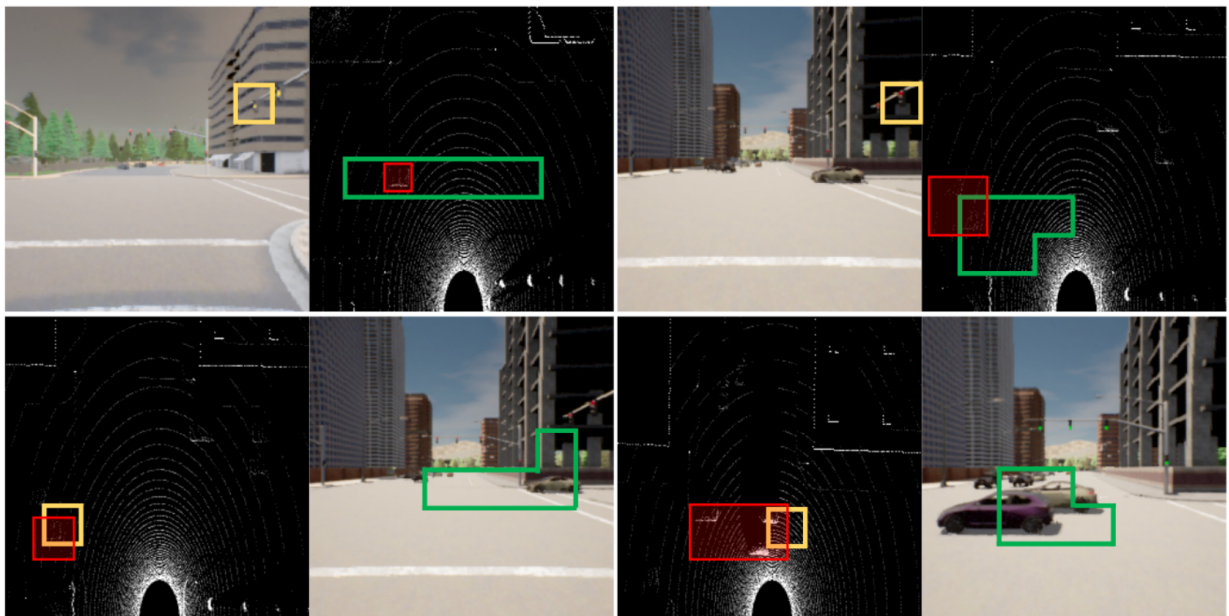


Figure 3.8: Attention maps in AD. Source: [3].

extractor that fuses RGB image and LiDAR features by using self-attention mechanisms. Through these modules, the model handles the issue of integrating the global environment

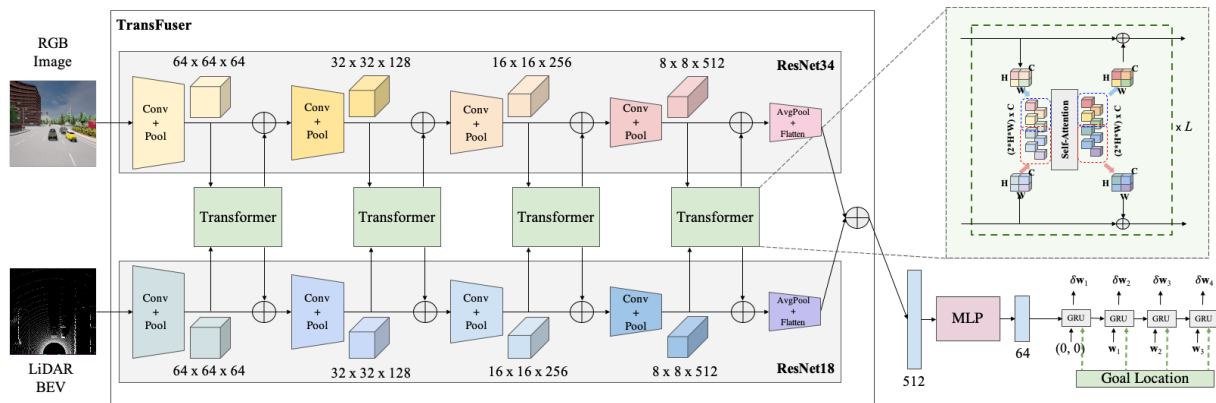


Figure 3.9: TransFuser model architecture. Source: [3].

context from complementary modalities at different resolutions.

In order to fully understand the scope of this model in the panorama of AD, Tables 2.4, 3.1 and 3.2 show examples of state-of-the-art AD models that have been submitted to the CARLA Leaderboard platform in 2020, 2021, and 2022, respectively. The rest of this section focuses on describing the transformer aspects essential for the functionality of this multi-modal model. This description includes the notion of self-attention, the transformer modules, and the ablation studies that have been done so far over the TransFuser system model.

3.3.1 Performances on the CARLA Leaderboard

The performance metrics shown on Tables 2.4, 3.1 and 3.2 are placed in order of DS ranking with respect to their submission year and are based on the information publically available on the CARLA Leaderboard platform [2] at the time of this writing. The submission of the original TransFuser model is found in sixth place in the 2021 submissions in Table 3.1 with a DS of 16.93. Here, two things can be appreciated about the TransFuser model. First, this model represents an improvement over all winning submissions of the previous year. Although the first place submission of 2021, MaRLn [54], obtained a higher DS of 24.98, their reinforcement learning approach required 20 days for training a model in a single CARLA town. In contrast, TransFuser required 20 hours for training a model on a dataset containing information on all CARLA towns. Second, despite the original model only reaching sixth place in the 2021 submissions, it has also been the inspiration for other TransFuser-based models that have managed to outperform the original submission. An

example of these alternative versions is TransFuser+ which applied the TransFuser model architecture with a different expert driving policy [82], reaching second place in the 2021 submissions with a DS of 34.58, twice as much as the original model. The authors that submitted the original model have also submitted two new TransFuser implementations in the 2022 submissions. Although, as of date, only three submissions constitute this list, the two new TransFuser implementations outperform all previous submissions with a DS of 42.36 and 50.57, as seen in Table 3.2. However, the changes in the methodology used in these new implementations remain unknown.

Table 3.1: Top public submissions to the CARLA AD Leaderboard platform in 2021.

Submission	DS	RC	IP
GRIAD	36.79	61.86	0.60
TransFuser+ [82]	34.58	69.84	0.56
World on Rails [83]	31.37	57.65	0.56
NEAT [84]	21.83	41.71	0.65
AIM-MT [84]	19.38	67.02	0.39
TransFuser [3]	16.93	51.82	0.42

Table 3.2: Top public submissions to the CARLA AD Leaderboard platform in 2022.

Submission	DS	RC	IP
LAV	61.85	94.46	0.64
TransFuser	50.57	73.84	0.68
Latent TransFuser	42.36	86.67	0.51

The publically available submissions in the CARLA AD Leaderboard platform are evidence of the growing popularity and research centered around the TransFuser model. In less than a year, the performance score of the model has seen a threefold increase, managing to surpass all but one model in the official Leaderboard ranking. This enhanced performance is likely due to the introduction of attention-based mechanisms for fusing sensor modalities in end-to-end AD models.

3.3.2 Transformer Module

As with the previous baseline models, TransFuser also considers a single RGB front-view camera and LiDAR BEV point cloud data as inputs to output a feature vector that passes through an MLP and a GRU decoder network, resulting in a sequence of four predicted

2D waypoints. It also uses a ResNet-34 encoder and a ResNet-18 encoder for encoding the image and LiDAR data, along with the convolutional blocks, to downsample the feature maps and match their embedding dimensions to the set value of 512. In this manner, data is fused four times throughout the ResNet encoders. After exiting each respective transformer module, the output feature maps are upsampled through bilinear interpolation and are processed to match the embedding dimension of the original feature map. This process is equivalent to the one described in more detail in the geometric fusion model.

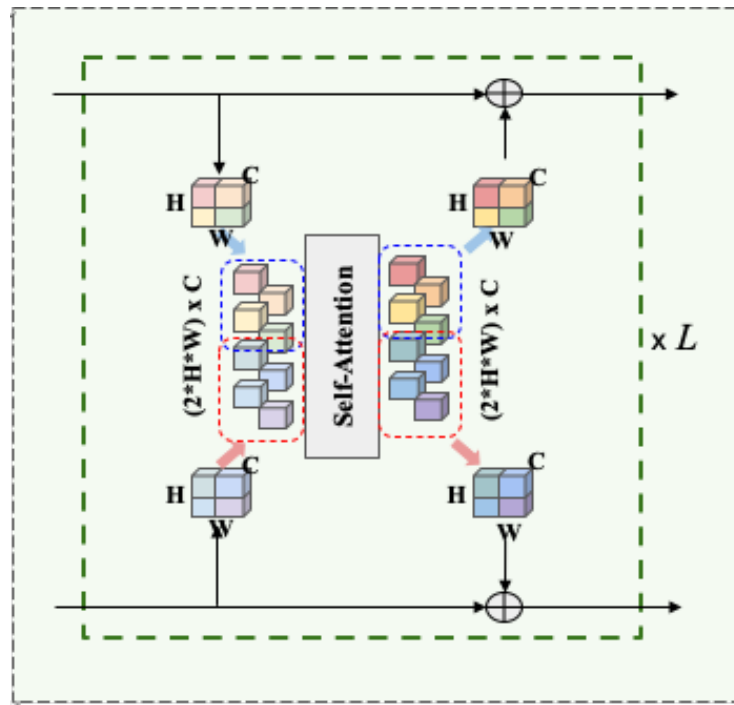


Figure 3.10: Transformer module architecture. Source: [3].

Figure 3.10 focuses on the architecture of a single transformer module that fuses sensor features. Here, the module first stacks the two feature maps of dimension $8 \times 8 \times 512$ into a single tensor of dimension $(2 \times 8 \times 8) \times 512$ representing an entire sequence of 64 image feature tokens and 64 LiDAR feature tokens. Seeing how these dimensions represent the original 256×256 image and LiDAR BEV images, each token corresponds to a 32×32 patch of its respective input modality. The dimension of the input token sequence can be denoted as $N \times D_f$ where N is the number of tokens in a sequence, and D_f is the dimension of the feature vector that represents each token.

In addition to these feature tokens, positional embedding and velocity embedding are also added via element-wise summation, although these inputs are not visualized in Figure

3.10. The positional embedding is a trainable parameter of the same dimension as the feature tokens that essentially provides the model with a sense of order about the position of the ego-vehicle in a driving scenario. This embedding allows the trained network to infer spatial dependencies between different tokens. Meanwhile, the velocity embedding added is a projection of the current velocity of the ego-vehicle through a linear layer into a 512-dimensional vector. Together, these elements make up the entirety of the tokens being input into the self-attention module.

Once the stack of tokens is passed through the self-attention module, a sequence of tokens of the same dimensions is then output. This sequence is then reshaped into two feature maps of dimension $8 \times 8 \times 512$, which are then upsampled and processed through 1×1 convolutions. Then, the feature maps are fed back into their ResNet streams via element-wise summation of the feature maps prior to entering the transformer module.

3.3.3 Self-Attention Module

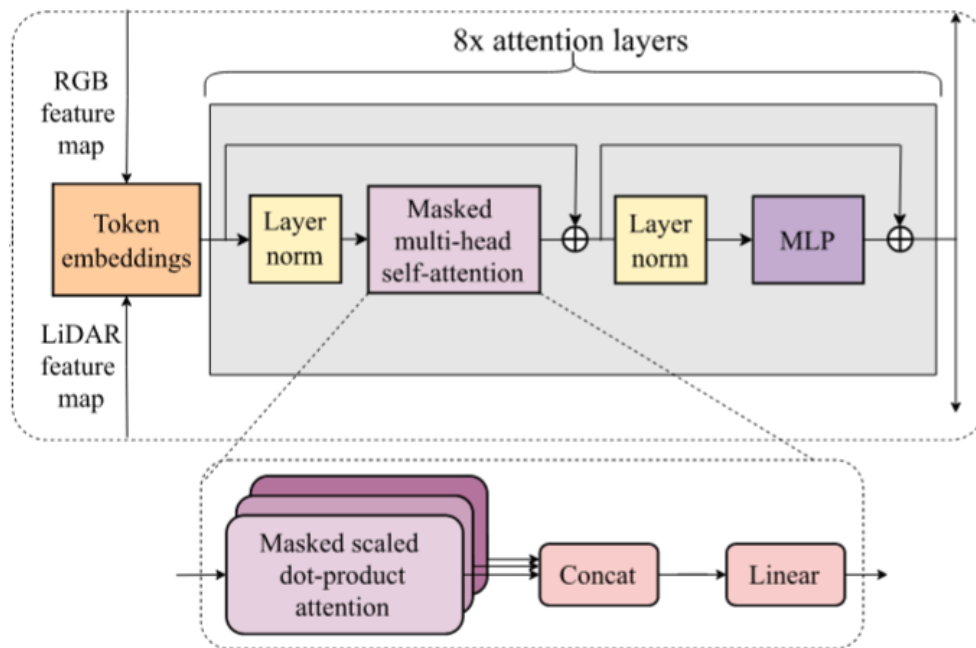


Figure 3.11: Self-attention mechanism of a TransFuser transformer.

The implementation of self-attention mechanisms in the TransFuser model is based on the GPT model. The TransFuser attention mechanism originally consists of eight attention layers and four attention heads, as seen in Figure 3.11. This multi-layered approach allows

for encoding different aspects of a scene in different attention layers by operating on intermediate grid-structured feature maps throughout the transformer blocks and including positional and velocity embeddings. Each multi-head attention layer generates four sets of Q , K and V values in parallel using Equation 3.2 to compute the attention weights and add the value for each query. These results are later concatenated with the initial output features with the following equation,

$$F^{out} = MLP(Attention(Q, K, V)) + F^{in} \quad (3.3)$$

Here, the calculated attention weights are passed through an MLP, a non-linear transformation, and combined via element-wise summation. This equation is applied for each attention layer in each transformer block.

3.3.4 Ablation Study

In most cases, it is not completely understood how NNs process information at the neuron level and how different parts of a NN affect the weights of the resulting trained models [85]. Since the weight values of a model are directly related to the performance capacity over a specific task, studying how different components of a NN affect the overall system performance becomes an indispensable part of tuning NN architectures. Ablation studies are techniques used for gaining insight into the contributions of specific NN system components by modifying them and evaluating the performance of the resulting system.

Current works provide ablation studies on seven TransFuser model components. Some of these components are involved in the feature extraction process and performing sensor fusion at multiple resolutions with multiple transformer blocks, multiple attention layers, positional embedding, and the ResNet encoders for image and LiDAR features [3]. Meanwhile, other components consist of the expert driving policy used for data generation and the controller used for outputting actions based on the predicted waypoints [82]. We summarize the results and conclusions derived from these studies as follows,

Sensor Fusion Scale Fusion was tested on scales ranging from one to four different resolutions. Results show an overall degradation in performance when reducing the number of scales.

Transformer Blocks A version was tested that used shared parameters for all transformer blocks and resulted in a significant drop in DS. This performance drop was expected since different convolutional layers in the ResNet encoders extract different features, meaning that multiple transformers should be used to fuse different types of features at each resolution.

Attention Layers Results were reported for two versions of the model, one for a model with one attention layer and another with four. Even though the one-layer variant obtained a higher RC, the DS was significantly lower than the four-layer variant. It was also mentioned that increasing the number of attention layers to eight leads to an increase in DS, indicating that multiple attention layers could cause the agent to drive more cautiously.

Positional Embedding When tested without including positional embedding into the feature tokens, although the RC increased by 25%, the DS dropped significantly. This performance drop is expected since modeling spatial dependencies is crucial for an agent to drive safely in an environment with multi-agent dynamics.

Sensor Encoders Different perception encoders were tested for the two sensor modalities in the late fusion model. Performance degradations were observed when applying ResNet-34 for Lidar, Inception v3 for image only, and Inception v3 for image and LiDAR. These results indicate that deeper perception backbone networks may hinder performance through overfitting.

Expert Policy This study implemented a new expert driving policy called SEED that mainly differs from the original implementation in the way that it handles traffic rules and collision avoidance. Although this change decreased the RC by 16.97%, it also increased the IP by 30% and the DS by 3.12%. This new policy showed a slight overall improvement but also caused the driving agent to unnecessarily stop on many occasions and get stuck.

PID Controller This study focused on making modifications to the original PID controller and fixing the problem of an agent getting stuck because of the SEED expert. This change significantly improved the DS by 10.75% and the RC by 28.86%.

3.4 PID Controller

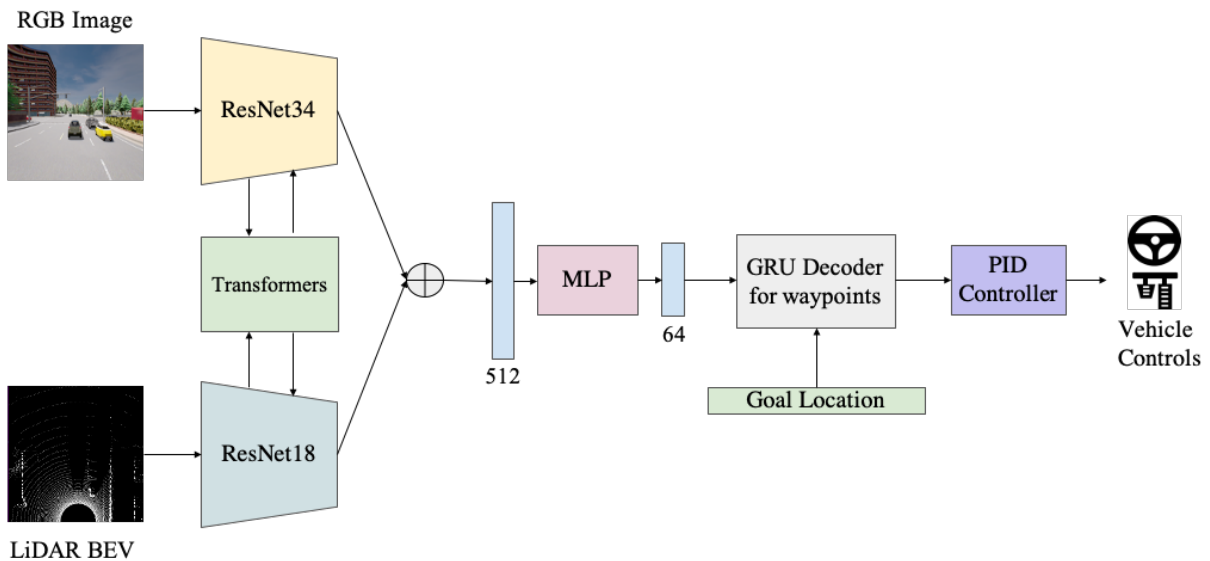


Figure 3.12: Complete end-to-end TransFuser AD model. Source: [3].

So far, five models have only partially been described in previous sections with a particular focus on the components relevant to perception. The only exception is the CILRS method which automatically outputs vehicle controls without the use of a GRU network. For full end-to-end automation in all other models, a control component is needed for translating predicted waypoints to vehicle actions. This component is the PID controller seen in at the end of the complete TransFuser architecture in Figure 3.12.

The PID controller is responsible for deriving vehicle controls from sensory inputs. This module applies to all described architectures that end with waypoint prediction. This section describes the PID controller used for performing all low-level vehicle controls based on the predicted waypoint inputs. For further information on the PID controller used for the expert driving policy, see Section 2.4.3.

The PID controllers implemented are based on an inverse dynamics model that provides joint torques and forces in terms of joint positions, velocities, and accelerations [86]. The control components for these AD models are composed of two controllers: a longitudinal controller and a lateral controller. In this manner, the PID controller can generate steer, throttle, and brake values based on input waypoints, controlling the motion of an ego-vehicle by applying the process indicated in Algorithm 1 to generate actions.

Algorithm 1: Generating actions from predicted waypoints

```

input :  $v, \{w_t\}_{t=1}^T$ 
output:  $\text{steer} \in [-1, 1], \text{throttle} \in [0, 1], \text{brake} \in \{0, 1\}$ 
1  $\gamma = 0;$ 
2 for  $t \leftarrow 1$  to  $T - 1$  do
3    $\gamma_+ = \lambda_t \|w_{t+1} - w_t\|;$ 
4    $\omega = \frac{w_1 + w_2}{2};$ 
5    $\alpha = \tan^{-1}\left(\frac{\omega[1]}{\omega[0]}\right);$ 
6    $\text{steer} = \text{LatPID}(\alpha);$ 
7   if  $\gamma \leq \beta_{min}$  or  $\gamma \leq v\beta_{ratio}$  then
8      $\text{throttle} = 0;$ 
9      $\text{brake} = 1;$ 
10  else
11     $\text{throttle} = \text{LonPID}(\gamma - v);$ 
12     $\text{brake} = 0;$ 

```

Keeping in mind that this implementation uses a time-step of $T = 4$, the first step is to compute the desired velocity γ by computing the weighted average of the $T - 1$ vectors between consecutive waypoints. Then, the longitudinal controller (LonPID in Algorithm 1) outputs a throttle value as it tries to match the vehicle velocity v to the desired velocity γ . For this, it uses weights $\lambda = \{1, 0, 0\}$ to prioritize the closest pair of waypoints, as this has shown to provide better empirical results. Finally, the lateral controller (LatPID in Algorithm 1) computes the aim direction α as the midpoint between w_1 and w_2 in order to output a steering angle that will orient the vehicle along that aim direction. Meanwhile, the brake value is set to one whenever the desired velocity γ is less than the brake threshold speed β_{min} or the brake speed ratio β_{ratio} which are set to 0.4 and 1.1, respectively.

Each controller also contains three gain parameters tuned alongside the two brake parameters. These parameters are set to $K_p = 1.25, K_i = 0.75, K_d = 0.3$ for the lateral controller and $K_p = 5.0, K_i = 0.5, K_d = 1.0$ for the longitudinal controller. Furthermore, both controllers also have a buffer size parameter that is set to 40. The buffer size is used to approximate the integral term as a running average.

3.5 Model Hyper-Parameters

In the previous sections, we have described the main components of end-to-end AD models. In addition, we have described three types of hyper-parameters involved with the training and evaluating of these models on the CARLA simulation framework. These are the hyper-parameters that model the system design of a transformer (see Section 3.2.1) and the three gain parameters for each PID controller (see Section 3.4).

In this section, we describe the principal hyper-parameters that have not yet been mentioned and are relevant to the optimization of the baseline and TransFuser models. The hyper-parameters considered here are the ones used to control the learning process and are set before training initiates, including the loss function, the optimizer, and other relevant hyper-parameters. Furthermore, we describe the concept of grid-searches as it is the method utilized in this work for carrying out HPO.

Loss Function Following other works [68], all models use an L1 loss function to train a network and learn a policy π and imitate an expert policy π^* . In the case of CILRS, the loss function is the weighted sum of an imitation loss and a velocity loss, as seen in the following equation,

$$\mathcal{L}_{CILRS} = \|\mathbf{a} - \hat{\mathbf{a}}\|_1 + \zeta \|\mathbf{v} - \hat{\mathbf{v}}\|_1 \quad (3.4)$$

where \mathbf{a} = predicted control, \mathbf{a}^* = ground-truth expert control, v = predicted speed, \hat{v} = actual vehicle speed, and $\zeta = 0.05$ for best empirical performance. The CIL optimization objective generalized in Equation 2.1 then makes use of this loss function.

For all other models that predict waypoints instead of direct vehicle controls, the loss function is the sum of the L1 distance between predicted waypoints, \mathbf{w}_t , and ground-truth expert waypoints, $\hat{\mathbf{w}}_t$, for $T = 4$ time-steps, as seen in the following equation,

$$\mathcal{L}_W = \sum_{t=1}^T \|\mathbf{w}_t - \hat{\mathbf{w}}_t\|_1 \quad (3.5)$$

In order to implement this loss function, assume a given expert driving policy π^* and a dataset $\mathcal{D} = \{(X^i, W^i)\}_{i=1}^Z$ of size Z . Here, X = high-dimensional observations of the environment, including image and LiDAR input from a single time-step, and $W = \{w_t = (x_t, y_t)\}_{t=1}^T$, a sequence of 2D waypoints in BEV space for $T = 4$ time-steps. Using this

dataset, and based on Equation 2.1, the optimization objective of the waypoint prediction models is generalized in the following equation,

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_i \mathcal{L}_{\mathcal{W}}(\pi(X), W) \quad (3.6)$$

Optimizer A significant amount of optimization problems are solved using stochastic gradient descent methods (SGD) or SGD with momentum. Another significant amount of problems are solved using adaptive gradient methods, such as Adam [87] or RMSprop, that scale the gradient entry according to a running average of its magnitude. This study focuses on the AdamW [88] optimizer since it is the adaptive gradient method implemented for training all previously described models with the weight decay coefficient set to 0.01, the learning rate set to 0.0001, and Adam beta values set to default values of 0.9 and 0.999 by PyTorch.

Like Adam, AdamW uses momentum and adaptive learning rates for faster convergence. However, Adam itself still has some disadvantages such as the method not always converging and a weight decay problem resulting from L_2 regularization not being as effective for Adam as it is for SGD. AdamW improves the regularization aspect of Adam by decoupling the weight decay from the gradient-based update, resulting in improved performance for computer vision tasks [88] as well as AD tasks [89]. This study also demonstrates that the decoupled weight decay eases HPO by rendering the optimal settings of the learning rate and the weight decay factor in a more independent way.

Other Main Hyper-Parameters In addition to the loss function, the optimizer, weight decay, and learning rate, there are two other hyper-parameters that are involved in the optimization process, namely the number of epochs and the batch size. In the original TransFuser study [3], all models are trained for up to 100 epochs since the best checkpoint would always be found within that range. Also, the batch size for each model is set to the maximum batch size that could fit on a single 1080 Ti GPU.

3.5.1 Hyper-Parameter Optimization

NNs use a wide range of hyper-parameters to define things such as system model architectures, regularization, and optimization. Recent interests in modeling computationally expensive machine learning methods have led to researchers focusing on ways of optimizing these models through their hyper-parameters [90]. Several methods can be used for HPO, one of those is the grid-search method. Considering a set of m hyper-parameters $\nu = (\nu_1, \nu_2, \dots, \nu_m)$, a simple way to set up a grid-search consists in defining a vector of lower bounds $a = (a_1, a_2, \dots, a_m)$ and a vector of upper bounds $b = (b_1, b_2, \dots, b_m)$ for each component of ν . Grid-search involves taking n points in each interval of the form $[a_i, b_i]$ including a_i and b_i . This creates a total of n^m possible grid points to check. Finally, once each pair of points is calculated, the ones that lead to maximum performance are chosen as the values for model optimization. The problem with this type of method is that the number of evaluations increases exponentially as n and m increase. Since we cannot really reduce m , decreasing n is the only possible way of assuring that the method stops in a reasonable time, but this decreases the validity of the solution [91].

3.6 High Performance Computing

HPC is defined as a multidisciplinary field that combines hardware, architecture, operating systems, programming tools, software, and algorithms to solve end-user problems [92]. HPC is also synonymous with supercomputing since the problems that require HPC to solve tend to be too computationally expensive to solve on a common general-purpose PC system.

An application is the combination of a problem that needs to be solved and the algorithmic methods that are the means of solving that method. In this context, depending on the nature of a problem and the proposed solution, an application may benefit from being computed on a system with greater computing capabilities. By applying parallelization techniques with HPC systems designed to optimize large-scale computing efficiency, one can take advantage of said systems to go beyond the peak computing performance that common PCs can reach, making certain applications more feasible to implement on an HPC system with faster computations and increased overall throughput.

The field of research that has dominated HPC usage is heavy-duty plasma simulations. Interest in machine learning applications has also increased mainly due to deep learning models that could benefit from using the accelerated hardware of HPC infrastructures, such as GPUs, which allows for scaling up application performance [93]. In order to measure computational performance and efficiency of HPC systems for solving any given problem, performance metrics can be obtained through benchmarks and system monitors. These are vital for HPC since the metrics obtained allow for a comparison of how a computational problem can be solved on one HPC system versus another.

One type of HPC machine learning application is one designed for the task of AD. This combines aspects of computer vision, simulations, and deep learning into one application. In this section we describe HPC clusters, performance metrics, and the fusion between HPC, simulations, and deep learning in the context of AD.

3.6.1 Clusters

There are four main classes of supercomputing systems that are used for handling HPC applications to solve computationally expensive problems, these are supercomputers, clusters, cloud-computing, and grid-computing. Their components, the way they are structured and organized, along with the rules they follow are the main reasons why these systems are capable of accelerating the calculations of results. They are also the main ways in which each class of systems differ from one another.

Of these four classes of parallel computing the one we will describe is the cluster system as it is the most relevant for this work since, alongside supercomputers, clusters are among the most commonly used HPC systems for academic purposes. In fact, according to a late 2021 ranking by TOP500 [94] of the top 500 supercomputers in the world, 495 out of 500 of those use a cluster-based HPC system while 100% of these machines use some Linux-based operating system.

A computer cluster, also known as a Beowulf cluster, is a collection of regular computers with commodity hardware and specific software installed. If provided a proper network infrastructure, the computing devices of this collection, referred to as computing nodes, can communicate with one another to handle incoming jobs using their available resources. The available resources can include computing nodes, processing cores, inter-

connection, permanent storage, I/O options, and accelerators. These characteristics make clusters a scalable and accessible form of computation that allow users to get the greatest amount of computation at a lower cost. This is usually a cheaper alternative to the use of supercomputers due the relatively high cost of buying and maintaining them.

Clusters are commonly used by multiple users who interact with it by submitting jobs to a resource manager through a command-line interface. Resource managers are an inherent part of HPC systems that perform three principal functions: (1) resource allocation, (2) workload scheduling, and (3) support for executing and monitoring distributed workloads. Once a job is submitted, the resource manager takes the job to a queue where, if the job is accepted, it will be allocated access to resources and the job will be scheduled for running the specified application. Depending on the requirements specified by the user and the available computational resources, among a number of other affecting factors, a job may be allowed to continue down the computational pipeline where it will either continue executing until completion or be canceled prematurely by the user through resource manager-specific commands.

An example of an HPC cluster is the system owned by CEDIA, an Ecuadorian Network Corporation for Research and Education. Although currently it only consists of one computing node, it is set up with high-performance hardware capable of processing information more efficiently and will in fact soon obtain a second computing node with similar capabilities as the first. This cluster allows for the executing simulations that require intensive processing with large volumes of data for topics such as DNA analysis, artificial intelligence, chemical modeling, and climate analysis, among others. More practical details related to working on this cluster are given in Section 4.3.

3.6.2 Simulations and Deep Learning in HPC

The latest technological advances such as block chain, deep learning, artificial intelligence and HPC, have played a major role during the latest Covid-19 pandemic. These technological advances helped predict contaminated areas and patients, as well as predicted future tendencies from the spread of the disease. The use of HPC and machine learning have shown effectiveness in controlling the spread of COVID-19. Studies providing insights on the applications of machine learning and HPC have shown that HPC is advantageous

when processing large scale data of all citizens and machine learning techniques assist in the extraction of knowledge [95]. These techniques can be used to identify patients carrying the COVID-19 disease from those without it. It can divide citizens into categories based on their hygiene practices, living location, and areas visited. Using HPC the information compiled can be loaded, processed, and deliver immediate results to citizens.

A study done by Alfianato *et al.* uses HPC to solve complex parallel computing problems which students are not able to practice solving on general-use PC systems [96]. Results generated showed that tree problems could be solved directly using more than four AMD Ryzen 7 processors. Parallel computer lectures discuss the use of multiple processors simultaneously to generate results on highly complex problems. As mathematical problems become more complex to solve, increased production is necessary from the processor. Using clusters, students were able to find the root of the Gauss elimination method and backward method using a 100x100 matrices with a random number generated in under 3 seconds.

Another study written by Labib discusses the use of machine learning algorithms with a relatively small sample of a simulation to predict the optimal daylight and energy performance of buildings [97]. Machine learning is known to be precise and accurate, but computations are time-consuming processes. These processes must execute a set of simulations used as training for data validation. In some cases, there is only a small sample utilized, which produces inaccurate results. This work utilizes HPC to save crucial time to execute all simulations necessary for the machine learning. Using HPC they are able to execute thousands of tasks simultaneously and time-efficiently which increases the subset size of the simulation. To calculate the daylight performance, a NN model was designed from 506 simulations. This NN consisted of 404 training samples and 102 tests samples which validated the NN model. To develop the NN model, a small NN was first created with two intermediate layers consisting of 64 units each, and one layer with only one output unit. Results showed the used of the proposed NN model produced the best results using a point-on-time illuminance simulation with 130 epochs. The NN reduced the time necessary to examine 5000+ rooms configurations, from hours to minutes with an error margin of 0.94%. The model was applied to nine different room configurations to examine accuracy and predicted results.

Chapter 4

Methodology

In this work, we experiment with various configurations of a transformer-based sensor-fusion architecture to determine the optimal set of parameters to use for further development of transformer-based AV models. This procedure consists of four main components: (1) setting up an environment for training and testing AD models, (2) replicating results of previous TransFuser studies [3], (3) HPO of the TransFuser model through hyper-parameter sweeps, and (4) evaluating the performance of the resulting trained models. This chapter describes the methodology used in order to define and carry out the necessary experimentation process. This includes describing the problem, the proposed solutions and model designs, the experimental setup for establishing the computing environment for testing AVs, and how we go about implementing the methodology.

4.1 Phases of Problem Solving

This section describes the methods proposed for solving the problems stated in Section 1.2. For this, we revisit the stated problems and include a deeper analysis of the problems, deeming necessary the use of HPC. Furthermore, we describe the experimental design aimed at solving these problems. In essence, the proposed methodology revolves around training the baseline and TransFuser AD models and evaluating them using the CARLA Leaderboard benchmark. For this, we perform HPO sweeps by varying a chosen set of hyper-parameters that affect the proposed system model and the optimization process. We also propose the use of a visualization tool called WandB to monitor all training processes, sweeps, and the HPC system metrics.

4.1.1 Description of the Problem

The creation of full AVs capable of safely driving through any unseen environment where the only human inputs are the destination goals is an ongoing challenge for AD researchers. This challenge envelopes many sub-tasks related to AD as well as multiple solutions proposed over time. On top of these challenges, regions falling behind in AD research, such as those in South America, have a longer road for developing and implementing full AVs.

In end-to-end driving, vehicles can be equipped with sensors for perceiving the environment through multiple modalities. This, however, brings a new challenge of what is the best way to fuse these input modalities. This problem becomes more complicated when considering that ANNs are complex structures with many components that must usually go through ablation studies in order to understand how those components affect the overall performance of the ANN.

TransFuser is a state-of-the-art end-to-end AD model that proposes a novel way of fusing these modalities through self-attention mechanisms whose driving performance is on par with other state-of-the-art models. However, being a relatively new model, it has not been through sufficient investigations to understand the relationship between the different models components and the driving capabilities of the trained model. Even though some researchers have managed to duplicate the driving performance scores of the original model when tested against the official CARLA Leaderboard platform, the methods that were applied in order to achieve these performance gains are not of public knowledge. In other words, the full potential of the proposed TransFuser model has yet to be explored by HPO of specific model components in order to maximize driving performance.

One challenge that limits the extent to which most model components can be tuned is the computational complexity that usually accompanies the training of AD models. Firstly, a TransFuser model takes about 24 hours to fully train on an above-average GPU. However, due to the problem of high variance commonly suffered by IL models, it is important to keep in mind that in order for the resulting performance metrics of this model to be statistically accurate, at least three models should be trained from scratch, evaluated, and averaged. This means that acquiring accurate performance results of a single TransFuser model requires about 72 hours of training. Therefore, each variation of the model that

could be tested would require about 72 additional hours of training, representing a costly challenge for HPO.

In this work, training AVs entails the problems described thus far, including lack of studies on novel multi-modal fusion models and high computational costs. To design an HPO experiment to handle these tasks, it is first necessary to further analyze the problem and explore possible solutions.

4.1.2 Analysis of the Problem

After reviewing the only publically available investigations done with the TransFuser model, a total of seven model components can be related with performance gains, of which only five have to do with the central sensor-fusion process. These five relationships are that sensor fusion done at multiple resolution scales, with the use of multiple transformer blocks, each of which utilize multiple attention layers, all lead to performance gains when compared to their single-scaled implementations. The use of positional embedding also seems to improve performances along with the use of ResNet encoders as perception backbones over other encoders tested in previous works. More significant improvements can also be achieved outside of the sensor-fusion process by improving upon the expert driving policy that the model imitates and the PID controller that translates predicted waypoints into vehicle actions.

As far as these studies show, the biggest improvements in driving performances have been thanks to modifications done over the expert driving policy and the PID controller. Meanwhile the studies done over other model components mainly serve to give some justification to the current sensor-fusion architecture of the TransFuser model, leaving the relevance of many other components unknown. More specifically, there are two types of components that make up the TransFuser architecture. These are transformer components and non-transformer components, each of which consist of their own set of hyper-parameters that can be tuned. The non-transformer components consist of more well-known hyper-parameters such as the learning rate, number of training epochs, and batch size, while the transformer components consists of lesser-known hyper-parameters such as the number of attention layers, attention heads, and attention dropout.

In order to contribute to the existing research over the TransFuser model, one of the

goals of this work is to perform HPO over a select few of the TransFuser hyper-parameters, namely the number of attention layers, attention heads, the learning rate, and the batch size. Therefore, one way of exploring the impact of these hyper-parameters on the driving performance of the TransFuser model is to perform hyper-parameter sweeps over a fixed set of values. Also, HPC through clusters is a viable option for overcoming the computational limitation of the TransFuser model. By taking advantage of parallel computing techniques, a greater amount of models can be trained and evaluated in the same given time as opposed to a sequential approach on a typical desktop computer.

4.1.3 Experimental Design

In this work, we are going to test how four independent variables of the TransFuser model affect the overall training and driving performance of said model. These four variables are the attention layer, attention head, learning rate, and batch size hyper-parameters. Of these variables, the attention variables are more closely related to the inner-structure of the transformer blocks that fuse the data from different sensor modalities while the learning rate and batch size variables are more closely related to the rate of convergence during training. In other words, the dependant variables we are considering are the driving performance metrics and the HPC system metrics. Considering these variables, we hypothesize that by modifying these four hyper-parameter variables we can determine an optimal set of hyper-parameters that optimize the driving performance of the TransFuser model without sacrificing a significant amount of training time and computing resources.

Baseline Experiment First we train three instances of the CILRS, AIM, late fusion, and geometric fusion baseline models followed by the TransFuser model using the default configurations obtained from the TransFuser project repository [3] with the dataset described in Section 2.4, resulting in a total of 15 trained models that will be evaluated and averaged into five sets of performance metrics. These configurations consists of a default value of 0.0001 for the learning rate, 101 for the training epochs, and batch sizes of 256, 192, 128, 56 and 56 for the CILRS, AIM, late fusion, geometric fusion, and TransFuser models, respectively. This baseline experiment is so that we can obtain performance metrics of the original baseline and TransFuser implementations and compare our driving performances

to that of the original work taking into consideration that the only difference is that while the original work trained their models using only clear weather data, the models in this work are trained using data that spans 14 different weather conditions.

Afterwards, we begin treating our independent variables by defining two experiments that will be carried out sequentially and are aimed at determining the best HPO configurations through hyper-parameter sweeps. Each sweep is a grid-search that trains a single model for each possible combination of hyper-parameters taken from a fixed set of values that we define beforehand. Considering the high variance in performance results common in IL models, each tested configuration requires training three separate models from scratch, evaluating the performance of all three models, and average the resulting performance metrics for statistical accuracy. Therefore, both of the proposed experiments consist of three sweeps where the only things changing between the three models with identical hyper-parameters are the randomly set training seeds and the batch sampling order.

Experiment #1 The first experiment sweeps consists of a grid-search that trains nine transformer configurations of the original TransFuser model by sampling the number of attention heads and attention layers from the values $\{4, 8, 16\}$. Each transformer configuration is assigned an ID according to Table 4.1 where Config-4 represents the original transformer configuration. Once all three sweeps have been carried out, a total of 27 trained models will need to be evaluated and averaged into nine sets of performance metrics, one for each transformer configuration.

Table 4.1: IDs for the transformer configurations of the first sweep.

		Attention Heads		
		4	8	16
Attention Layers	4	Config-1	Config-2	Config-3
	8	Config-4	Config-5	Config-6
	16	Config-7	Config-8	Config-9

Experiment #2 Based off the performance results from the models of the first experiment sweeps, we will determine the average top three performing transformer configurations. The second experiment sweeps will then consist of three grid-searches, one for each

of the top transformer configurations. Each grid-search will train 18 TransFuser model configurations by sampling the batch sizes from $\{64, 32\}$ and by sampling the learning rates from $\{0.00001, 0.00005, 0.0001, 0.0003, 0.0005, 0.001, 0.005, 0.01, 0.05\}$.

Each of these model configurations is assigned an ID according to Table 4.2 where X represents a Config ID taken from Table 4.1 and where Config-X13 and Config-X23 pertain to the original learning rate values. For example, Config-123 refers to a model trained with four attention layers, four attention heads, a batch size of 32, and the original learning rate of 0.0001. As with the first experiment, the three experiment sweeps of this experiment will be carried out three times, resulting in a total of 162 trained models that will need to be evaluated and averaged into 54 sets of performance metrics, one for each unique TransFuser configuration.

Table 4.2: IDs for the TransFuser configurations of the second set of sweeps.

		Batch Sizes	
		64	32
Learning Rates	0.00001	Config-X11	Config-X21
	0.00005	Config-X12	Config-X22
	0.0001	Config-X13	Config-X23
	0.0003	Config-X14	Config-X24
	0.0005	Config-X15	Config-X25
	0.001	Config-X16	Config-X26
	0.005	Config-X17	Config-X27
	0.01	Config-X18	Config-X28
	0.05	Config-X19	Config-X29

Monitoring Sweeps Carrying out the sweeps described in experiments #1 and #2 requires training a total of 189 TransFuser models of which 63 models contain similar hyper-parameters. In order to simplify the processes of training and documenting we will implement a visualization tool called WandB that can connect directly from training scripts to an online platform for the monitoring of the training process. This tool allow us to implement each grid-search systematically. The online central dashboard allows for the visualization and management of HPO sweeps in real-time by monitoring training and system metrics. The training metrics consist of hyper-parameter values as well as other parameters including epochs, training losses and validation losses. The system metrics

consist of training times, training losses, validation losses, best validation epochs, and GPU usage.

4.2 Model Proposal

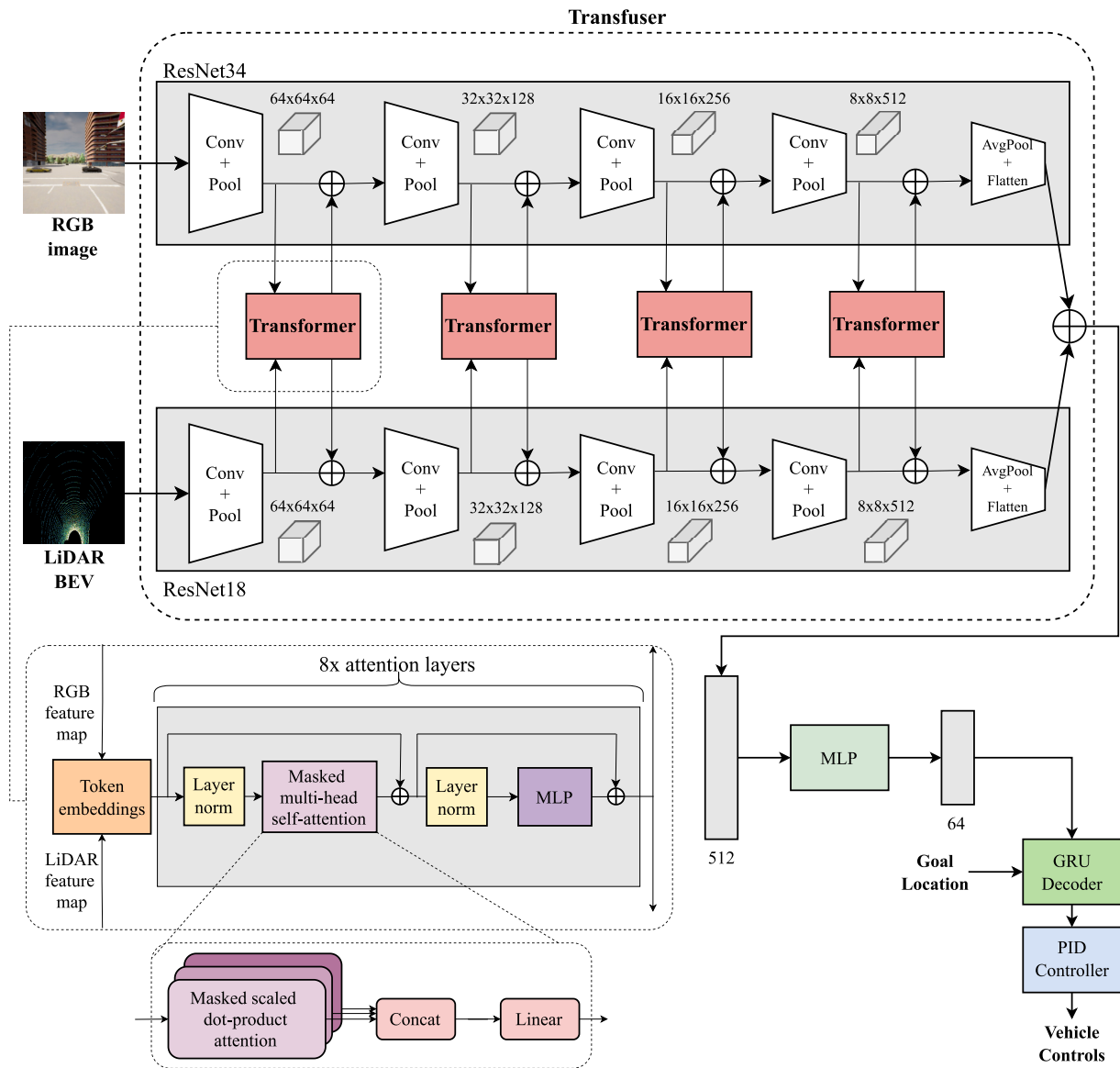


Figure 4.1: Transformer systems model of the TransFuser model.

The four hyper-parameters being tested in this work can be categorized by those that affect the TransFuser model design and the optimization process. In Figure 4.1 we can see the complete TransFuser model architecture with more emphasis on two of the components being modified, namely the number of attention layers and attention heads. Even though

only one transformer block is being dissected, we should keep in mind that the modification of attention hyper-parameters equally affect all transformer blocks.

4.2.1 Self-Attention System Design

First, we take a closer look at the inner-workings of the self-attention mechanism of the transformer module seen in Figure 3.11. Here we can see that a self-attention module is composed of multiple attention layers through which the input token embeddings are processed once combined. The tokens embeddings only depict the stacking of RGB and LiDAR token embeddings. However, these are also combined with positional and velocity embeddings via element-wise summation. Next, by further digging into a transformer block we can see that each attention layer contains a masked multi-head self-attention mechanism that calculates attention through Equation 3.2 and while the original GPT model architecture includes a normalization layer after the respective self-attention and MLP modules, the TransFuser model includes these layers beforehand.

Tuning the attention layer hyper-parameter creates system models with attention layers varying from the set $\{4, 8, 16\}$. The proposed model configurations also vary the number of attention heads that work in parallel throughout each attention layer from the set $\{4, 8, 16\}$. These modifications to the TransFuser model create model variations with the configurations stated in Table 4.1. In other words, the proposed transformer configurations include models with less, the same, and more attention layers as well as models with the same and more attention heads. For each proposed model configuration, the batch size and learning rate hyper-parameters are also tuned, affecting the number of RGB image and LiDAR BEV dataset samples processed before the model is updated and the step-size taken at each iteration of the optimization process.

4.3 Experimental Setup

For our work we use the CARLA AD simulation framework which includes a driving simulation (Section 2.3), an open-source dataset (Section 2.4), a benchmark (Section 2.5), and scripts for training and evaluating our models. In order to implement this framework on an HPC system, it was also necessary to set up a working environment on a local system

with a Linux operating system, compatible Nvidia drivers, and access to port 22 through which the secure-shell (SSH) protocol allows for an SSH client to connect with an SSH server. This communication enables remote login to an HPC system and command-line execution from a local system.

For initial testing purposes all of the software described in Table 4.3 was set up on both systems unless specified otherwise. However, since the objective of this work was to carry out the proposed experimental design on an HPC system (Section 4.1.3), this section mainly describes the setup process needed for training and evaluating models on an HPC system that works in conjunction with a local system. Still, the process for setting up the framework on an HPC system can also apply for setting it up on a local system.

Table 4.3: Software used for running the CARLA simulator.

Software	Version
Linux (Local)	Ubuntu 18.04.6 LTS
Linux (HPC)	Ubuntu 20.04.2 LTS
Nvidia Driver (Local)	v470
Nvidia Driver (HPC)	v450
Miniconda Environment	Python v3.7.11
Torch	v1.9.0+cu111
Torch Vision	v0.10.0+cu111
WandB	v0.12.9
CARLA Simulator	v0.9.10.1

4.3.1 Setting Up the Simulation Environment

CARLA supports the Ubuntu 18.04 and 20.04 Linux platforms, however, the one chosen for this work is Ubuntu 18.04 since it is the officially supported Linux platform for running CARLA. With a connection successfully established to an HPC system through a remote SSH login, a miniconda virtual environment can be setup with the Python version detailed in Table 4.3. Then, the pip package installer for Python can be used to install a list of package requirements. Of these packages, the most important are torch and torch vision libraries that provide the building blocks for our system models such as linear layers, convolutional layers, pooling, activation functions, and other architectural components. To avoid compatibility issues, we recommend installing the "+cu111" versions of these libraries that include support for CUDA tensors and operations on a CUDA-capable GPU

device. We also recommend installing the latest version of the WandB library which at the time of this writing is 0.12.9. This virtual environment must be activated before running any training or evaluation scripts.

With these requirements met, we proceed with setting up the CARLA simulator. Specifically, we use CARLA 0.9.10 for training and testing, however, newer versions of the simulator should also be compatible for this implementation. We download and install a packaged version of CARLA that weighs about 9.8 GB and includes all of the binaries, engine, maps, and API tools needed for working with the CARLA framework. Also, an optional package that includes an additional set of assets and maps is added and imported for the simulation to use, giving us access to all of the towns described in Table 2.1. More advanced customization and development options can be found through the Unreal Engine editor, however, for the purpose of this work, this editor is not built or used. In order to install all of these dependencies and use the CARLA simulator, our system must meet the following requirements:

- A Windows or Linux system.
- An adequate GPU with at least 6 GB for running the CARLA server.
- About 20 GB of disk space (30 GB with the Unreal Engine build).
- Python 3 for Windows and Python 2.7 or 3 for Linux.
- Pip package installer.
- Access to TCP ports 2000 and 2001.
- Pygame and numpy dependencies.

With the simulation environment set up, a CARLA server can be started with the OpenGL graphics API by running the following command on a terminal window,

```
./carla/CarlaUE4.sh -opengl
```


4.3.2 Parallel Computing Environment

The computing environment on which the CARLA simulation will be run is the CEDIA HPC cluster. This system consists of eight A100-SXM4-40GB GPUs with the hardware specifications described in Table 4.4.

Table 4.4: Hardware specifications of an A100-SXM4-40GB GPU.

Feature	Performance values
Double precision (FP64) peak performance	9.7 TFLOPS
Single precision (FP32) peak performance	19.5 TFLOPS
Half precision (FP16) peak performance	78 TFLOPS
Host-to-GPU transfer bandwidth	64 GB/s
GPU-to-GPU transfer bandwidth	600 GB/s
Memory bandwidth	1,555 GB/s
GPU base clock	1095 MHz
GPU boost clock	1410 MHz
Compute capability	8.0
Max wattage	400W

Each model that we train is trained on a single GPU at a time for consistency. Meanwhile, we run between 1-4 CARLA servers on different GPUs simultaneously when evaluating our models. To achieve this multi-client-server parallelization, we define two jobs for the SLURM workload manager, one for running a server and one for running a client, and define their environment variables through Bash script. The GPUs chosen for computation depends on their availability at the moment that they are needed.

In order to run multiple instances of the simulator server in parallel, we define four additional environment variables when running a server to enable the off-screen rendering mode, choose a GPU device for computation, and a world port through which the server will communicate to the CARLA client. This multi-client scheme requires running multiple servers on a GPU device with a command similar to the following but varying the GPU device from $\{0 - 7\}$ and the world port for each pair of client-server pair,

```
SDL_VIDEODRIVER=offscreen SDL_HINT_CUDA_DEVICE=7 ./carla/CarlaUE4.sh
--world-port=2300 -opengl
```

On the client-side, for each client to communicate with a corresponding server, the following environment variables must also be set to match,

```
export PORT=2300    # same as the carla server port
export TM_PORT=8300 # port for traffic manager
```

The traffic manager port defined in the server-side Bash script must be unique for each of the 15 client-server job pairs so that traffic and the simulation itself is managed independently for each evaluation. Also, jobs can be submitted to SLURM through Bash scripts that define all the necessary environment variables needed for evaluation. These Bash scripts are submitted with scheduler directives to run a server while logging the command-line output rather than displaying it.

4.3.3 Evaluation Environment

In order to evaluate the driving performance of a trained agent, the client-side environment must also be set up with additional environment variables and connected to a corresponding server. For this, we set up a Bash script indicating static and dynamic variables. Static variables remain the same through each evaluation and include paths used for running the simulation, the track modality, the number of times the vehicle should be evaluated through all routes, and because we want to evaluate all models on the same set of routes with the same set of scenarios we include the directory of the routes file and the scenario file, along with the correct port entries.

Meanwhile, the dynamic variables set paths to the specific agent Python script, the model to be evaluated, the name of the results file, and the path where images of each driving episode should be stored. Below is an example of a Bash script that defines both static and dynamic environment variables,

Client Bash Script Example

```
# static variables
export CHALLENGE_TRACK_CODENAME=SENSORS
export PORT=2300
export TM_PORT=8300
export REPETITIONS=3
export ROUTES=evaluation_routes/routes_town05_long.xml
export SCENARIOS=scenarios/town05_all_scenarios.json
```

```
# dynamic variables
export TEAM_AGENT=transfuser_agent.py
export TEAM_CONFIG=myModels/transfuser/myModel
export CHECKPOINT_ENDPOINT=results/myResults.json
export SAVE_PATH=data/mySavedEpisodes
```

The CARLA client-server architecture allows us to set up our evaluation environments with Bash scripts similar to the two previous server and client examples. In order to facilitate the running of multiple corresponding servers and clients for evaluating our trained models with the CARLA framework, we implement 15 pairs of client-server evaluation scripts. These scripts are used for the process of running multiple servers and logging the output of each terminal in an organized manner while setting up the desired evaluation environments in the CARLA simulator.

4.4 Implementation

The implementation of this work consists of four main tasks, initial testing, training, evaluating, and monitoring. In this section, we describe how the proposed methodology was carried out to accomplish these four tasks using a simulation framework running on an HPC system. This section is complemented by Section 4.2 that provides a more visual representation of the system model modifications proposed for this implementation and Section 4.1.3 that designs the process for experimentation with the chosen independent variables.

4.4.1 Testing

After setting up the experimental environment, but before HPO sweeps can be performed, initial testing of said experimental environments is required, both locally and on an HPC system. Training AVs is not a straight-forward task, and considering that some models take more than a day to train it becomes quite helpful to perform smaller-scaled training and evaluating sessions to ensure that everything is working properly and that we understand how to train and evaluate AV models on an HPC system while monitoring the process.

Initial training tests consisted of training the CILRS model for 20 epochs locally and

on an HPC system. Once that training was completed, each model would be evaluated on the machine that it was trained on and the performance metrics would be compared. After this, we eventually tested the WandB tool by modifying the training scrips to include and configure its parameters so that the training process can be monitored via a central dashboard.

4.4.2 Training Autonomous Vehicles

For our implementation, training an AV requires defining the training parameters, picking a GPU to train on, and sending a job to the HPC system to run a script that will train a specific type of model. GPU selection for training is based on their availability at that moment and on the task at hand. Since eight GPUs are available for multiple users on the CEDIA HPC system, their usage varies from time to time. Even though we cannot isolate the HPC resources from other users for our computing needs, we always picked the least occupied GPU for training.

Complementary to the training process itself is the monitoring of the training process. With WandB correctly integrated into the training scripts, running the model sweeps for HPO requires additional steps. First, we define the specifications to our model sweep in a yaml file. This file includes information regarding the training script, the type of sweep, the training metric, and the hyper-parameter values. For our implementation, we define a grid-search sweep and minimizing the validation loss as the main training goal. Then, we initialize the sweep with a WandB command and place the resulting sweep ID into a Bash script where, rather than running a job to directly train a model, we run a job to create a WandB agent that will begin performing the grid-search with the values defined in the yaml file.

4.4.3 Evaluating Autonomous Vehicles

From the perspective of the HPC resource manager, two jobs need to be sent in order for a model to be evaluated, one job for running a server and one job for running a client. In order to evaluate, with some level of parallelism, the models generated by hyper-parameter sweeps, Table 4.5 defines a fixed set of ports through which any server and client jobs can

communicate at any given time during an evaluation. We can now make use of this table to create 14 Bash scripts, each with their own set of ports for client-server communication and other environment variables. In this manner, multiple evaluations can take place by running one of 14 pairs of Bash scripts. This is to facilitate the process of evaluation since the ports will never cross and multiple evaluations can be run at any time independently of each other. This also limits the maximum amount of evaluations that can be simultaneously run to 14.

Table 4.5: Fixed ports used for running CARLA servers/clients in parallel.

Evaluation ID	World Port	Traffic Manager Port
0	2000	8000
1	2100	8100
2	2200	8200
3	2300	8300
4	2400	8400
5	2500	8500
6	2600	8600
7	2700	8700
8	2800	8800
9	2900	8900
10	3000	9000
11	3100	9100
12	3200	9200
13	3300	9300
14	3400	9400

Furthermore, we use the environment and scenarios described in detail in Section 2.5.1 for evaluating our trained models on the CARLA Leaderboard benchmark. This evaluation process is the same as the one used by Prakash *et al.* in the original TransFuser study [3]. Once all of the evaluation routes have been completed, the global set of performance metrics described in Section 2.5.3 are calculated and presented in a JSON file. Since each model configuration consists of three trained instances of that model, then the three sets of driving scores and infraction rates pertaining to each configuration will be averaged after the completion of each respective experiment. Statistical dispersion of these sets calculated through standard deviation will be added to the driving scores. Meanwhile, the infraction rates will be presented per kilometer driven.

Chapter 5

Results and Discussion

In this chapter, we present the results of implementing the methodology described in Chapter 4. For this, a total of 204 models are trained for carrying out the baseline experiment and the transformer and TransFuser HPO sweep experiments. After evaluating each model, we present tables detailing the performance metric values of the resulting evaluations. For analyzing performance, we consider the three driving scores (DS, RC, IP) and the type one infraction metrics associated with a penalty coefficient (PC, VC, LC, RL, SS). These benchmark results presented in tables are rates of infractions per kilometer driven and are placed in order of descending severity, from left to right, according to CARLA standards. Of the score metrics, the DS is the most important base for comparison between models since it reflects overall driving capabilities. With the RC metric already representing the percentage of driving completed, it is unnecessary to include the type two infraction that only represents a portion of the route driven off-road.

5.1 Baseline Experiment

After completing the baseline experiment, we analyze the DS and infraction metrics obtained from all evaluated baseline and TransFuser models as shown in Table 5.1 and in Table 5.2. In these tables, we highlight the best result of each respective column with boldface text. As expected, the CILRS model has the worst performance with respect to DS and RC. Even though it has the highest IP average of all the models tested, this value is ignored for all other comparisons since the main reason the CILRS agent commits less infraction penalties is that it only completes 13% to 21% of the total routes completed by

the other models. With less driven distances, there are less opportunities for infractions to be committed by the driving model. This analysis holds true for other evaluated models who obtain a low DS despite having an IP close to one. Therefore, even though the IP score is a representative of the overall infractions committed, it can be misleading if the RC score of two evaluations are relatively far apart.

Table 5.1: Driving scores of baseline models.

Model	Max DS	DS	RC	IP
CILRS	6.69	5.79±0.97	9.85±1.12	0.63±0.08
AIM	17.62	15.44±2.94	51.10±16.90	0.44±0.12
Late Fusion	28.78	21.49±8.64	48.15±27.16	0.64±0.21
Geometric Fusion	26.29	23.27±2.83	64.76±22.56	0.50±0.19
TransFuser	29.16	26.15±2.62	70.08±7.09	0.45±0.05

When looking at the average number of infractions committed on Table 5.2, we see that the CILRS method has the highest PC, VC, LC, and RL rates per kilometer driven, showing that the image-based model has the worst overall performance in terms of infractions. The AIM model shows a significant improvement as it outperforms CILRS on the DS, RC score, and the average number of infractions committed. AIM even obtains the best PC and SS infraction rates if we ignore the lower SS rate of CILRS due to the difference of more than 40% in RC scores. This significant improvement in performance shows that conditioning an AV agent on sparse goal locations with the use of an auto-regressive image-based waypoint prediction encoder can be better than conditioning an agent on navigational commands.

Table 5.2: Infraction rates of baseline models per kilometer driven.

Model	PC	VC	LC	RL	SS
CILRS	0.556	0.857	3.338	1.684	0
AIM	0.035	0.344	0.503	0.977	0.050
Latefusion	0.085	0.193	0.024	0.605	0.052
Geometric Fusion	0.047	0.387	0.013	0.367	0.081
TransFuser	0.085	0.281	0	0.649	0.106

The performance results of the two fusion baseline models show an improvement over the two image-based baseline models with respect to their DS (Table 5.1), and most notably the LC and RL infraction rates (Table 5.2). Also, between late and geometric fusion, geometric fusion provides the best results, achieving an average DS of 23.27. Also, even

though late fusion models perform almost as well as the other fusion models on average, the standard deviation of these show that there is a higher level of variance in these models than in any of the other models tested. Finally, out of all models compared in the baseline experiment, the TransFuser model comes out on top with an average DS of 26.15. When compared to geometric fusion, a general trade-off can be witnessed between safety and distance traveled as an IP about 5% lower and an RC about 5% higher lead to a DS score almost 3% higher. These improvements show that multi-modal fusion techniques are useful for improving AD models and that even simple element-wise summation techniques, can result in performance gains.

The results presented in this section are comparable to those obtained by Prakash *et al.* [3] in the original TransFuser work. Although not all performance metrics are provided, a similar growing trend in performance can be appreciated amongst the tested baseline and fusion models, as well as an overall improved capacity for global contextual reasoning in 3D environments. One thing that stands out from these results is that all of the waypoint-prediction models evaluated in long routes obtain a higher DS than the scores presented in this work. These scores range from 25.30 to 33.15 compared to our scores ranging from 15.44 to 26.15 (Table 5.1). Besides the high variance typical in IL models, this difference can be attributed to two things: the datasets used for training and the evaluation environment. Prakash *et al.* use a clear weather dataset for IL and evaluated in environments with only clear weather. Meanwhile, this work considers a 14 weather dataset while also evaluating in environments with only clear weather. It is reasonable to think that an agent trained on clear weather data would perform better in clear weather than an agent trained on different data.

5.2 Transformer HPO Sweeps (Experiment #1)

At the conclusion of the three transformer HPO sweeps are the performance metrics for the nine configurations of the TransFuser models identified in Table 4.1. After taking a closer look at the summarized performance results in Table 5.3, the first thing to notice is that the average DS of Configs 1, 2, 5, 7, 8, and 9 are below the fusion baseline model scores seen in Table 5.1. In other words, the only configurations that achieved a higher

average DS score are Configs 3, 4, and 6. Therefore, these are considered the top-three performing models of this experiment.

Table 5.3: Driving scores of transformer configurations.

Model	Max DS	DS	RC	IP
Config-1	15.53	13.06±2.29	37.89±15.01	0.57±0.09
Config-2	29.32	20.73±7.98	56.09±22.11	0.47±0.08
Config-3	27.65	24.00±3.80	68.69±5.64	0.46±0.12
Config-4	22.71	21.55±1.53	66.86±5.11	0.41±0.04
Config-5	22.00	21.27±0.84	66.53±3.73	0.40±0.07
Config-6	22.65	21.77±1.08	54.18±24.34	0.54±0.20
Config-7	24.65	18.25±5.55	62.88±3.98	0.43±0.09
Config-8	22.52	17.88±4.12	60.83±19.82	0.43±0.15
Config-9	20.10	18.80±2.24	53.77±6.62	0.47±0.07

Here, we use boldface text to highlight the top two results of each performance metric analyzed. With these highlighted values, we can see that the highest DS belongs to Config-3 and Config-6, the highest RC scores belong to Config-3 and Config-4, and the highest IP score belongs to Config-1 and Config-6. Also, the highest obtained DS in individual evaluations belong to Config-2 and Config-3. We discard Config-1 due to the relatively low DS and Config-2 since, even though a Config-2 model obtained the highest DS score thus far, it has the highest standard deviation in its average DS score, indicating Config-2 models are less likely to perform at the level that we expect them to.

Another thing to notice is the fact that the configurations with 16 attention heads have a higher DS than their counterparts with less attention heads. There is also a reduction in overall performance when it comes to configurations with 16 attention layers and with Config-1, giving a strong indication that more attention heads can be beneficial for performance independently of the number of layers. In these four cases, the DS did not even reach a value of 20, falling below all baseline fusion methods.

In the infraction rates of the transformer configurations presented in Table 5.4, we see that the top two results tend to present themselves on the configurations with more attention layers and attention heads. However, the values for each metric remain within a relatively close range, especially when compared to the infraction rates of the baseline models. We can also see that the infractions most commonly occurred are RL violations with the number of overall committed infractions ranging from 0.902 to 1.213, indicating

that collisions were less of a problem for the evaluated models than RL violations. More specifically, collisions with the environment layout are about 10 times less likely to occur than other collisions types and SS violation, and about 200 times less likely to occur than RL violation.

Table 5.4: Infraction rates of transformer configurations per kilometer driven.

Model	PC	VC	LC	RL	SS
Config-1	0.11	0.34	0.007	1.111	0.068
Config-2	0.193	0.318	0.005	1.125	0.116
Config-3	0.054	0.229	0.019	0.902	0.076
Config-4	0.088	0.288	0.05	1.05	0.076
Config-5	0.109	0.244	0.005	1.024	0.102
Config-6	0.119	0.228	0.007	1.002	0.055
Config-7	0.07	0.237	0.017	0.983	0.091
Config-8	0.053	0.291	0.004	1.133	0.089
Config-9	0.059	0.184	0.004	1.213	0.079

Table 5.5: Performance averages for each attention layer value tested.

Performance metric		4 attention layers	8 attention layers	16 attention layers
Scores	DS	21.53 \pm 5.62	24.13 \pm 4.24	18.91 \pm 3.28
	IP	0.43 \pm 0.10	0.52 \pm 0.15	0.41 \pm 0.11
	RC	65.05 \pm 14.67	60.21 \pm 12.41	60.47 \pm 10.84
Infractions	CP	1.25 \pm 0.51	0.88 \pm 0.24	0.47 \pm 0.32
	CV	2.48 \pm 0.26	2.81 \pm 0.98	1.92 \pm 1.31
	CL	0.09 \pm 0.02	0	0.08 \pm 0.15
	RV	9.79 \pm 1.02	6.84 \pm 0.55	12.41 \pm 1.21

For better analyzing the effects of varying the number of attention layers and attention heads on the performance scores and number of infractions, the performance metrics are averaged according to their attention parameters and are presented in Tables 5.5 and 5.6. Here, we see that on average, the models trained and tested with eight attention layers obtain a better DS and lower number of collisions with the static environment and RL violations. Also, the models trained with four attention layers obtain a higher RC but at the cost of committing more infractions, leading to a lower DS. On the other hand, while the models trained with 16 attention layers commit less collisions with pedestrians and vehicles, which are the most important type of infractions to consider, they commit a significantly greater amount of the less severe infractions, leading to a lower IP and

Table 5.6: Performance averages for each attention head value tested.

Performance metric		4 attention heads	8 attention heads	16 attention heads
Scores	DS	19.54 ± 5.08	22.82 ± 4.07	21.78 ± 5.42
	IP	0.41 ± 0.12	0.44 ± 0.06	0.50 ± 0.17
	RC	64.07 ± 13.85	64.82 ± 7.57	56.85 ± 14.44
Infractions	CP	1.02 ± 0.47	0.92 ± 0.74	0.66 ± 0.01
	CV	2.48 ± 0.33	2.81 ± 0.78	1.92 ± 1.02
	CL	0.12 ± 0.13	0.02 ± 0.04	0.03 ± 0.06
	RV	9.39 ± 2.52	9.47 ± 3.39	10.18 ± 2.77

consequently, a lower DS. With respect to the number of attention heads implemented in different models, we can see in Table 5.6 that the use of four attention heads tend to lead to a lower DS and more road infractions. The use of eight or 16 attention heads appear to be the ideal choice as they lead to somewhat similar results. This also leads to believe that the ideal number of attention heads could be in between eight and 16. However, these ideal values can also vary depending on the number of attention layers since it was also proven that the best score was obtained in Config-4 with eight attention layers and four attention heads. Thus, the ideal number of attention heads also depends on the number of attention layers being used in the transformer block.

5.3 TransFuser HPO Sweeps (Experiment #2)

This section presents and analyzes the results obtained from the TransFuser HPO sweeps done for each transformer configuration. We compare the best performing TransFuser models to each other and to the baseline models. Finally, we present results obtained from the WandB central dashboard platform for monitoring the training processes and the use of resources.

One of the things we conclude from the transformer sweeps is that the top three performing models are Config-3, Config-4, and Config-6. Thus, the TransFuser HPO sweeps produce models with IDs resembling the following format, Configs 3XX, 4XX, 6XX. The performance results for the Config-3 models are shown in Tables 5.7 and 5.8, the performance results for the Config-4 models are shown in Tables 5.9 and 5.10, and the performance results for the Config-6 models are shown in Tables 5.11 and 5.12. For tables

containing driving scores, we highlight the top three DS and RC scores in boldface text. Also, for all tables containing both driving scores and infraction rates, we highlight the models who obtained an average DS greater than the average DS (21.49) of the baseline late fusion model in boldface text.

Table 5.7: Driving scores of Config-3.

Model	Max DS	DS	RC	IP
Config-311	22.21	18.64±3.57	60.71±16.34	0.45±0.17
Config-312	25.58	18.09±6.59	60.52±13.74	0.42±0.12
Config-313	27.65	24.00±3.80	68.69±5.64	0.46±0.12
Config-314	16.33	12.66±3.26	29.30±9.70	0.67±0.14
Config-315	11.08	5.54±7.84	20.43±28.89	0.75±0.35
Config-316	0.39	0.19±0.27	0.19±0.27	1
Config-317	4.75	5.20±0.63	7.29±2.92	0.83±0.08
Config-318	1.71	0.86±1.21	0.97±1.38	0.95±0.07
Config-319	2.53	2.38±0.21	8.78±6.94	0.56±0.33
Config-321	17.75	16.63±1.19	52.45±27.21	0.50±0.29
Config-322	27.30	20.76±7.04	64.46±17.71	0.46±0.22
Config-323	21.41	17.25±5.37	50.79±15.73	0.50±0.00
Config-324	11.38	9.67±1.51	21.29±4.50	0.76±0.04
Config-325	7.84	3.92±5.54	4.20±5.94	0.97±0.04
Config-326	5.20	3.82±1.95	9.04±8.93	0.81±0.19
Config-327	4.67	3.66±1.44	5.48±2.84	0.85±0.06
Config-328	1.54	1.31±0.33	3.27±2.44	0.91±0.13
Config-329	1.58	0.91±0.95	2.81±0.54	0.85±0.01

From Tables 5.7, 5.9, and 5.11, we notice a significant decrease in performance when the learning rate is equal to or greater than 0.0003 (Config-XX4) since the DS ranges from 0.19 to 12.66 for Config-3, from 0 to 10.05 for Config-4, and from 0 to 14.80 for Config-6. A similar trend is seen for RC as no model with these learning rates surpassed an RC score of 36.98, falling below the worst performance of the transformer configurations tested and presented in Table 5.3. In some cases, all driving capabilities were lost as a DS of 0 indicates that there was no movement whatsoever by the ego-vehicle. Again, here we don't analyze the IP score. Also, as described in previous examples, while a high IP score could be an indication of safe driving, it can also be an indication of minimal driving, meaning that until we separate the poorly performing models from the rest, it is difficult to analyze performance based off of the IP. The same logic applies for the infraction rates seen on Tables 5.8, 5.10, 5.12.

Table 5.8: Infraction rates of Config-3.

Model	PC	VC	LC	RL	SS
Config-311	0.052	0.289	0.022	0.865	0.093
Config-312	0.136	0.193	0.012	1.298	0.084
Config-313	0.054	0.229	0.019	0.902	0.076
Config-314	0.176	0.14	0.858	0.534	0.027
Config-315	0.184	0.146	0.005	0.461	0.064
Config-316	0	0	0	0	0
Config-317	0.024	0.6	1.455	0.566	0.014
Config-318	0.4	0	0.27	0.05	0
Config-319	0.061	3.316	1.599	0.539	0
Config-321	0.128	0.167	0	0.946	0.05
Config-322	0.081	0.221	0	0.901	0.07
Config-323	0.052	0.394	0.046	0.879	0.139
Config-324	0	0.183	0.223	0.525	0.006
Config-325	0	0	0	0.188	0
Config-326	0	0.722	1.5305	0.457	0
Config-327	0	0.1285	1.545	0.352	0
Config-328	0.1875	0.4285	0.02	0.277	0.099
Config-329	0.078	1.661	0.689	0.535	0

In order to simplify the analysis of infractions and better understand how the tested hyper-parameters affect driving performances of different TransFuser configurations, we summarize the best performing TransFuser models in Table 5.13. This table consists of models who either performed better than the late fusion baseline or whose DS or RC score were amongst the top three for their respective transformer configuration. We also highlight the top three values of each driving score in boldface text, including the maximum DS values. Unlike previous tables, this one has excluded the poorly performing models, making the analysis of infractions more fair. The following discussions are with respect to this table unless specified otherwise.

Here, it is clearer to see that models whose learning rates hit past 0.0003 are prone to reduced driving capabilities as none of those models are included. Meanwhile, reducing the originally set learning rate value of 0.0001 to 0.00005 lead to performance gains in all three transformer configurations. This is seen in the max DS values obtained for each model since the Config-XX2 models all surpass the Config-XX3 models in max DS values with the exception of Config-312 and Config-313. Further evidence of this is that the best DS average, 24.21, the best max DS, 29.56, and the two best RC scores, 73.97 and 75.05, were

Table 5.9: Driving scores of Config-4.

Model	Max DS	DS	RC	IP
Config-411	27.76	21.27±6.08	69.56±7.58	0.42±0.13
Config-412	29.56	24.21±5.43	73.97±9.10	0.42±0.10
Config-413	22.71	21.55±1.53	66.86±5.11	0.41±0.04
Config-414	7.22	4.07±3.70	17.15±19.90	0.73±0.27
Config-415	0.74	0.37±0.52	0.52±0.74	0.98±0.03
Config-416	0	0	0	1
Config-417	3.86	1.93±2.73	2.40±3.39	0.91±0.13
Config-418	0	0	0	1
Config-419	3.17	2.47±0.99	21.64±17.97	0.28±0.15
Config-421	24.56	23.17±1.30	64.65±1.76	0.47±0.02
Config-422	26.22	21.82±3.91	59.68±9.16	0.48±0.03
Config-423	23.69	20.86±4.81	65.78±5.18	0.42±0.04
Config-424	13.49	10.05±3.05	19.64±3.05	0.74±0.15
Config-425	0.04	0.02±0.03	0.10±0.15	0.99±0.02
Config-426	5.85	3.09±3.90	4.80±6.31	0.94±0.09
Config-427	0.51	0.25±0.36	0.30±0.42	0.98±0.03
Config-428	0	0	0	1
Config-429	1.78	1.66±0.16	1.96±0.15	0.88±0.00

all obtained with a learning rate of 0.00005.

Despite the Config-XX1 models being included among the best performing models for all transformer configurations, their average DS scores are all below the other top-performing models of their respective transformer configurations. This suggests that a minimum learning rate for decent performance is 0.00001 as smaller values could potentially lead to more significant performance losses. However, this would require further testing to confirm.

For the most part, the IP scores range from 0.41 to 0.54. The only exceptions are the Config-622 models who obtained an IP score of 0.33. Once again, we notice a trade-off between the distance traveled and the infractions committed as this model also has the highest RC score of 75.05. However, the payoff of this trade-off is not very beneficial as the average DS is the second lowest of the top-performing Config-6 models. This specific trade-off stands out since all of the other Config-6 models follow a trend of sacrificing a higher RC score for obtaining a higher IP score. This is seen because the majority of the Config-6 RC scores are in the 50s while most of the Config-3 and Config-4 RC scores lie in the 60s. Also, Config-6 models obtained the highest average IP scores and were

Table 5.10: Infraction rates of Config-4.

Model	PC	VC	LC	RL	SS
Config-411	0.047	0.139	0.002	0.998	0.075
Config-412	0.108	0.249	0.003	0.838	0.064
Config-413	0.088	0.288	0.05	1.05	0.076
Config-414	0.003	0.285	1.269	0.528	0.024
Config-415	0	0	0	0.304	0
Config-416	0	0	0	0	0
Config-417	0.278	0.565	0.848	0.094	0
Config-418	0	0	0	0	0
Config-419	0.107	4.172	3.985	0.691	0.11
Config-421	0.054	0.217	0.014	0.764	0.056
Config-422	0.074	0.414	0.04	0.964	0.098
Config-423	0.065	0.374	0.022	0.831	0.097
Config-424	0.017	0.261	1.463	0.431	0.012
Config-425	0.109	0	0	0.054	0
Config-426	0	0.067	0	0.157	0
Config-427	0.101	0	0	0.101	0
Config-428	0	0	0	0	0
Config-429	0	0.723	2.658	0	0

the only transformer configuration to obtain average IP scores above 0.50, meaning that the combination of models with eight attention layers and 16 attention heads cause the system to prioritize safety slightly more than the other transformer configurations. Even though this difference is slight, the standard deviations that accompany these models with IP scores above 0.50 indicate that the distance-safety trade-off leans towards prioritizing safety more than RC.

When analyzing the infraction metrics corresponding to the best performing TransFuser models, we come to see that the rate of each infraction committed by the three transformer configurations are relatively similar. By averaging the five infraction rates of the best performing TransFuser models, we obtain the infraction rate averages and standard deviations presented in Table 5.14. Here, we get a general idea of how well the top-performing models handle the five penalty infractions considered for this analysis. Organized in descending order, the most commonly occurring infractions over any TransFuser model are RL, VC, SS, PC, and LC. The fact that the same order holds true for any TransFuser model, and that the values are relatively within the same range, means that the hyper-parameters tuned in this work did not significantly affect the rate at which the main penalty infractions occur.

Table 5.11: Driving scores of Config-6.

Model	Max DS	DS	RC	IP
Config-611	24.12	19.21±4.47	58.68±8.70	0.44±0.12
Config-612	24.75	21.09±3.68	56.25±12.28	0.51±0.16
Config-613	22.65	21.77±1.08	54.18±24.34	0.54±0.20
Config-614	12.61	9.89±3.84	36.98±28.53	0.60±0.28
Config-615	7.53	3.77±5.32	16.05±22.70	0.75±0.36
Config-616	2.73	1.36±1.93	1.62±2.29	0.97±0.05
Config-617	0.11	0.05±0.08	0.06±0.09	0.99±0.01
Config-618	0	0	0	1
Config-619	3.69	2.91±1.10	8.31±8.47	0.64±0.44
Config-621	29.22	22.53±5.80	57.02±6.12	0.53±0.13
Config-622	26.89	20.90±5.51	75.05±9.85	0.33±0.10
Config-623	24.35	21.55±3.31	64.17±7.47	0.45±0.07
Config-624	17.78	14.80±3.44	35.52±3.29	0.65±0.07
Config-625	0	0	0	1
Config-626	0	0	0	1
Config-627	8.43	6.34±2.96	9.71±6.18	0.81±0.14
Config-628	2.48	1.95±0.75	2.78±0.78	0.92±0.08
Config-629	2.23	2.04±0.27	3.44±2.06	0.83±0.17

5.4 Sweep Metrics

As of now we have focused solely on analyzing the driving performances resulting from the applied TransFuser HPO sweeps. In this section, we present charts and a table containing metrics relevant to the actual sweep training sessions performed on an HPC system. This data was recollected with the WandB tool and includes the following metrics: training time, training loss, validation loss, best validation epoch, best validation loss, and GPU usage. The information in this section is taken directly from the WandB dashboard as we look at the Config-3, Config-4, and Config-6 training sessions.

5.4.1 Training and Validation Losses

In Figure 5.1, we present a parallel coordinates chart containing information regarding how the optimization hyper-parameters and the training loss relate to the best validation loss obtained during training for a Config-4 model. Here, the values in the learning rate and training loss axes are displayed in logarithmic scale for visual clarity, the hot (yellow) lines represent lower validation losses, and the cold (blue) lines represent higher validation

Table 5.12: Infraction rates of Config-6.

Model	PC	VC	LC	RL	SS
Config-611	0.038	0.237	0.023	1.089	0.138
Config-612	0.064	0.223	0.02	0.853	0.035
Config-613	0.119	0.228	0.007	1.002	0.055
Config-614	0.025	0.271	0.949	0.738	0.062
Config-615	0.012	0.2	1.147	0.387	0.006
Config-616	0	0.035	0.283	0.089	0
Config-617	0	0	0	0	0
Config-618	0	0	0	0	0
Config-619	0.028	1.546	1.195	0.346	0.051
Config-621	0.043	0.16	0.032	0.769	0.087
Config-622	0.097	0.325	0.007	1.307	0.089
Config-623	0.059	0.438	0.079	0.719	0.088
Config-624	0.064	0.234	0.176	0.58	0.038
Config-625	0	0	0	0	0
Config-626	0	0	0	0	0
Config-627	0	0.579	0.333	0.924	0
Config-628	0	0.076	1.413	0.076	0
Config-629	0	1.289	1.123	0.487	0

losses. We have also excluded the outlier values whose best validation losses were on the other side of the heat-map spectrum as they make analyzing the chart more difficult.

First, we notice that the batch size has no noticeable effect on the resulting model performances as both values tested lead to similar results. On the other hand, a cluster of hot lines can be seen gathering at the center of the learning rate axis. This confirms the existence of a sweep spot where values outside of that range lead to significantly worse performances. We also see that training losses generally tend to zero as the learning rate decreases within its sweep spot. Complementary to this, we see a general trend in Figure 5.2 as most models with a decent learning rate tend to zero while others converge to a sub-optimal solution.

In the case of the validation losses checked during training, they typically range from about 0.27 to 0.45. Unlike Figure 5.2 that excludes extreme training losses, Figure 5.3 shows how models trained with extreme learning rates jump to values at around 1.5 before reaching epoch 40. Also, in Figure 5.4, we compare multiple instances of transformer configurations with instances of the waypoint-prediction baseline models. Here, the different in loss is much more discernible since the three baseline models lie above all transformer-based

Table 5.13: Summary of the best performing models from the TransFuser HPO sweeps.

Model	Max DS	DS	RC	IP
Config-311	22.21	18.64±3.57	60.71±16.34	0.45±0.17
Config-313	27.65	24.00±3.80	68.69±5.64	0.46±0.12
Config-322	27.3	20.76±7.04	64.46±17.71	0.46±0.22
Config-411	27.76	21.27±6.08	69.56±7.58	0.42±0.13
Config-412	29.56	24.21±5.43	73.97±9.10	0.42±0.10
Config-413	22.71	21.55±1.53	66.86±5.11	0.41±0.04
Config-421	24.56	23.17±1.30	64.65±1.76	0.47±0.02
Config-422	26.22	21.82±3.91	59.68±9.16	0.48±0.03
Config-611	24.12	19.21±4.47	58.68±8.70	0.44±0.12
Config-612	24.75	21.09±3.68	56.25±12.28	0.51±0.16
Config-613	22.65	21.77±1.08	54.18±24.34	0.54±0.20
Config-621	29.22	22.53±5.80	57.02±6.12	0.53±0.13
Config-622	26.89	20.90±5.51	75.05±9.85	0.33±0.10
Config-623	24.35	21.55±3.31	64.17±7.47	0.45±0.07

Table 5.14: Average infraction rates for the best performing TransFuser models.

Config IDs	PC	VC	LC	RL	SS
3	0.062±0.016	0.243±0.037	0.014±0.012	0.889±0.021	0.080±0.012
4	0.074±0.025	0.261±0.101	0.022±0.022	0.923±0.118	0.074±0.016
6	0.070±0.032	0.269±0.098	0.028±0.027	0.957±0.221	0.082±0.035
3, 4, 6	0.070±0.025	0.261±0.085	0.023±0.022	0.930±0.155	0.079±0.024

models.

5.4.2 HPC Usage

The WandB platform provides access to various system metrics including training times, GPU usage, power usage, time spent accessing memory, among others. However, the problem with visualizing most of these metrics when applied to an HPC system is that, as regular users, we don't have complete isolation of HPC resources. This means that when tracking HPC system metrics with WandB, the recorded values represent the usage of all users, not just a single one. For example, in Figure 5.5, we filter the chart information to show the average usage percentage of only the three GPUs that were used for carrying out a set of three HPO sweeps with the hyper-parameters mentioned in Table 4.2. In this case, GPU 0 was used for performing an HPO sweep on the Config-6 model, GPU 1 for performing an HPO sweep on the Config-4 model, and GPU 3 for performing an

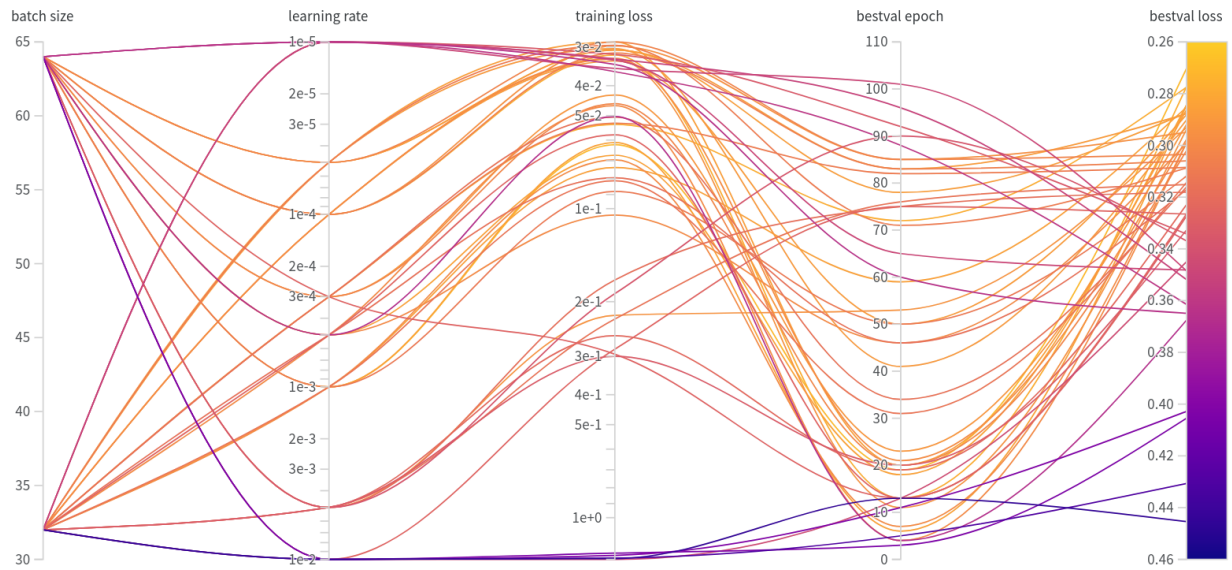


Figure 5.1: Config-4 Parallel Coordinates Chart.

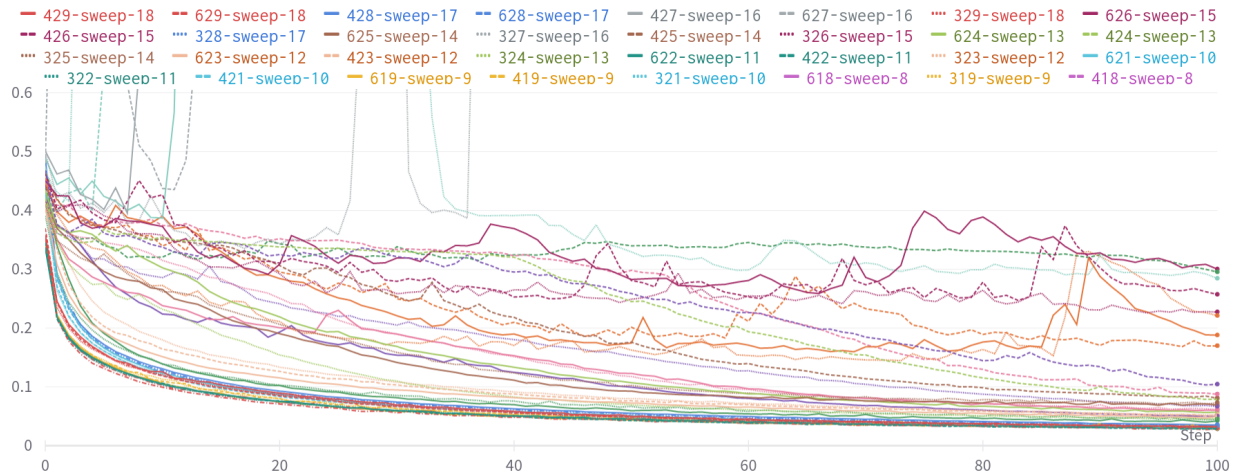


Figure 5.2: Training Losses vs. Epochs in TransFuser HPO Experiment.

HPO sweep on the Config-3 model. Since at the time of running sweeps the least occupied GPUs were chosen, we can trust that Figure 5.5 more accurately depicts the overall GPU computing power used for every sweep performed to be at about 70% to 80%.

Complementary to the GPU usage is the amount of time that each GPU dedicated for training. For this, we state the average training times in hours and minutes of the baseline and transformer configurations in Table 5.15, and the average training times of all TransFuser configurations in Table 5.16. From Table 5.15, we see that training most of the baseline models required less than half the time than training any of the TransFuser models. The only exception to this is the geometric fusion model that takes about twice

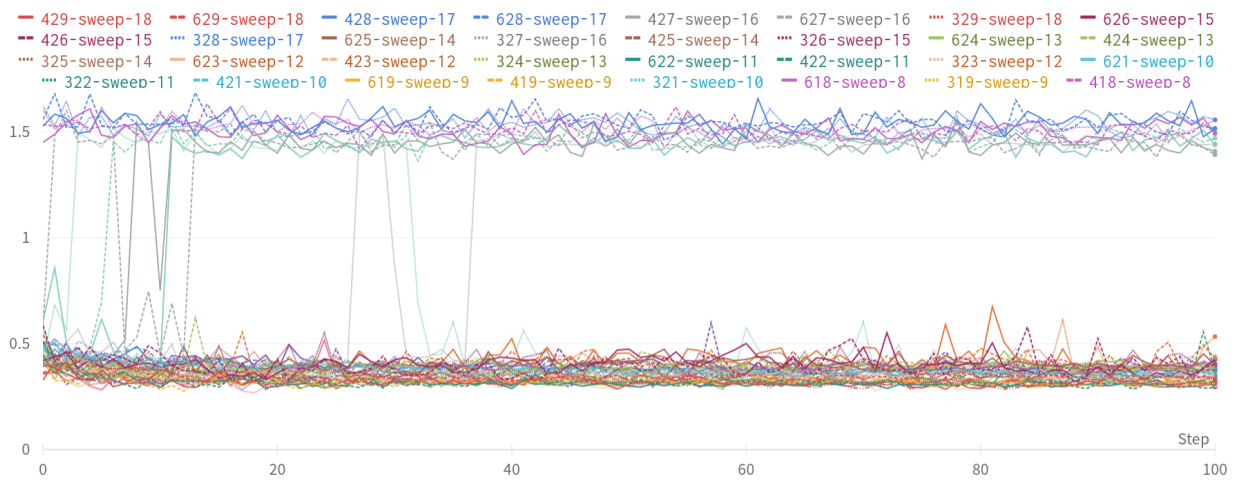


Figure 5.3: Validation Losses vs Epochs in TransFuser HPO Experiment.

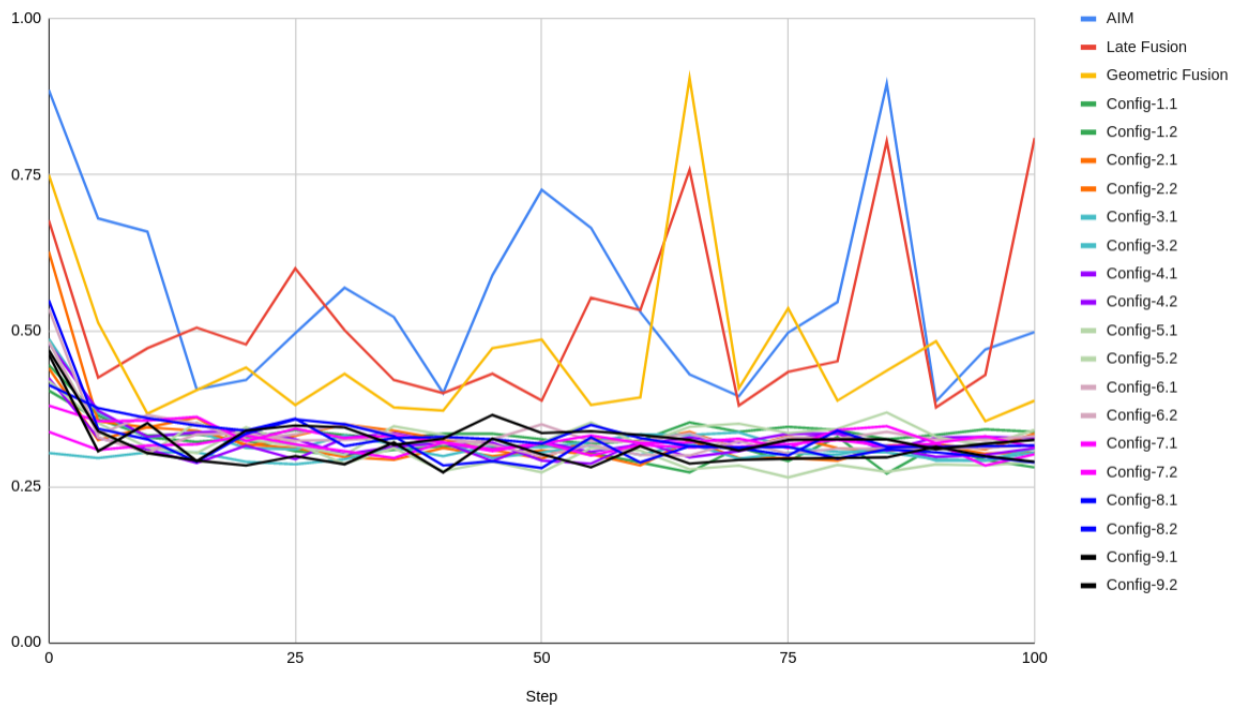


Figure 5.4: Validation Losses vs Epochs in Baseline and Transformer HPO Experiments.

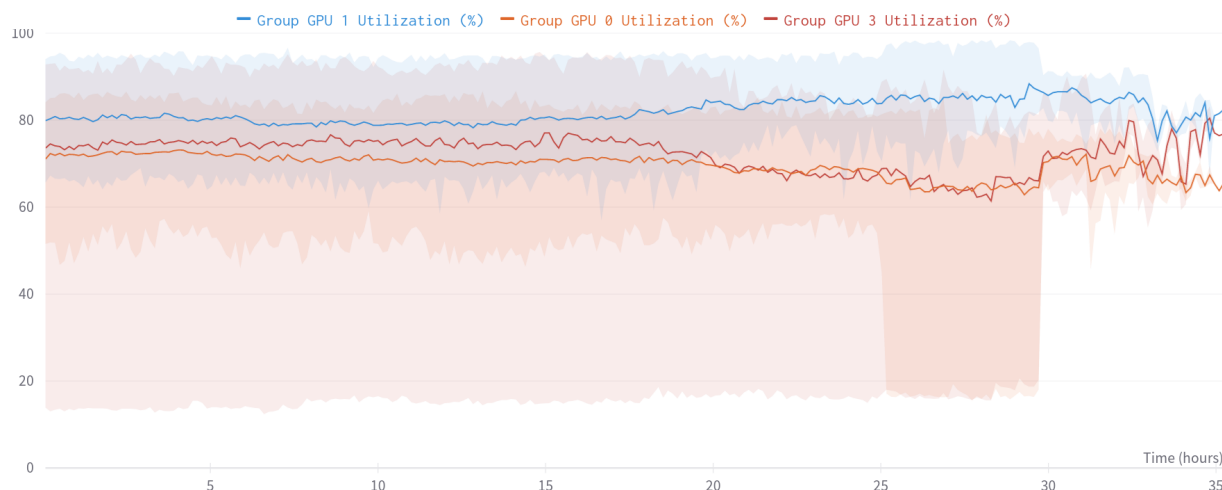


Figure 5.5: GPU Usage Percentage vs. Training Hours.

as long to train. Despite geometric fusion being the overall second best approach studied in this work in terms of performance, this shows a big flaw in this approach as this method requires more computation time for obtaining performance results equal to or barely below the TransFuser models.

Another notable pattern in Table 5.15 is that increasing the number of layers also increases the training time while increasing the number of attention heads does not have the same effect. From this, we can deduct that training models with four, eight, and 16 attention layers, requires approximately 19.6 hours, 22.1 hours, and 28.2 hours, respectively. This outcome is expected because additional attention heads can be computed in parallel while attention layers can not, and thus, would require additional training time.

In the training times shown in Table 5.16, we notice a pattern additional to the ones resulting from modifying attention hyper-parameters. The performance metric results have already shown no indication of a causal relationship between the batch size and driving performance. However, when it comes to training times, the batch size is the hyper-parameter that has the greatest impact. Even though halving the batch size makes each training iteration faster, it also doubles the total number training iterations needed, making the overall training time longer.

Table 5.15: Average training times for baseline and transformer sweep models.

Models	Average Training Times
CILRS	4h 5m
AIM	6h 42m
Late Fusion	7h 10m
Geometric Fusion	54h 14m
Config-1	19h 41m
Config-2	19h 50m
Config-3	19h 21m
Config-4	22h 25m
Config-5	22h 47m
Config-6	20h 59m
Config-7	26h 19m
Config-8	28h 43m
Config-9	29h 31m

Table 5.16: Average training times for TransFuser configurations.

Model IDs	Config-3XX	Config-4XX	Config-6XX
Config-X11	14h 9m	17h 12m	19h 20m
Config-X12	14h 56m	17h 10m	19h 30m
Config-X13	19h 21m	22h 25m	20h 59m
Config-X14	15h 28m	18h 29m	20h
Config-X15	18h	19h 57m	19h 44m
Config-X16	17h 27m	18h 15m	18h 58m
Config-X17	16h 17m	18h 10m	20h 27m
Config-X18	19h 8m	19h 6m	20h 15m
Config-X19	15h 49m	20h 43m	24h 11m
Config-X21	22h 15m	29h 44m	27h 43m
Config-X22	22h 42m	29h 44m	30h 13m
Config-X23	21h 15m	26h 45m	26h 43m
Config-X24	23h 52m	29h 10m	30h 11m
Config-X25	26h 21m	30h 24m	32h 31m
Config-X26	25h 48m	28h 31m	31h 1m
Config-X27	24h 6m	27h 24m	27h 10m
Config-X28	24h 6m	28h 54m	26h 12m
Config-X29	24h 4m	28h 2m	25h 54m

Chapter 6

Conclusions

The use of attention-based models applied to AD tasks is a relatively new field of research which requires further investigation in order to evaluate, optimize and evolve existing models and extrapolate the full potential of attention mechanisms. These types of models have shown significant improvements in the field of AD when compared to other state-of-the-art models. TransFuser is one of these models which makes use of transformer architectures to focus on fusing multiple sensor modalities in order to learn a driving policy with a greater global context which is often needed to handle challenging traffic scenarios.

This work implements HPO through grid-search sweeps in order to test various configurations of the TransFuser model and compare them to each other and to four baseline models. For training, we used an open-source dataset along with the CARLA simulation framework and evaluated the impact of the two hyper-parameters pertaining to the transformers and the two pertaining to the optimization process, namely the number of attention layers, attention heads, the learning rate, and the batch size.

After conducting three HPO experiments, all of which require the use of HPC, we analyze the performance of multiple transformer and TransFuser variations on the CARLA Leaderboard benchmark and conclude four main things regarding the hyper-parameters used to optimize this AD model. First, regarding the attention hyper-parameters, the optimal number of attention layers appears to be in between four and 16, more likely closer to eight, since the performances of models trained with eight layers have resulted in the overall best performance while those trained with 16 layers tend to under-perform. This,

however, is true when considering the main performance metric (DS) of the Leaderboard benchmark. When considering other metrics such as RC, four attention layers led to higher scores at the costing of getting more points deducted from the IP score. Meanwhile, even though the number of attention heads has a less overall affect on driving performance, general averages of performance metrics show that more attention heads can lead to some performance gains independently of the number of attention layers. This is beneficial when considering that attention heads are used in a parallelized manner without adding significant time to the training process.

Second, regarding the optimization hyper-parameters, the optimal learning rate value tested in this work is 0.00005 as most of the top-performing results are centered around this value. However, values within the range of 0.00001 to 0.0001 also appear to work approximately the same in terms of driving performance. Also, while the batch size does not affect driving performances, it does significantly affect the time needed for training as smaller batch sizes involve more iterations to compute. Considering this, the best thing to do regarding this hyper-parameter is to use the maximum batch size that can fit on a single GPU so as to take the most advantage of the available hardware.

From our first baseline experiment, we confirmed that TransFuser is a state-of-the-art improvement on other AD models. From the second experiment focused on performing HPO on the system model design, we determined that the best attention hyper-parameters taken from $\{4,8,16\}$ are those that we labeled as Config-3, Config-4, and Config-6. Of these three model configurations, we also found that the Config-6 model prioritizes safety more than the other top-performing models. Finally, from the third experiment focused on the optimization of non-transformer hyper-parameters, we can say that while these parameters have a less affect on driving performance, they should still be chosen with caution so as to make the most of the available hardware and avoid needless performance losses.

After completing our experimentation process of training and evaluating AD models through HPO sweeps, there are two notable models that stand out above the rest. Even though no DS averages were higher than the ones obtained during for the original TransFuser model in the initial baseline experiment, both the Config-621 model and the Config-412 models obtained higher maximum DS values of 29.22 and 29.56, surpassing the original maximum value of 29.16. With respect to the average RC scores, both the Config-

412 and the Config-622 managed to obtain higher averages of 73.97 and 75.05, surpassing the original value of 70.08. With respect to the average IP scores, seven of our models managed to obtain higher IP values than the original value of 0.45 while still obtaining sufficient DS points to be on a par with the other top-performing models summarized in this work. Of these seven, the three that stand out are the Config-612, Config-621, and Config-613 models who obtained average IP values of 0.51, 0.53, and 0.54.

6.1 Future Works

This work provides an analysis on the resulting performances caused by varying the number of attention layers and attention heads within a transformer architecture. However, there a number of other hyper-parameters and system components that require further investigation and tuning in order to determine their effect on AD performance and optimize state-of-the-art models such as TransFuser. With respect to the self-attention module implemented by TransFuser, these hyper-parameters could include the attention, residual and embedding dropout, the scale resolution at which fusion occurs, and the size of the embedding layer.

Regarding the other model components of TransFuser, past studies have shown state-of-the-art improvements by modifying the expert driving policy and the PID controller. Therefore, one approach for future works to take is to combine HPO with modifications made to the driving dataset and to the controller output. The main options with respect to the driving policy are to generate driving data from zero through simulation, using other driving datasets for IL, and improving upon the quantity or quality of existing datasets. This combined implementing could result in a more optimized version of the TransFuser AD model and, consequently, lead to an overall improved driving performance.

At the simulation level, it would also be interesting to implement a similar work on the latest version of the CARLA simulator or on other driving simulators. Recent versions of CARLA have improved integration with tools such as ROS and SUMO that can be used to go beyond the aspect of autonomous learning and delve into the area of robotics or urban simulation. In doing so, it would also be interesting to implement the TransFuser model to these different tools and see how it adapts.

Bibliography

- [1] NHTSA, “U.s. department of transportation’s fatality analysis reporting system (fars),” 2019. [Online]. Available: <https://www-fars.nhtsa.dot.gov/Main/index.aspx>
- [2] CARLA, “Carla autonomous driving leaderboard,” 2019. [Online]. Available: <https://leaderboard.carla.org/>
- [3] A. Prakash, K. Chitta, and A. Geiger, “Multi-modal fusion transformer for end-to-end autonomous driving,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 7077–7087. [Online]. Available: <https://github.com/autonomousvision/transfuser>
- [4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.
- [5] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, “Improving language understanding by generative pre-training.” [Online]. Available: http://openai-assets.s3.amazonaws.com/research-covers/language-unsupervised/language_understanding_paper.pdf
- [6] R. Hussain and S. Zeadally, “Autonomous cars: Research results, issues, and future challenges,” *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1275–1313, 2018.
- [7] NHTSA, “2016 fatal motor vehicle crashes: Overview,” 2016. [Online]. Available: <https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812456>

- [8] W. H. Organization, “Road traffic injuries,” June 2021. [Online]. Available: <https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries>
- [9] SAE, “Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles,” 2021. [Online]. Available: https://www.sae.org/standards/content/j3016_202104/
- [10] A. Ghansiyal, M. Mittal, and A. K. Kar, “Information management challenges in autonomous vehicles: a systematic literature review,” *Journal of Cases on Information Technology (JCIT)*, vol. 23, no. 3, pp. 58–77, 2021.
- [11] T. Campisi, A. Severino, M. A. Al-Rashid, and G. Pau, “The development of the smart cities in the connected and autonomous vehicles (cavs) era: From mobility patterns to scaling in cities,” *Infrastructures*, vol. 6, no. 7, p. 100, 2021.
- [12] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA grand challenge: the great robot race*. Springer, 2007, vol. 36.
- [13] —, *The DARPA urban challenge: autonomous vehicles in city traffic*. springer, 2009, vol. 56.
- [14] K. Muhammad, A. Ullah, J. Lloret, J. Del Ser, and V. H. C. de Albuquerque, “Deep learning for safe autonomous driving: Current challenges and future directions,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 7, pp. 4316–4336, 2020.
- [15] N. Ismayilova and E. Ismayilov, “Convergence of hpc and ai: Two directions of connection,” *Azerbaijan Journal of High Performance Computing*, vol. 1, no. 2, pp. 179–184, 2018.
- [16] K. Huang, B. Shi, X. Li, X. Li, S. Huang, and Y. Li, “Multi-modal sensor fusion for auto driving perception: A survey,” *arXiv preprint arXiv:2202.02703*, 2022.
- [17] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, “Exploring the limitations of behavior cloning for autonomous driving,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9329–9338.

- [18] A. Bolor, K. Garimella, X. He, C. Gill, Y. Vorobeychik, and X. Zhang, “Attacking vision-based perception in end-to-end autonomous driving models,” *Journal of Systems Architecture*, vol. 110, p. 101766, 2020.
- [19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.
- [20] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An open urban driving simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16. [Online]. Available: <http://carla.org/>
- [21] A. Prakash, K. Chitta, and A. Geiger, “Supplementary material for multi-modal fusion transformer for end-to-end autonomous driving,” 2021.
- [22] S. E. Shladover, “Connected and automated vehicle systems: Introduction and overview,” *Journal of Intelligent Transportation Systems*, vol. 22, no. 3, pp. 190–200, 2018.
- [23] J. McCarthy, “Computer controlled cars,” *University of Stanford*, 1968. [Online]. Available: www-formal.stanford.edu/jmc/progress/cars/cars.html
- [24] D. A. Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” *Advances in neural information processing systems*, vol. 1, 1988.
- [25] G. Xiao, “Research on key manufacturing technologies of new energy vehicles based on artificial intelligence,” in *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*. IEEE, 2020, pp. 451–454.
- [26] J. D. Rupp and A. G. King, “Autonomous driving-a practical roadmap,” SAE Technical Paper, Tech. Rep., 2010.
- [27] Bloomberg, “Initiative on cities and autonomous vehicles,” 2017. [Online]. Available: <https://avsincities.bloomberg.org/>
- [28] A. Broggi, P. Cerri, M. Felisa, M. C. Laghi, L. Mazzei, and P. P. Porta, “The vislab intercontinental autonomous challenge: an extensive test for a platoon of intelligent

- vehicles,” *International Journal of Vehicle Autonomous Systems*, vol. 10, no. 3, pp. 147–164, 2012.
- [29] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3354–3361.
- [30] N. Kalra and S. M. Paddock, “Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?” *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016.
- [31] G. P. Vivan, N. Goberville, Z. Asher, N. Brown, and J. Rojas, “No cost autonomous vehicle advancements in carla through ros,” SAE Technical Paper, Tech. Rep., 2021.
- [32] J. Leudet, T. Mikkonen, F. Christophe, and T. Männistö, “Virtual environment for training autonomous vehicles,” in *Annual Conference Towards Autonomous Robotic Systems*. Springer, 2018, pp. 159–169.
- [33] W. Li, C. Pan, R. Zhang, J. Ren, Y. Ma, J. Fang, F. Yan, Q. Geng, X. Huang, H. Gong *et al.*, “Aads: Augmented autonomous driving simulation using data-driven algorithms,” *Science robotics*, 2019.
- [34] K. Holt, “Waymo’s autonomous vehicles have clocked 20 million miles on public roads,” 2021. [Online]. Available: <https://www.engadget.com/waymo-autonomous-vehicles-update-san-francisco-193934150.html>
- [35] M. R. Bachute and J. M. Subhedar, “Autonomous driving architectures: Insights of machine learning and deep learning algorithms,” *Machine Learning with Applications*, vol. 6, p. 100164, 2021.
- [36] A. Faisal, M. Kamruzzaman, T. Yigitcanlar, and G. Currie, “Understanding autonomous vehicles,” *Journal of transport and land use*, vol. 12, no. 1, pp. 45–72, 2019.
- [37] M. Bugała, “Algorithms applied in autonomous vehicle systems,” *Szybkobieżne Pojazdy Gąsienicowe*, vol. 50, no. 4, pp. 119–138, 2018.

- [38] M. Liang, B. Yang, S. Wang, and R. Urtasun, “Deep continuous fusion for multi-sensor 3d object detection,” in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 641–656.
- [39] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun, “Multi-task multi-sensor fusion for 3d object detection,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7345–7353.
- [40] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [41] Y. Chen, D. Zhao, L. Lv, and C. Li, “A visual attention based convolutional neural network for image classification,” in *2016 12th World Congress on Intelligent Control and Automation (WCICA)*. IEEE, 2016, pp. 764–769.
- [42] C. Cao, X. Liu, Y. Yang, Y. Yu, J. Wang, Z. Wang, Y. Huang, L. Wang, C. Huang, W. Xu *et al.*, “Look and think twice: Capturing top-down visual attention with feedback convolutional neural networks,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2956–2964.
- [43] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-end object detection with transformers,” in *European conference on computer vision*. Springer, 2020, pp. 213–229.
- [44] L. L. Li, B. Yang, M. Liang, W. Zeng, M. Ren, S. Segal, and R. Urtasun, “End-to-end contextual perception and prediction with interaction transformer,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5784–5791.
- [45] C. Yu, X. Ma, J. Ren, H. Zhao, and S. Yi, “Spatio-temporal graph transformer networks for pedestrian trajectory prediction,” in *European Conference on Computer Vision*. Springer, 2020, pp. 507–523.
- [46] Y. Chen, C. Dong, P. Palanisamy, P. Mudalige, K. Muelling, and J. M. Dolan, “Attention-based hierarchical deep reinforcement learning for lane change behaviors

- in autonomous driving,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0.
- [47] B. Zhou, P. Krähenbühl, and V. Koltun, “Does computer vision matter for action?” *Science Robotics*, 2019.
- [48] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer *et al.*, “Autonomous driving in urban environments: Boss and the urban challenge,” *Journal of field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.
- [49] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt *et al.*, “Towards fully autonomous driving: Systems and algorithms,” in *2011 IEEE intelligent vehicles symposium (IV)*. IEEE, 2011, pp. 163–168.
- [50] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, “Deepdriving: Learning affordance for direct perception in autonomous driving,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2722–2730.
- [51] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, “A survey of autonomous driving: Common practices and emerging technologies,” *IEEE access*, vol. 8, pp. 58 443–58 469, 2020.
- [52] I. Sobh, L. Amin, S. Abdelkarim, K. Elmadawy, M. Saeed, O. Abdeltawab, M. Gamal, and A. El Sallab, “End-to-end multi-modal sensors fusion system for urban automated driving,” *Advances in neural information processing systems workshops*, 2018.
- [53] J. Heylen, S. Iven, B. De Brabandere, J. Oramas, L. Van Gool, and T. Tuytelaars, “From pixels to actions: Learning to drive a car with deep neural networks,” in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 606–615.
- [54] M. Toromanoff, E. Wirbel, and F. Moutarde, “End-to-end model-free reinforcement learning for urban driving using implicit affordances,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 7153–7162.

- [55] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, “Deep reinforcement learning for autonomous driving: A survey,” *IEEE Transactions on Intelligent Transportation Systems*, 2021.
- [56] J. Koutník, G. Cuccu, J. Schmidhuber, and F. Gomez, “Evolving large-scale neural networks for vision-based reinforcement learning,” in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 2013, pp. 1061–1068.
- [57] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, “End to end learning for self-driving cars,” *arXiv preprint arXiv:1604.07316*, 2016.
- [58] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. Cun, “Off-road obstacle avoidance through end-to-end learning,” *Advances in neural information processing systems*, vol. 18, 2005.
- [59] A. Attia and S. Dayan, “Global overview of imitation learning,” *arXiv preprint arXiv:1801.06503*, 2018.
- [60] A. Torralba and A. A. Efros, “Unbiased look at dataset bias,” in *CVPR 2011*. IEEE, 2011, pp. 1521–1528.
- [61] P. De Haan, D. Jayaraman, and S. Levine, “Causal confusion in imitation learning,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [62] B. Neal, S. Mittal, A. Baratin, V. Tantia, M. Scicluna, S. Lacoste-Julien, and I. Mitliagkas, “A modern take on the bias-variance tradeoff in neural networks,” *arXiv preprint arXiv:1810.08591*, 2018.
- [63] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, “End-to-end driving via conditional imitation learning,” in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 4693–4700.
- [64] K.-T. Lai, C.-C. Lin, C.-Y. Kang, M.-E. Liao, and M.-S. Chen, “Vivid: Virtual environment for visual deep learning,” in *Proceedings of the 26th ACM international conference on Multimedia*, 2018, pp. 1356–1359.

- [65] J. Bernhard, K. Esterle, P. Hart, and T. Kessler, “Bark: Open behavior benchmarking in multi-agent environments,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 6201–6208.
- [66] P. Kaur, S. Taghavi, Z. Tian, and W. Shi, “A survey on simulators for testing self-driving cars,” in *2021 Fourth International Conference on Connected and Autonomous Driving (MetroCAD)*. IEEE, 2021, pp. 62–70.
- [67] P. Pirri, C. Pahl, N. El Ioini, and H. R. Barzegar, “Towards cooperative maneuvering simulation: Tools and architecture,” 01 2021, pp. 1–6.
- [68] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, “Learning by cheating,” in *Conference on Robot Learning*. PMLR, 2020, pp. 66–75.
- [69] —, “Learning by cheating,” in *Conference on Robot Learning*. PMLR, 2020, pp. 66–75.
- [70] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [71] A. Sauer, N. Savinov, and A. Geiger, “Conditional affordance learning for driving in urban environments,” in *Conference on Robot Learning*. PMLR, 2018, pp. 237–252.
- [72] Z. Li, T. Motoyoshi, K. Sasaki, T. Ogata, and S. Sugano, “Rethinking self-driving: Multi-task knowledge for better generalization and accident explanation ability,” *arXiv preprint arXiv:1809.11100*, 2018.
- [73] X. Liang, T. Wang, L. Yang, and E. Xing, “Cirl: Controllable imitative reinforcement learning for vision-based self-driving,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 584–599.
- [74] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation.” *In Proc. of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

- [75] Y. Xiao, F. Codevilla, A. Gurram, O. Urfalioglu, and A. M. López, “Multimodal end-to-end autonomous driving,” *IEEE Transactions on Intelligent Transportation Systems*, 2020.
- [76] A. Behl, K. Chitta, A. Prakash, E. Ohn-Bar, and A. Geiger, “Label efficient visual abstractions for autonomous driving,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 2338–2345.
- [77] D. Bahdanau, K. H. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” in *3rd International Conference on Learning Representations, ICLR 2015*, 2015.
- [78] H. Chefer, S. Gur, and L. Wolf, “Transformer interpretability beyond attention visualization,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 782–791.
- [79] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun, “Point transformer,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 16 259–16 268.
- [80] S. Liu, E. Johns, and A. J. Davison, “End-to-end multi-task learning with attention,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 1871–1880.
- [81] J. Cheng, L. Dong, and M. Lapata, “Long short-term memory-networks for machine reading,” in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 551–561.
- [82] B. Jaeger, “Expert drivers for autonomous driving,” Master’s thesis, Eberhard Karls Universität Tübingen, 2021.
- [83] D. Chen, V. Koltun, and P. Krähenbühl, “Learning to drive from a world on rails,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 15 590–15 599.

- [84] K. Chitta, A. Prakash, and A. Geiger, “Neat: Neural attention fields for end-to-end autonomous driving,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 15 793–15 803.
- [85] Y. Vishnusai, T. R. Kulakarni, and K. Sowmya Nag, “Ablation of artificial neural networks,” in *International Conference on Innovative Data Communication Technologies and Application*. Springer, 2019, pp. 453–460.
- [86] R. E. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 2015, vol. 2456.
- [87] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [88] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *International Conference on Learning Representations*, 2018.
- [89] C. Luo, X. Yang, and A. Yuille, “Self-supervised pillar motion learning for autonomous driving,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 3183–3192.
- [90] M. Feurer and F. Hutter, “Hyperparameter optimization,” in *Automated machine learning*. Springer, Cham, 2019, pp. 3–33.
- [91] J.-M. Dufour and J. Neves, “Finite-sample inference and nonstandard asymptotics with monte carlo tests and r,” in *Handbook of statistics*. Elsevier, 2019, vol. 41, pp. 3–31.
- [92] T. Sterling, M. Anderson, and M. Brodowicz, *High Performance Computing, Modern Systems and Practices*, 1st ed. Morgan Kaufmann, 2017.
- [93] D. R. Ferreira and J. Contributors, “Using hpc infrastructures for deep learning applications in fusion research,” *Plasma Physics and Controlled Fusion*, vol. 63, no. 8, p. 084006, 2021.
- [94] J. J. Dongarra, H. W. Meuer, E. Strohmaier *et al.*, “Top500 supercomputer sites,” *Supercomputer*, vol. 13, pp. 89–111, 1997. [Online]. Available: <https://www.top500.org>

-
- [95] A. Majeed and S. Lee, “Applications of machine learning and high-performance computing in the era of covid-19,” *Applied System Innovation*, vol. 4, no. 3, p. 40, 2021.
- [96] E. Alfianto, A. Sa’diyah, F. Rusydi, and I. Puspitasari, “High-performance computing (hpc) design to improve the quality of introduction of parallel computing lectures,” in *IOP Conference Series: Materials Science and Engineering*, vol. 462, no. 1. IOP Publishing, 2019, p. 012020.
- [97] R. Labib, “Utilizing high performance computing to improve the application of machine learning for time-efficient prediction of buildings’ daylighting performance,” in *Journal of Physics: Conference Series*, vol. 2069, no. 1. IOP Publishing, 2021, p. 012153.