



**HAL**  
open science

# End-to-End Autonomous Driving in CARLA: A Survey

Youssef Al Ozaibi, Manolo Dulva Hina, Amar Ramdane-Cherif

► **To cite this version:**

Youssef Al Ozaibi, Manolo Dulva Hina, Amar Ramdane-Cherif. End-to-End Autonomous Driving in CARLA: A Survey. IEEE Access, 2024, 12, pp.146866 - 146900. 10.1109/access.2024.3473611 . hal-04954649

**HAL Id: hal-04954649**

**<https://uvsq.hal.science/hal-04954649v1>**

Submitted on 18 Feb 2025

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

## SURVEY

# End-to-End Autonomous Driving in CARLA: A Survey

YOUSSEF AL OZAIBI<sup>1,2</sup>, MANOLO DULVA HINA<sup>1</sup>, (Member, IEEE),  
AND AMAR RAMDANE-CHERIF<sup>2</sup>

<sup>1</sup>ECE Paris School of Engineering, 75015 Paris, France

<sup>2</sup>LISV Laboratory, Université de Versailles Paris-Saclay, 78140 Vélizy-Villacoublay, France

Corresponding author: Youssef Al Ozaibi (yalozaibi@ece.fr)

This work was supported by the École centrale d'électronique (ECE) Paris Engineering School, Laboratoire d'Ingénierie des Systèmes de Versailles (LISV), Université de Versailles—Paris-Saclay.

**ABSTRACT** Autonomous Driving (AD) has evolved significantly since its beginnings in the 1980s, with continuous advancements driven by both industry and academia. Traditional AD systems break down the driving task into smaller modules—such as perception, localization, planning, and control—and optimize them independently. In contrast, end-to-end models use neural networks to map sensory inputs directly to vehicle controls, optimizing the entire driving process as a single task. Recent advancements in deep learning have driven increased interest in end-to-end models, which is the central focus of this review. In this survey, we discuss how CARLA-based state-of-the-art implementations address various issues encountered in end-to-end autonomous driving through various model inputs, outputs, architectures, and training paradigms. To provide a comprehensive overview, we additionally include a concise summary of these methods in a single large table. Finally, we present evaluations and discussions of the methods, and suggest future avenues to tackle current challenges faced by end-to-end models.

**INDEX TERMS** Autonomous driving, autonomous vehicles, end-to-end models, deep learning, motion planning, CARLA.

## I. INTRODUCTION

The first autonomous driving methods can be traced back to the 1980s [1], [2], and over the years, the field has continued evolving, attracting significant interest and research from both industry and academia [3]. Autonomous Vehicles (AVs) have the potential to address major issues such as road congestion and traffic-related accidents [4], with the latter being the leading cause of deaths for children and young adults aged between 5-29 years according to the World Health Organization [5].

Classical autonomous driving systems decompose the autonomous driving task into smaller modules such as perception, localization, planning, and control. The modules are solved and optimized independently before being incorporated into the final autonomous driving system. On the other hand, end-to-end models directly map sensory inputs to

the appropriate controls by optimizing the entire autonomous driving process as a single task, often through neural networks and deep learning. Recent advancements in deep learning have also driven increased interest in end-to-end models, and this approach is the central topic discussed in this survey.

Since safety is of the utmost importance for AV's commercial viability, AVs have to go through rigorous tests in both real-world and simulation environments before deployment [6]. While real-world tests are the most straightforward way to develop, verify, and validate AD systems, they can be costly due to long manual driving hours for dataset collection, potential accidents, and vehicle and sensor costs. Therefore, simulation environments emerge as a popular solution in AV research [7] to alleviate those problems.

The CARLA (Car Learning to Act) simulator [8] is a high-fidelity simulator with a flexible API, a high potential for environment and agent customizability, and a diverse range of sensor configurations. It is a popular tool in end-to-end autonomous driving research due to its

The associate editor coordinating the review of this manuscript and approving it for publication was Zheng Chen<sup>1</sup>.

simplicity in collecting data, training models, and evaluating their performance using official and community-created benchmarks.

In this paper, we choose to limit our review of end-to-end autonomous driving models to those that have used the CARLA simulator. This decision is mainly motivated by two factors. First, we aim to compare a wide range of methods within a common framework. CARLA, widely adopted since its release in 2017, provides a substantial body of literature, making it suitable for evaluating the evolution of end-to-end driving over a significant period of time. Second, since this survey focuses on end-to-end driving, high-fidelity sensory data from one end, and sufficiently accurate vehicle dynamics for control on the other end, are essential components for a testing environment. While benchmarks such as nuScenes [9], nuPlan open-loop [10], and VISTA [11] provide photo-realistic sensory data from real-world logs compared to CARLA's rendered environment, output commands do not alter the next input state, or only affect it in limited capacity using novel view synthesis. In terms of control and vehicle dynamics, closed-loop benchmarks such as CommonRoad [12], nuPlan closed-loop [10], WayMax [13], and Nocturne [14] can offer more realistic traffic behavior using data-driven scenarios from real-world logs, compared to CARLA's model-driven traffic behavior. However, these benchmarks use vectorized/abstracted inputs, making them more suited to mid-to-end, rather than fully end-to-end, evaluations. In summary, we focus on CARLA for this review because it satisfies both the need for an extensive body of research and a closed-loop system that provides high-fidelity sensory inputs, along with decent vehicle dynamics and traffic behavior for end-to-end evaluations. Additionally, we focus on goal-driven end-to-end driving, i.e., when there is a pre-defined start-to-end route to follow. We highlight the general weaknesses of end-to-end models and demonstrate how implementations address these weaknesses through various model inputs, outputs, architectures, and training paradigms.

## A. RELATED WORKS

Several related surveys review end-to-end driving, with the differences mainly in the scope or area of focus.

Chib and Singh [15] give a global overview of end-to-end autonomous driving, examining methodologies, sensory inputs, model outputs, learning approaches, evaluations, explainability, and safety aspects. Teng et al. [16] study both classical pipeline motion planning as well as end-to-end driving using Imitation Learning (IL), Reinforcement Learning (RL), and confusion learning approaches. Chen et al. [17] present the end-to-end driving history and roadmap, methodologies, critical challenges faced, as well as future trends. Coelho and Oliveira [18] review end-to-end autonomous driving under the CARLA simulator, and provide comparisons based on two of its benchmarks (CoRL [8] and NoCrash [19]). Kiran et al. [20] mainly review deep

reinforcement learning approaches for autonomous driving and briefly explore other learning approaches. Hagedorn et al. [21] review both motion planning and prediction (motion forecasting of other vehicles), and the various architectures used to incorporate both tasks. Yang et al. [22] review the application of large language models for autonomous driving.

## B. CONTRIBUTIONS

This survey shares similarities with a few previous works. We follow a structure that is similar to [15], where we dedicate sections to each end-to-end model's components, such as inputs, outputs, and training. We focus on reviewing *CARLA-based* autonomous driving state-of-the-art techniques, similar to [18]. However, our review additionally covers recent up-to-date methods, includes expert "privileged" autonomous driving models used for data collection, and reviews more benchmarks. The main contributions of this survey are as follows:

- We focus on CARLA-based end-to-end autonomous driving methods, reviewing their model inputs, outputs, architectures, and training. To provide a global, comprehensive overview of the reviewed end-to-end methods, we compile a detailed table describing their associated inputs, architectures, outputs, training approaches, and the benchmarks used.
- We explore both modular, rule-based, and end-to-end learning-based expert autonomous driving models that are used for data collection.
- We explore several CARLA benchmarks and present evaluations from various papers.
- We highlight current weaknesses and suggest future avenues that can be potentially worth exploring.

## C. SURVEY STRUCTURE

The survey is divided into three main parts. The first part serves as a preliminary overview. After giving a general introduction of the survey in Section I, it provides background information on autonomous driving, motion planning, and end-to-end models in Section II, and describes how such models are evaluated using benchmarks that are presented in Section III. Section IV then highlights end-to-end models' weaknesses, concluding the first part and setting the stage for part two, where these weaknesses are addressed by various methods. Equipped with the relevant background from part one, the second part forms the heart of the survey and showcases CARLA end-to-end driving methods by exploring model inputs in Section V, outputs in Section VI, architectures in Section VII, and training in Section VIII. Finally, the last part presents the evaluations and discussions, challenges and future works, and the conclusion in Sections IX, X, XI, respectively.

## II. BACKGROUND

In this section, we provide a brief background on the concepts and terms that will be used throughout this paper. We begin

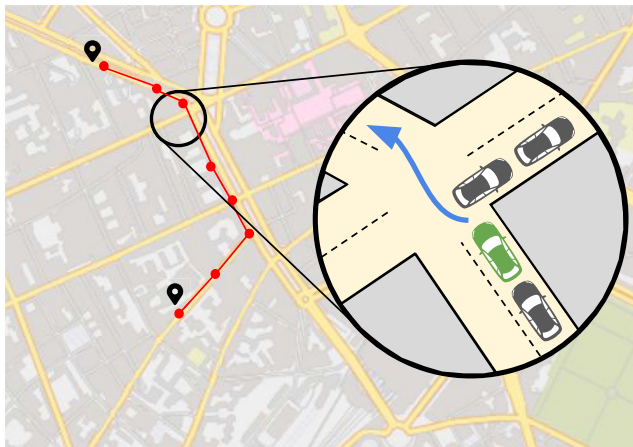
by presenting the autonomous driving motion planning context, and afterwards explain how modular vs. end-to-end autonomous driving systems differ when tackling this task. The latter is further elaborated by explaining reinforcement learning and imitation learning. Finally, we give a brief overview of the CARLA simulator itself.

### A. AUTONOMOUS DRIVING MOTION PLANNING

Planning in autonomous driving is primarily done in two stages. First, a high-level path planner plans a route from a starting point to a final destination. Afterwards, a low-level motion planner generates trajectories to guide the vehicle along the given high-level route (Fig. 1). This survey focuses on methods addressing the latter; the high-level route is typically given as part of the input.

In the context of autonomous driving, high-level path planning, also called route planning, solves shortest path problems in graph or node-like structures, using optimal solution algorithms such as Dijkstra [23] and A\* [24], among other algorithms [25]. This involves finding the best roads to follow in a city's road network to arrive from a starting location to a target destination as fast as possible, possibly taking into account factors such as traffic congestion, speed limits, road lanes, etc.

Low-level motion planners generate dynamically feasible trajectories to follow the high-level route [26]. Compared to high-level planners, the trajectories they generate are much shorter and are constantly updated to react to the continuously changing environment and surroundings [27]. Safety, obstacle avoidance, and traffic rules constraints are necessary to generate valid routes for urban driving. Passenger comfort and energy usage can also be included in the set of constraints, although they are typically not considered in CARLA implementations. This is due to the fact that low-level motion planning has not yet been solved with a 100% success rate at the time of writing [28].



**FIGURE 1.** High-level planning (red) is responsible for planning a route from start to finish based on a city network graph. Low-level planning is responsible for generating local trajectories (blue) for the ego-vehicle (green) in order to follow the high-level topological route while avoiding obstacles and respecting traffic laws.

### B. MODULAR VS END-TO-END AUTONOMOUS DRIVING

The low-level trajectories and output controls in autonomous driving can either be generated by planning and control modules that are part of a bigger modular pipeline, or they can be generated in an end-to-end manner using deep learning models.

Modular pipelines [29] decompose autonomous driving into multiple modules of perception, localization and mapping, planning and prediction, and finally, control (see Fig. 2, top). The perception module takes raw sensory input and processes it to understand the environment and the objects in the scene. It is responsible for perceiving, detecting, and tracking objects such as other vehicles, pedestrians, road lanes, traffic signs, etc. This information is then passed to the localization and mapping module to map the scene and localize the ego-vehicle in it. The output is then passed on to the planning and prediction module. It generates appropriate trajectories based on the perceived environment and the predicted motion of other actors in the scene. Finally, the control module outputs the appropriate steering, throttle, and break actions to follow the generated trajectory.

Under the modular pipeline, each module can be engineered independently, allowing engineers to customize each component separately without having to revamp the whole system. This also has the added benefit of a simpler debugging process during failure, and improved interpretability. However, the disadvantages lie in the fact that developing each component requires specialized expertise and considerable manual design and parameter tuning. In addition, since the modules are optimized separately, errors in one module can accumulate and propagate throughout the software stack, which can lead to cascading failure.

End-to-end systems, on the other hand, are optimized as a whole towards the final autonomous driving output (see Fig. 2, bottom). They receive sensory inputs and output the appropriate actions and/or trajectories using a neural network. As a result, end-to-end models require less development effort and specialized knowledge for each module. However, their reliance on data can make them susceptible to failure when encountering unseen scenarios. Compared to modular pipelines, they are also harder to debug due to the black-box nature of deep learning. The weaknesses of end-to-end models are further detailed in Section IV.

End-to-end learning is split into two categories based on the learning paradigm that is used to train the model. Reinforcement Learning (RL) trains a model through direct interactions with the environment and trial and error techniques. Imitation Learning (IL), on the other hand, trains the model under supervised learning using demonstrations from a dataset.

### C. REINFORCEMENT LEARNING

Reinforcement learning (RL) [30] is a machine learning paradigm where an agent learns a policy by interacting with an environment through trial and error. The formulation of

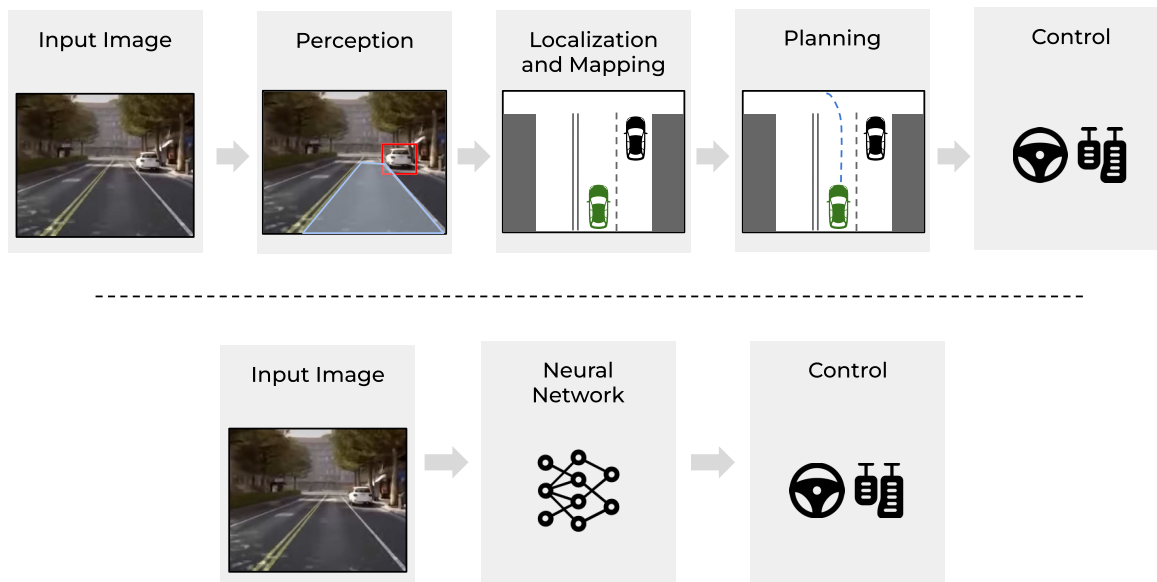


FIGURE 2. Comparison between modular (top) and end-to-end architectures (bottom).

RL can be understood through the framework of a Markov Decision Process (MDP), which is defined by a set of states  $S$ , a set of actions  $A$ , a transition function  $P(s'|s, a)$  which determines the probability of transitioning from state  $s$  to state  $s'$  given action  $a$ , and a reward function  $R(s, a)$  that assigns rewards based on the state and action taken. The objective in an MDP is to find a policy  $\pi(s)$ , which maps states to actions, that maximizes the expected cumulative reward. A policy  $\pi$  is a strategy or a rule that the agent follows to choose actions based on the current state. It can be deterministic, where a specific action is chosen for each state, or stochastic, where actions are chosen according to a probability distribution.

In RL, the cumulative reward is often referred to as the return, denoted as  $G_t$ , and is defined as the sum of discounted rewards from time step  $t$  onwards. The discount factor  $\gamma$  (with  $0 \leq \gamma < 1$ ) reduces the value of future rewards, making rewards received at immediate time steps more valuable. Additionally, it ensures the convergence of the infinite series of future rewards, preventing the return from exploding to infinity and thus keeping it finite.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad (1)$$

The goal is to find an optimal policy  $\pi^*$  that maximizes the expected return  $\mathbb{E}[G_t|s_t, a_t, \pi]$  for each state  $s$  and action  $a$ . By iterating over different policies and updating them based on observed returns, the agent learns to choose actions that maximize the expected cumulative reward over time.

The challenge in RL is to design reward functions and models that can effectively learn to produce desirable driving behavior. For example, a positive reward can be given to the agent for actions that lead to progress towards the destination, maintaining a safe speed, or complying with traffic laws.

On the other hand, a negative reward can be assigned for actions that result in traffic infractions, such as running a red light, speeding, or causing a collision. By attempting to maximize the expected reward, the RL model might learn to drive in a manner that is both safe and efficient. However, the explicit design of rewards does not always lead to the expected behavior, as the agent may find unintended ways to maximize rewards. Therefore, careful consideration and refinement of reward functions are essential to ensure the RL agent learns the desired behaviors and avoids exploiting loopholes in the reward structure.

#### D. IMITATION LEARNING

Imitation Learning (IL), also sometimes called Learning from Demonstrations (LfD), is a machine learning technique used to teach an agent to learn to perform a task autonomously by mimicking expert demonstrations, usually performed by humans. Autonomous agents train under a supervised learning setting to learn an appropriate behavioral policy  $\pi$ , using a dataset of expert demonstrations  $\mathcal{D}$ , which is comprised of state-action pairs  $(s, a)$  generated by an expert policy  $\pi^*$ . Demonstrations can follow a trajectory-based representation [31] to account for the sequential nature of some tasks, where a trajectory  $\zeta$  is comprised of a sequence of state action pairs that span over a horizon  $t$ ,  $\zeta = \{(s_0, a_0), (s_1, a_1) \dots (s_t, a_t)\}$ . In end-to-end IL applied to AVs, states  $s$  typically correspond to sensor-readings, such as images from RGB camera or point clouds from LiDARs, and actions correspond to a trajectory composed of future waypoints to follow, or direct steering angle and brake/throttle values.

Imitation learning can be broken down into two categories, Behavior Cloning (BC) or Inverse Reinforcement Learning



(IRL). Learning a behavioral policy using BC boils down to learning a direct mapping between states to actions by training on the demonstration dataset  $\mathcal{D}$ . On the other hand, IRL attempts to understand what motivates an expert to exhibit the demonstrated behaviors by learning an underlying reward function that is maximized under expert demonstrations. Both types of IL are further explained below.

### 1) BEHAVIOR CLONING

Behavior cloning algorithms learn a policy  $\pi$  that minimizes the error when mapping from the features to the labels of  $\mathcal{D}$ . Under the state-action representation, this equates to minimizing the loss between the predicted action given a state  $\pi_\theta(s)$  and the ground-truth action  $a$  in  $\mathcal{D}$ .

$$\operatorname{argmin}_{\theta} \mathbb{E}_{\mathcal{D} \sim (s,a)} [\mathcal{L}(\pi_\theta(s), a)] \quad (2)$$

### 2) INVERSE REINFORCEMENT LEARNING

Inverse Reinforcement Learning (IRL) algorithms learn to derive a reward function  $R^*$  from observed expert demonstrations that makes the expert's behavior  $\pi^*$  appear optimal [32].

Inferring the reward function can be done in several ways. Maximum margin methods [33] find a reward function that maximizes the margin between the reward obtained from the demonstrated expert policy and other policies. Maximum causal entropy methods [34] employ a probabilistic approach to find a reward distribution with maximal entropy. This approach can handle ambiguities when learning from situations where multiple decisions lead to similar outcomes. By maximizing entropy, these methods reduce bias towards any specific policy and maintain a more balanced and generalized reward distribution. Generative Adversarial Imitation Learning (GAIL) [35] methods employ adversarial techniques to generate a policy that is indistinguishable from the expert policy. GAIL follows the Generative Adversarial Networks (GAN) formulation [36], where two networks—a generator and a discriminator—are trained simultaneously with adversarial techniques. The generator attempts to create policies that are indistinguishable from expert policies, and the discriminator attempts to learn how to distinguish between them. Therefore in the case of GAIL, the “learned reward” is how well generated policy can fool the discriminator.

## E. THE CARLA SIMULATOR

CARLA [8] is a high-fidelity simulator with a flexible API that allows modification and access to various autonomous driving-related factors. It can modify weather and lighting conditions, generate maps, simulate actors such as pedestrians, vehicles, and traffic lights, and create custom driving scenarios. It also offers a scalable multi-client server architecture which can accelerate deep learning and dataset collection through parallelization. Most importantly, CARLA can simulate various sensor data as well as vehicle dynamics based on steering and throttle/brake inputs.

CARLA offers 12 publicly available pre-built maps, called Towns, featuring different road types and environments for training and testing. This is particularly useful for evaluations and benchmarks, where identical conditions need to be established for fair comparisons. Several benchmarks, whether official or community-created, have been made publicly available for use to evaluate end-to-end methods. We explore the CARLA towns and benchmarks in more detail in Section III.

Since the ground-truth environment and agent states can be accessed at will using the CARLA API, a distinction has to be stated between “privileged” and “sensorimotor” models to avoid confusion. Privileged autonomous driving models operate under a perfect perception assumption and have access to ground-truth states such as other vehicles locations, lane boundaries, traffic light states, etc. This allows implementations to focus solely on solving how to act, without having to tackle solving how to see. On the other hand, sensorimotor models can only sense the environment through sensor readings, and are tasked to solve both seeing and acting. Sensorimotor models are thus considered to be closely related to real and physical autonomous driving experiments since they do not have access to the ground truth. In fact, a few sensorimotor models in CARLA [37], [38], [39], [40], [41], [42], [43] have also been evaluated using real sensory data in nuScenes' [9] open-loop tests or using closed-loop tests in real-world environments by deploying miniature cars [44], [45].

## III. BENCHMARKS

Benchmarks refer to evaluation protocols that allow different driving methods to be compared on a common ground. Over the years, several CARLA benchmarks, both official and community-created, have been established to evaluate autonomous driving methods' performance. The benchmarks expose driving models to a wide variety of towns (Table 1),<sup>1</sup> routes, weather and daylight conditions, as well as scenarios to evaluate their generalization capacity. In this section, we explore several benchmarks that evaluate urban driving methods' performance in navigating from a start to an end location. We also briefly cover benchmarks that assess more specific aspects of urban driving, such as occlusion scenarios.

### A. EARLY BENCHMARKS

#### 1) CORL

The first official benchmark was established alongside CARLA's release paper [8], and it is commonly called CoRL in the literature, following the name of the conference where the paper was published. It evaluates driving agents on a set of four increasingly difficult goal-directed navigation driving tasks, where the goal is to drive from point A to point B given a high-level topological route in a town. The first task is to drive on a straight route with no other dynamic agents, and

<sup>1</sup>Information regarding CARLA towns has been obtained from the publicly online documentation <https://carla.readthedocs.io/en/latest/>

**TABLE 1. CARLA towns.**

Town	Description
Town01	A small, simple town with a river and several bridges.
Town02	A small simple town with a mixture of residential and commercial buildings.
Town03	A larger, urban map with a roundabout and large junctions.
Town04	A small town embedded in the mountains with a special "figure of 8" infinite highway.
Town05	Squared-grid town with cross junctions and a bridge. It has multiple lanes per direction. Useful to perform lane changes.
Town06	Long highways with many lanes, entrances, and exits. It also has a Michigan left.
Town07	A rural environment with narrow roads, corn, barns and hardly any traffic lights.
Town08	Secret "unseen" town used for the Leaderboardv1 challenge
Town09	Secret "unseen" town used for the Leaderboardv1 challenge
Town10	A downtown urban environment with skyscrapers, residential buildings and an ocean promenade.
Town11	A large map that is undecorated. Serves as a proof of concept for the Large Maps feature.
Town12	A Large Map with numerous different regions, including high-rise, residential and rural environments.
Town13	A Large Map that shares a similar scale with Town12, but with different road surfaces, buildings and vegetation.
Town14	Secret "unseen" Large Map used for the Leaderboardv2 challenge.
Town15	A map that is based on the road layout of Universitat Autònoma de Barcelona.

the second adds one turn to the route. The third task involves driving on a route that can have any starting and end point in the town, meaning that the driving agent can encounter multiple turns and intersections. The fourth task additionally incorporates pedestrians and cars as dynamic obstacles to the route. Methods are evaluated by measuring the success rate of multiple episodes, where an episode is considered successful if an agent completes a route within a defined time limit. Speed limits and traffic lights can be violated, and such infractions are not taken into account during evaluation. Town01 is used for training, while Town02 is used for tests. In addition, the weather and daylight conditions used are different for the training and testing phases.

## 2) NOCRASH

The NoCrash benchmark [19] builds upon CoRL's Navigation tasks (third and fourth tasks) with similar routes (Fig. 3) and three degrees of traffic densities. Contrary to CoRL, NoCrash considers episodes with a collision above a certain magnitude as unsuccessful. They similarly use Town01 for training and Town 02 for testing.

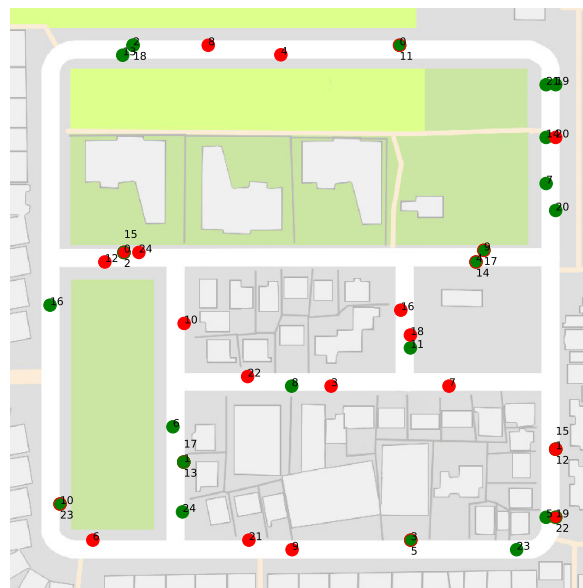
## 3) ANYWEATHER

The AnyWeather benchmark [46] follows the CoRL benchmark but with an increased amount of testing weather and

daylight conditions to further examine the generalization capability of autonomous driving methods.

## 4) TRAFFIC-SCHOOL

The traffic-school benchmark [47] builds upon NoCrash but only considers routes with no overtime, no crashes, no traffic light violations, and no lane deviation infractions as successful.



**FIGURE 3. Town02 start (red) and goal (green) locations of routes for the NoCrash benchmark [19].**

## B. LEADERBOARDV1

As end-to-end driving methods have developed and become more advanced, the original official benchmark, CoRL, has been nearly solved [48], [49]. The leaderboardv1 [28] was later introduced as the next official benchmark, and it evaluates urban driving agents on a set of longer, more complex routes that include challenging scenarios. The offline leaderboardv1 offers 76 publicly available pre-defined routes, split into 50 for training, and 26 for testing. Training routes are done using Towns 01, 03, 04, and 06, whereas testing is done using Towns 02, 04, and 05. A hidden set of routes based on Towns 08 and 09 are used for online server-side evaluations that are inaccessible to users to ensure fair tests. The hidden routes are based on 10 routes, with tests done on two conditions, and with five repetitions. This results in a total of  $10 \times 2 \times 5 = 100$  routes.

There are 10 possible traffic scenarios that can be encountered in leaderboard v1 routes, and driving agents have to cope safely with the encounters without crashing. Those scenarios have been chosen from the NHTSA (National Highway Traffic Safety Administration) pre-crash typology [50], which is a report that consists of a database of several police-reported pre-crash scenarios involving at least one vehicle. The scenarios depict vehicle movements and dynamics, and are translated in CARLA by using the

**TABLE 2. Leaderboard v1 scenarios.**

Scenario Number	Description
01	<i>Control Loss</i> : The ego-vehicle loses control due to bad conditions on the road and it must recover, coming back to its original lane.
02	<i>Leading vehicle sudden brake</i> : The ego-vehicle must perform an emergency brake or an avoidance maneuver.
03	<i>Obstacle avoidance without prior action</i> : The ego-vehicle encounters an obstacle and must perform an emergency brake or an avoidance maneuver.
04	<i>Obstacle avoidance with prior action</i> : While performing a maneuver, the ego-vehicle encounters an obstacle on the road and must perform an emergency brake or an avoidance maneuver
05	<i>Slow leading vehicle</i> : The ego-vehicle must perform a lane change to evade a leading vehicle that is moving too slowly.
06	<i>Blocking object</i> : The ego-vehicle must go around a blocking object using the opposite lane yielding to oncoming traffic.
07	<i>Intersection red light violation</i> : 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.
08	<i>Unprotected left turn at intersection</i> : The ego-vehicle is performing an unprotected left turn at an intersection, yielding to oncoming traffic.
09	<i>Right turn at an intersection with crossing traffic</i> : The ego-vehicle is performing a right turn at an intersection, yielding to crossing traffic.
10	<i>Crossing negotiation at an unsignalized intersection</i> : The ego-vehicle needs to negotiate with other vehicles to cross an unsignalized intersection. In this situation it is assumed that the first to enter the intersection has priority.

simulator to implement events such as spawning vehicles, suddenly emerging pedestrians, or loss of control by adding noise to steering control. The list of leaderboard v1 traffic scenarios is shown in Table 2 for reference, since the traffic scenarios found on the leaderboard site at the time of writing depict the newer version's scenarios (leaderboardv2).

Leaderboard and leaderboard-inspired benchmarks mainly use the Driving Score (DS) metric to evaluate their agents' driving performance for a given route  $i$ , with 100% (or 1) being the highest possible score, and 0 the lowest.

*Driving Score*: The Driving Score (DS) metric is calculated using the product of the Route Completion (RC) and Infraction Penalty (IP) metrics.

$$DS_i = RC_i \cdot IP_i \quad (3)$$

*Route Completion*: For a given route, the Route Completion (RC) metric is calculated by taking the percentage of the distance completed by the agent. A run in a route can end prematurely before completion if the agent deviates from the assigned route significantly, is blocked and does not take actions for a while, or takes too long to finish the route.

*Infraction Penalty*: The Infraction Penalty (IP) metric keeps track of how many infractions are committed along a given route  $i$ , starting with a value of 1 (100%), and decreasing each time an infraction is committed by multiplying

with the infraction's weighting factor. Different types of infractions have different weighting factors, summarized in Table 3. For example, if an agent runs a red light (0.7) and also collides with a pedestrian (0.5), the infraction penalty amounts to  $1 \cdot 0.7 \cdot 0.5 = 0.35$  for the route being evaluated. Thus, for a given route, the IP is calculated as the product of all infraction instances committed for each type of infraction.

$$IP_i = \prod_j^{\text{ped., ..., stop}_j} (p_j)^{\#\text{infractions}_j} \quad (4)$$

Finally, to obtain the final driving score DS over an entire benchmark, the driving scores  $DS_i$  for each route  $i$  in  $N$  total routes are simply averaged.

$$DS = \frac{1}{N} \sum_{i=0}^N DS_i = \frac{1}{N} \sum_{i=0}^N RC_i \cdot IP_i \quad (5)$$

The evaluations are done either under sensor or map tracks, the latter of which allows the possibility of using an HDMap in addition to sensors.

**TABLE 3. Infraction penalty coefficients. Infractions marked with "\*" are only applicable for leaderboardv2 evaluations.**

Infraction Type	Penalty coefficient
Collisions with pedestrians	0.50
Collisions with other vehicles	0.60
Collisions with static elements	0.65
Running a red light	0.70
Running a stop sign	0.80
Off-road driving	1 - % off-road
Scenario timeout*	0.7
Failure to maintain minimum speed*	0.7
Failure to yield to emergency vehicle*	0.7

### C. LEADERBOARDV1 VARIANTS

Since leaderboard evaluations are done server-side on a set of routes that are hidden from submitting users, conducting ablation studies or simply visualizing the ego-vehicle for qualitative evaluation is not possible. As a result, multiple "leaderboard-inspired" benchmarks were created using the same metrics in order to circumvent the inaccessibility problem.

#### 1) LAV

The LAV benchmark [51] uses four towns for training (Towns 01,03,04, and 06) and two unseen towns for evaluation (Towns 02 and 05). Four routes are used for evaluations, two from each town. The routes are evaluated under four different weathers, and each route is run three times to calculate mean and standard deviation values. Therefore, the LAV



benchmark uses 4 (routes) \* 4 (weather) \* 3 runs = 48 runs to evaluate driving models.

## 2) NEAT

The NEAT benchmark [52] uses eight towns for training (Towns 01-07 and 10) and six towns for evaluations (Towns 01-06). Each town has two unique routes, with the exception of towns 03 and 04, which contain three unique routes. This totals to 14 unique routes (4 towns \* 2 + 2 towns \* 3). Each unique route is repeated three times, but with new daylight and weather conditions, totalling 42 runs and 42 different daylight and weather combinations. In addition, the NEAT benchmark's routes explicitly incorporate more turns when compared to the leaderboard routes to tackle the heavy bias for straight driving.

## 3) TOWN05

The Town05 benchmark [53] uses seven towns for training (Towns 01-07, and 10), and reserves Town05 for evaluation. The benchmark has two evaluation modes. Town05 Short contains short routes of 100-500m comprising three intersections each. Town05 Long 10 contains long routes of 1000-2000m comprising 10 intersections each. The same weather condition is used for tests.

## 4) LONGEST6

The Longest6 benchmark [54] also uses the same 8 towns for training (Towns 01-07, and 10) and 6 towns for evaluations (Towns 01-06). Among leaderboard1's 76 routes, only the six longest routes per town are used for evaluation. This results in 36 routes (six routes for six towns), with each containing a unique daylight and weather combination similar to the NEAT benchmark. The Longest6 benchmark additionally ensures a higher density of dynamic agents when compared with leaderboard1.

## D. LEADERBOARDV2

Leaderboardv2 extends its predecessor with longer routes and more complex scenarios. As explained by Li et al. [55], leaderboard1 scenarios can be handled with simple skills such as lane following, adherence to traffic signs, and collision avoidance. However, leaderboardv2 scenarios require much more advanced behavior due to their complexity. For example, an incoming vehicle can invade the ego vehicle's lane and the ego vehicle is required to sidestep and avoid collision. Another example scenario involves yielding and making way to an emergency vehicle. Compared to leaderboard1's routes, which average roughly 1km in length, leaderboardv2's routes can extend up to 10km. The offline leaderboardv2 offers 90 training routes and 20 testing routes, based on Towns 12 and 13 respectively. The online server-side hidden routes are evaluated in Town14.

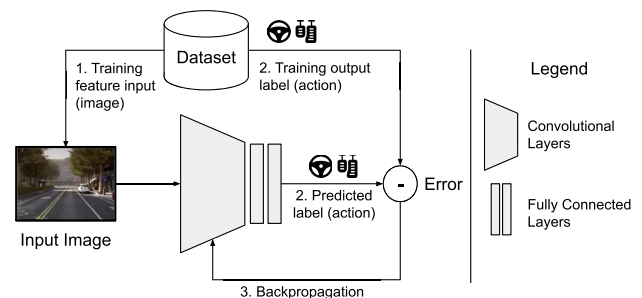
## E. SCENARIO-SPECIFIC BENCHMARKS

A few benchmarks have been developed to evaluate methods on individual scenarios. The Driving in Occlusion Simulation benchmark (DOS) [56] evaluates driving agents in four

occlusion scenarios. The CornerCaseRepository [55] tests driving agents under all 39 scenarios from leaderboardv2. However, contrary to the leaderboard routes, each route contains only one scenario to evaluate driving performance on each specific scenario. The Bench2Drive benchmark [57] similarly tests driving models to handle 44 different individual scenarios.

## IV. END-TO-END MODELS' WEAKNESSES

A straightforward implementation of an end-to-end driving model maps raw image input pixels to output steering commands [2], [58], [59]. As shown in Fig. 4, during the training phase, the model 1) receives an input image from the dataset, 2) predicts the output steering, throttle, and brake, and calculates the loss with respect to the ground truth, and 3) updates its weights through backpropagation. During the testing phase, the model simply receives an image input and predicts the corresponding output controls. While such approaches can yield decent results for basic tasks such as lane keeping, they struggle to handle more complex situations that commonly arise during urban driving due to several weaknesses.



**FIGURE 4.** An example of an end-to-end model inspired by Bojarski et al. [59]. It takes image inputs and outputs steering, throttle, and brake values.

## A. WEAKNESSES OF END-TO-END MODELS

### 1) DISTRIBUTION SHIFT

End-to-end models are purely data-driven and heavily influenced by the data seen during training. While they can infer appropriate actions in situations that are similar to what they have previously experienced, they can fail dramatically when encountering entirely new scenarios. This phenomenon, known as the distribution shift, occurs when there is a difference between the distributions of data encountered during training and testing. Consider an end-to-end model driving on a lane in ideal starting conditions: it is perfectly centered and aligned with the lane. As it begins to drive, it produces tiny errors, since neural networks are inherently statistical and not perfectly accurate. Over time, these errors accumulate, causing the vehicle to drift off the lane. Since the training data was collected mostly from perfect human demonstrations, there is no data that the model has seen that shows how to recover from such errors. As a result, the model fails to correct itself and veers off the lane.

## 2) LONG-TAILED DISTRIBUTION

A significant issue related to the distribution shift in autonomous driving is the long-tailed nature of the data distribution. Most training data predominantly feature routine and straightforward driving scenarios, such as driving straight or following a lane, because these actions are the most common in everyday driving. In contrast, safety-critical situations are far less common but are crucial for ensuring the safety and reliability of autonomous vehicles. Examples of these scenarios can include a car suddenly running a traffic light, a pedestrian unexpectedly crossing the street, or the sudden appearance of an animal like a deer. Such incidents require immediate and appropriate responses to avoid accidents. However, due to their rarity, these events are underrepresented or not seen at all in the datasets used for training autonomous driving models. This imbalance results in end-to-end models that can handle routine tasks efficiently, but may struggle or fail to act appropriately in safety-critical situations.

## 3) CAUSAL CONFUSION

Causal confusion [60] occurs in end-to-end models when they learn to associate irrelevant features with driving actions due to unintended correlations found in the training data. For example, a model might learn to stop at a traffic light because it correlates stopping with the presence of other stationary vehicles in its vicinity, rather than with the traffic light status itself. This happens because the vehicles generally occupy a much larger portion of the input image pixels compared to the traffic light. If those vehicles are absent in a similar future scenario, the model might fail to stop, leading to unsafe driving behavior.

## 4) INTERPRETABILITY

Interpretability is an essential component not only for end-to-end autonomous driving, but also for the deep learning community as a whole. Understanding the reasoning behind a model's output is essential for both debugging purposes and building trust in the system. It is challenging to understand why a car might output a specific steering command at a given moment. Without understanding the underlying decision-making process, it is difficult to determine the short-term maneuver the vehicle is attempting to execute from just the image-action pair at the observed moment. This lack of transparency makes it harder to diagnose issues and ensure the model's actions are safe and reliable.

## B. SOLUTIONS

Several solutions have been proposed to address the various shortcomings of end-to-end autonomous driving models, not limited to the aforementioned issues but also including others that will be discussed in the following sections. These sections explain how CARLA-based implementations tackle these challenges through various end-to-end model inputs, outputs, architectures, and training practices. An overview is provided in Table 6.

## V. MODEL INPUTS

As previously mentioned, a simple input structure for an end-to-end model involves using raw sensor data only, such as pixels from an image frame. The pixels are processed and encoded using fully connected layers [2] or convolutional layers [59]. Recent end-to-end models incorporate additional inputs to enhance the model's driving and comprehension of its surroundings. Such inputs may include additional sensors (e.g., LiDAR), goal conditioning, or sequences of past values (e.g., sequence of past images). This section aims to explain the various types of inputs that were adopted by state-of-the-art models and how they are processed.

### A. PERCEPTION SENSORS AND ENCODERS

#### 1) CAMERAS

The camera is the most commonly used sensor in autonomous driving due to its ability to capture rich contextual information of the scene by perceiving the colors, shapes, and textures of objects in its surroundings. Without cameras, information such as traffic light states, road signs, and lane markings would be challenging to interpret accurately, making them crucial for comprehensive scene understanding.

Most CARLA implementations encode their input images using Convolutional Neural Networks (CNNs) to extract relevant features (see Table 6), although recent methods began to adopt Vision Transformers (ViTs) [37], [38], [39] or Visual Language Models (VLMs) [61], [62] encoders instead. ViTs [63] encoders use transformer architectures (see Section VII-B) to treat input image patches as tokens, whereas VLMs encoders [64], [65] leverage internet-scale image-text pairs to learn rich multimodal representations, similar to how large language models exploit vast amounts of textual data.

End-to-end models either directly use the raw pixel data as input to an encoder to extract relevant features, or pre-process them beforehand. Pre-processing raw pixels can potentially simplify end-to-end training by removing noise or driving irrelevant information inputs. Reinforcement learning end-to-end models pre-process their input by pre-training a visual encoder using supervised learning to learn how to abstract the scene before beginning the RL phase for motion planning [49], [66], [67], [68]. Scene abstraction can also be done through semantic segmentation. By pre-training a visual encoder for semantic segmentation, the resulting segmented pixels can be used by the end-to-end driving model, with a more informative input for the training process [44], [69], [70]. Another option involves using object instances as inputs instead of dense pixel representations [71], [72]. Object instances can represent objects such as other vehicles, and can contain details such as their position and orientation with respect to the ego-vehicle.

#### 2) LIDARS

LiDAR (Light Detection and Ranging) sensors generate 3D point clouds by emitting laser pulses and measuring

the time of flight and the angle of the reflected pulses from the perceived objects. They are adopted as inputs by several models, usually alongside cameras, to boost spatial comprehension through accurate depth and ranging values. Some methods use LiDARs only without cameras as input [43], [73], and work under the assumption that information such as traffic light states is known.

LiDAR 3D point clouds are often processed before being input to end-to-end models to reduce their dimension, ensuring that the data is more manageable and only driving-relevant point clouds remain. A popular technique involves discretizing and projecting the 3D point cloud into a 2D Bird's Eye View (BEV) space using heuristic techniques [53], [54], [73], [74], [75], [76], [77]. Points on and above ground level are discretized into a 2-bin histogram, and only points within an arbitrary range around the ego-vehicle are conserved and divided into cells to obtain a 2D BEV grid with fixed resolution. For example, assuming the grid's resolution is arbitrarily set to  $256 \times 256$ , this technique "results in a two-channel pseudo-image with  $256 \times 256$  pixels" [53], i.e. of dimension  $\mathbb{R}^{256 \times 256 \times 2}$ . An alternative to heuristic techniques is using learning-based LiDAR encoders. The authors of [78], [79] use the SECOND (Sparsely Embedded Convolutional Detection) backbone encoder [80] to extract feature maps from point clouds discretized into voxels with VoxelNet [81] using sparse 3D-convolutions. References [56], [82] use PointPillars [83] to organize the point clouds into vertical "pillars" and encode them into 2D BEV space with simplified PointNets [84].

### 3) SENSOR FUSION

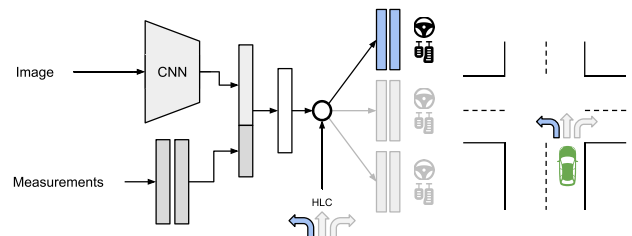
Sensor fusion integrates data from multiple sensors to allow end-to-end models to process multimodal input data under a more unified and coherent structure. Some implementations employ early fusion, where sensor fusion is implemented as a preliminary step, and afterwards use the fused sensory data as input to end-to-end models. Rosero et al. [85], [86] generate an image-based point cloud using a binocular setup and stereo-matching [87], and afterwards project it into BEV space and stack them with 2D projected LiDARs along the channel dimension. [51], [82] use image CenterPoint [88] and LiDAR PointPillars [83] perception backbones and fuse them using PointPainting [89] as a pre-trained perception module. Jia et al. [78] combine multi-camera inputs using the LSS (Lift, Splat, Shoot) approach [90] to transform multi-2D images into BEV space. They fuse the output with the encoded LiDAR using simple concatenation.

Other methods employ sensor fusion as part of the end-to-end model's learning process through middle fusion. Reference [53], [54], [75], [76], [91], [92] use transformer modules to combine intermediate features at multiple resolutions from multimodal input encoders to achieve sensor fusion. Reference [37], [52], [77], [93] perform sensor fusion by using transformers to process sensors after they have been fully encoded by feature extractors. Hu et al. [42] fuse

images from a four-camera setup using a perception module that leverages past ego motion and sensory inputs to achieve spatio-temporal fusion. Xiao et al. [94] explore both early and middle fusion schemes for RGB and depth channel images.

### B. GOAL CONDITIONING

When a car arrives at an intersection, it is presented with multiple equally viable choices: it can either turn left, right, or continue straight. This ambiguity is usually eliminated by the driver's intention, assuming a final destination point exists. However, the simple end-to-end example model shown earlier in Fig. 4 has no means to determine driving intentions through image pixels alone. During training, a different maneuver and action demonstration can exist for the exact same intersection input state (steering left or right can happen at the same intersection), causing the model to struggle to learn the correct action to take. As a result, goal conditioning is essential to eliminate the decision ambiguity. This is mainly done using High-level Commands (HLCs) and/or Target Points (TPs) in the literature.



**FIGURE 5. The Conditional Imitation Learning (CIL) [45] network architecture. It concatenates CNN-encoded features with measurement features and implements a high-level command switch afterwards. When a car is at the intersection, the decision ambiguity is eliminated by the given HLC ("go left" in this case).**

#### 1) HIGH-LEVEL COMMANDS

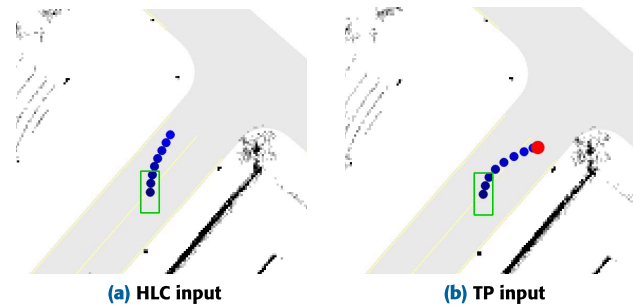
High-level commands indicate the high-level intention (follow lane or left, right, straight at intersection) for every input image frame. Codevilla et al. incorporate a switch mechanism that is conditioned based on the HLC input, and the switch is responsible for activating the corresponding dedicated sub-module (see Fig. 5). During training, each sub-module's Multilayer Perceptrons (MLPs) network parameters are trained separately based on the HLC, and thus become specialized in outputting the appropriate action based on the desired maneuver.

Earlier works mainly use 4 HLC outputs, which are go straight, left, or right at intersections, and follow lane. Later, the release of the official CARLA leaderboard [28] introduced left and right lane changes as additional HLCs.

#### 2) TARGET POINTS

Target Points (TPs) are sparse, noisy GPS signals provided by a high-level global planner that are sampled every 50-100 meters to indicate the desired path. Various methods that adopted the TP input saw a significant boost to the driving performance. This was later shown to be due to a

hidden bias in imitation learning models that exploits the rich geometric information provided by GPS signals, as explained by Jaeger et al. [92]. With the usage of TPs, imitation learning models gain the ability to periodically eliminate the accumulated errors caused by the distribution shift by correcting drift as they move towards each sparse GPS target point (Fig. 6).



**FIGURE 6.** End-to-end methods can correct drift by extrapolating their output towards target points' GPS locations. Image from [92].

Since GPS values provided by the leaderboard's TPs are noisy [54], filtering techniques have been adopted by some implementations to ensure that the GPS input is reliable. Chitta et al. [54] use a model-based denoising algorithm based on the kinematic bicycle model [95] and Monte Carlo estimates. Jaeger et al. [92] use an unscented Kalman filter [96] with Van der Merwe's scaled sigma point algorithm [97].

### C. OTHER INPUTS

#### 1) LANGUAGE

Multimodal Large Language Models (MLLMs) extend traditional LLMs with the ability to process and analyze various types of information beyond text only, such as image or audio [98]. These models demonstrate promising results in performing complex reasoning tasks that span multiple modalities and have seen significant interest across various domains, and autonomous driving is no exception [99]. MLLMs applied for autonomous driving have the potential to improve reasoning and facilitate interaction with driving models through language. They can perform more complex maneuvers using language instructions instead of relying only on goal conditioning inputs such as HLCs and TPs, as demonstrated by Wang et al.'s DriveMLM [39] CARLA implementation (see Fig. 7).

Shao et al. [100] similarly use navigation language instructions and human notice instructions as input to their end-to-end model. Mei et al. [62] utilize descriptions of critical objects using a VLM encoder as input to their model to improve understanding of the scene.

#### 2) VEHICLE STATE AND DYNAMICS

A commonly used input for end-to-end models is ego-vehicle state measurements, most notably speed (see Table 6). Other possible state measurement inputs include the current steering angle [49], [101], [102], steering angles generated by a pure pursuit algorithm [70], or previous actions [67].

Chen et al. [72] use an offline trajectory tree composed of Euler spirals as an input. This additional input ensures that generated trajectories are kinematically feasible.

#### 3) HD MAPS

HD Maps in CARLA are represented using OpenDRIVE files, which define road networks through detailed geometric descriptions, including lines, arcs and spirals to model road curvature. These files also include precise lane data, providing information such as lane width, boundaries, markings, and types. Additionally, OpenDRIVE files include information on nodes and junctions for connecting roads, as well as traffic signs and signals, to model intersection and traffic behavior in detail.

While HD-maps have been used almost exclusively by modular implementations in CARLA, Zhang et al. [75] have incorporated them as input in their end-to-end implementation alongside camera, LiDAR, and radar inputs. They experiment with two different HD-map representations, where OpenDRIVE files are either rasterized as an image from a BEV perspective and encoded using a CNN, or vectorized and encoded using VectorNet [103].

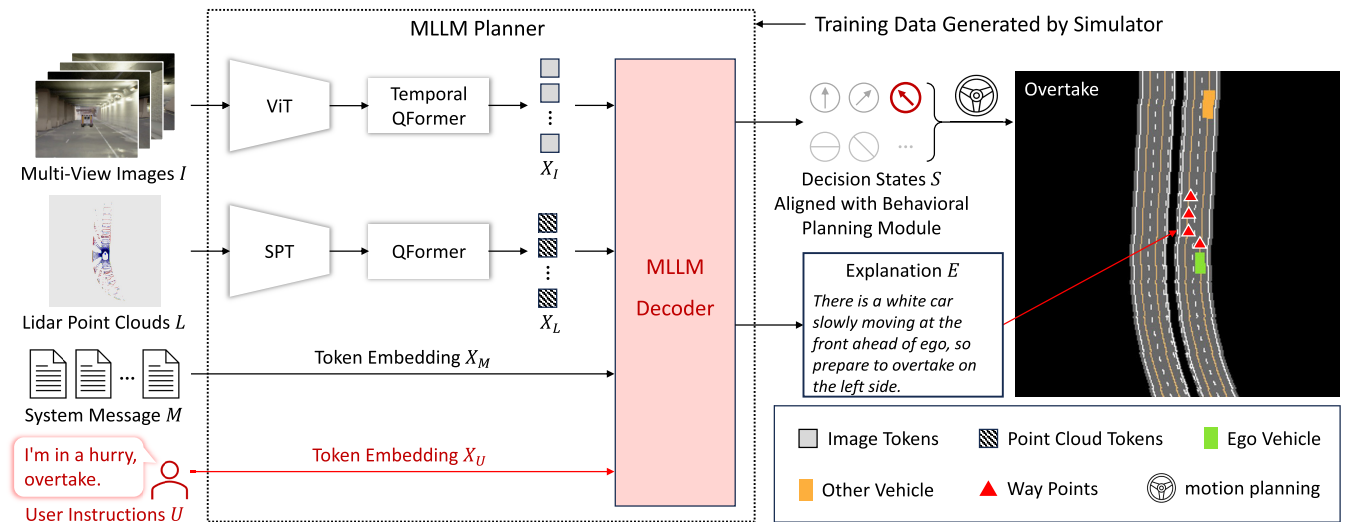
### D. SEQUENTIAL INPUTS

Incorporating sequential inputs to end-to-end models, such as past image frames, adds an additional temporal dimension to the system. The positive impact of historical input to end-to-end models has been inspected through ablation studies by a few implementations [42], [49], [56], [78], [104], although its impact varies from marginal to significant from one method to the other. On the other hand, some methods intentionally avoid using historical input [52], [54] due to the copycat problem [105], [106], and resort to using sensor inputs from the current timestep only. The copycat problem [105] falls under causal confusion, where end-to-end models with historical input learn a shortcut during training where expert actions are strongly correlated over time. This problem, also referred to as the inertia problem [19], causes the autonomous vehicle to struggle to restart movement after coming to a full stop. This occurs because the training dataset typically exhibits a high probability that the vehicle remains stopped once it has stopped, such as when it halts in front of a red light. In such demonstrations, there might be only one pair of consecutive frames showing the vehicle moving after stopping, whereas the majority of frames depict the vehicle continuing to stop after stopping.

### VI. MODEL OUTPUTS

After sensory inputs are processed and encoded, the primary objective of end-to-end models is to decode appropriate steering and throttle/brake values for each time step. While directly outputting control actions is possible, alternative techniques have been proposed to improve the quality, safety, and interpretability of end-to-end outputs.





**FIGURE 7.** The DriveMLM architecture [39]. The model receives inputs from 4 camera images, LiDAR point clouds, systems messages, and user instructions to output decision states and explanations. System messages refer to text data that contain a description of the driving state and tasks.

**A. DIRECT STEERING AND THROTTLE / BRAKE**

The simplest form of end-to-end models’ outputs is to directly output steering and throttle/brake controls. Models learn to map observations to three outputs of steering, throttle, and brake values at every time step. The output values are generally continuous and bounded between  $[-1, 1]$  for steering, indicating left or right, and range from  $[0, 1]$  for both throttle and brake values.

Imitation learning models learn to map observations to direct controls by minimizing the error, i.e., loss, between the predicted and ground truth controls given by a dataset. Several implementations use a mean squared error loss ( $L_2$  loss) [45], [67], [70], [107], [108], while others opt for an  $L_1$  loss [19], [47], [69], [79], [94], [109], [110], [111]. Deterministic hard targets such as  $L_1$  and  $L_2$  distances can fail to capture the underlying probabilistic, stochastic nature of complex environments [112], [113]. Therefore, a few implementations consider probabilistic information during training. Ohn-Bar et al. [46] train a mixture of experts model with Gaussian distributions to output steering and acceleration using a negative log-likelihood loss. Zhang et al. [114] use soft probabilistic targets with a Kullback-Leibler (KL) divergence loss during training, which enhances supervision with denser information by encouraging the model to learn a distribution of actions for a given state [115]. The supervision is provided by a reinforcement learning coach (Roach) [114], and is leveraged by the authors of [116] and [117] to implement a similar supervision loss. Hu et al. [118] also employ a KL divergence loss during training to model world dynamics and predict future actions and states.

Since direct control outputs for the current time-step only can be short-sighted, some implementations design their models to output future sequences of actions instead [109], [110], [116], [117]. This encourages models to implicitly learn to consider future actions when making decisions at the current time-step.

**B. WAYPOINTS**

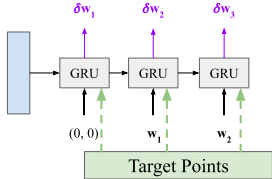
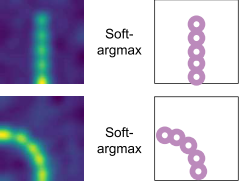
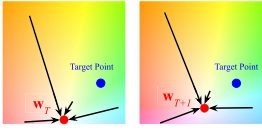
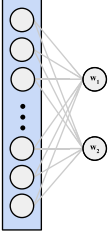
Waypoint outputs indicate future discrete locations the ego-vehicle must reach over a short future time horizon, and are usually updated over every time-step. Low-level controllers then track the given waypoints to obtain the appropriate steering and throttle/brake controls. Waypoint outputs are more interpretable than direct control since they can be directly visualized in BEV or image space and show the intended actions of an end-to-end model over the near future when it executes its actions (Fig. 8). Once end-to-end models encode sensory input using a feature extractor, waypoints can be decoded using several techniques (Table 4).



**FIGURE 8.** An example image from Azam and Kyrki [117] demonstrates how waypoint representations are interpretable and can be viewed in BEV space.

Auto-regression is the most common decoder architecture (Table 6) and is typically implemented using Gated Recurrent Unit (GRU) networks [119]. Instead of relying

TABLE 4. Waypoint generation techniques.

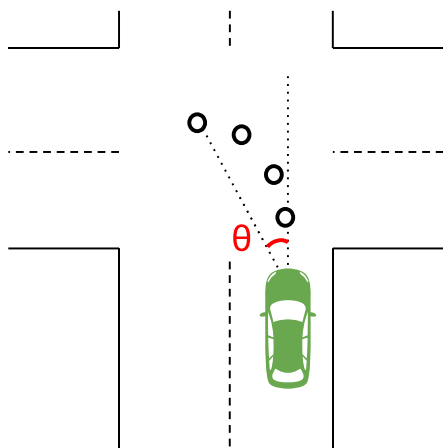
Type	Auto-regression	Heatmap	Offset map	Fully Connected
Image	 <p>Reproduced from : [53]</p>	 <p>Reproduced from : [48]</p>	 <p>Reproduced from : [52]</p>	
Description	<p>The model decodes a sequence of waypoints auto-regressively using RNN architectures. The example implementation [54] shows how the decoder outputs future differential waypoints using encoded features, previous waypoint locations, and target points for each iteration.</p>	<p>The model predicts a heatmap output that is passed through a soft-argmax layer to decode waypoints for each high level command (image example : continue straight, go-left). Proposed by [48].</p>	<p>The model learns to output offsets (black arrows) from query points in Bird's Eye View (BEV) space to decode waypoints (red) with respect to the Target Point (TP) (blue) for multiple iterations in a fixed future horizon [52].</p>	<p>The model decodes waypoints from encoded features using simple fully connected layers or Multilayer Perceptrons (MLPs) [44].</p>

solely on encoded visual features and TPs as inputs to the waypoint decoders, some implementations provide additional contextual information to further refine the waypoint outputs. Chen and Krähenbühl [51] use a two-stage motion planner to first output multiple coarse waypoint predictions of all vehicles in the scene, and afterwards refine the ego-vehicle prediction iteratively during both training and testing using another GRU network. Their viewpoint-invariant perception representations allow their model to learn motion planning from both ego and other vehicles' trajectories. Fu et al. [82] also consider trajectories of all agents in the scene, and refine the ego-vehicle generated waypoints through a refinement module based on another GRU network. Hu et al. [42] use past and predicted future cost maps and traffic light predictions to refine a sampled trajectory auto-regressively. Jia et al. [78] use a coarse-to-fine look-predict-refine strategy and an enlarged decoder architecture for waypoint generation. The look module checks if the initial coarse waypoint trajectory prediction contains no collisions or traffic violations, the predict module anticipates how surrounding agents might react to the initial coarse trajectory, and finally, the refinement module refines the coarse trajectory based on the look and predict modules output features. Shao et al. [56] use a consistency loss during training based on potential intersections between predicted waypoints and an occupancy map to reduce waypoints that are likely to incur collisions. Chen et al. [72] use an offline trajectory tree and motion forecasting of other agents to ensure their waypoints output is safe and kinematically feasible. Wu et al. [116] use an additional temporal module with a GRU network to implement trajectory-guided attention to improve control predictions and fuse them with waypoint trajectory outputs. Azam and Kyrki [117] similarly refine

waypoint outputs by combining them with direct controls through a learned situation-aware fusion network and the self-attention mechanism.

Since waypoints only contain positional information, a controller is necessary to produce control outputs and actuate the ego-vehicle. The control signal depends on the error calculated between the current ego-vehicle state and the defined reference target. The manner in which the reference target and errors are defined with respect to the waypoints vary slightly across implementations, however they generally follow the same principles. Longitudinal reference targets are defined by assuming waypoints are equally spaced in time [48], and lateral reference targets are defined by aiming towards a generated waypoint (Fig. 9). Disentangling velocity from waypoints output by predicting speed separately can lead to better results [37], [61], [92]. In this setup, waypoints are spaced equally in distance rather than time, and are responsible for defining lateral reference targets, whereas longitudinal targets can be calculated using linear programming [93], learned time-spaced waypoints [61], or learned target speeds [37], [92], [104].

Proportional – Integral – Derivative (PID) controllers [120] are the most common among waypoint-based end-to-end models, and are used for both lateral (steering) and longitudinal (throttle/brake) control. Rosero et al. [85], [86] use PIDs exclusively for longitudinal control, and use model predictive control (MPC) [121] instead for lateral control. The authors of [51], [56], [82], and [93] additionally incorporate rule-based safety heuristics for their controllers to perform emergency brakes in case of forecasted collisions. The authors of [78], [116], and [117] learn to predict direct control outputs alongside the waypoints instead of relying solely on rule-based reference targets for actuation.



**FIGURE 9.** The heading error for lateral steering control can be calculated by measuring the angle between the vehicle's heading and an intermediate waypoint target.

### C. AUXILIARY TASKS

The trajectory / direct control action outputs of end-to-end driving models are much sparser and lower in dimension when compared to the dense pixel or point-cloud inputs that they receive. Supervising only those outputs during training can cause causal confusion and make models struggle to generalize. Consequently, many works have incorporated auxiliary output tasks as additional prediction targets alongside motion planning, thereby augmenting the density of the supervision signal.

#### 1) EGO-VEHICLE SPEED

Ego-vehicle speed as an auxiliary task was first used by [107], although they only briefly elaborate on the motivation behind its usage. A more formal study on this task was done by [19], where they identified the inertia causal confusion problem and performed ablation studies to demonstrate the utility of speed prediction. Adding a speed prediction auxiliary task encourages the perception backbone to predict speed-related features, and helps build a more robust association between visual cues and the output velocity that is needed.

#### 2) IMAGE DEPTH

Estimating the depth of the scene using an image involves predicting the depth of each pixel, and models that employ this auxiliary task can consequently have more information about the spatial structure of the scene. However, extracting absolute depth using an image from a monocular camera alone is not possible, contrary to when images are obtained using a stereo camera setup with a known baseline [122]. Monocular depth estimation is therefore done using contextual cues in an image, and the usage of neural networks for estimation [123] emerges as an appealing solution for end-to-end networks as they can be easily integrated into the models as auxiliary tasks.

#### 3) SEMANTIC SEGMENTATION

Semantic segmentation categorizes pixels in an image to their corresponding object class label, allowing the model to better understand the contents in its image [124]. In the context of autonomous driving, this can help models to categorize different actors in the scene, e.g., pedestrians vs. vehicles, and to make better-informed decisions when motion planning.

#### 4) BIRD'S EYE VIEW MAP

Predicting a Bird's Eye View (BEV) map scene involves projecting image and/or LiDAR inputs to a top-down view of the scene. Such a view of the ego-vehicle surroundings "represents an orthographic projection of the physical 3D space which is better correlated with vehicle kinematics than the projective 2D image domain" [52]. Adding such an output as an auxiliary task supervised encourages the model to learn the spatial structure of the scene.

#### 5) OBJECT DETECTION BOUNDING BOX

An object detection auxiliary output is tasked with localizing and classifying objects in an image using bounding boxes. The image is typically a BEV map, and models aim to predict and decode bounding boxes for agents in the scene, such as other vehicles and pedestrians, with the correct location, size, and orientation.

#### 6) TRAFFIC LIGHT STATE

Respecting traffic lights is essential for safe urban driving, however, traffic lights occupy only a tiny portion in an image [49], and end-to-end models might struggle to capture their significance. Adding traffic light classification auxiliary tasks helps bridge the gap and provides more supervision towards this important task.

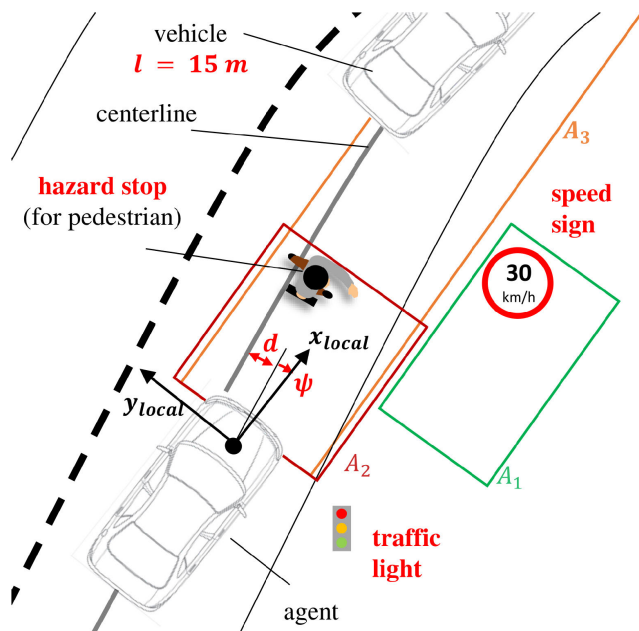
#### 7) REINFORCEMENT LEARNING VALUE

Models supervised using a dataset collected by a reinforcement learning expert [114] can harness value supervision targets such as the value calculated by the expert. Since reinforcement learning value is initially calculated using expected future return from the expert, supervising it allows the model to estimate how dangerous a situation is [114].

### D. AFFORDANCES

Affordances, first proposed by Chen et al. [125] for autonomous driving, refer to intermediate representations of the environment, such as the ego vehicle's angle or lateral distance with respect to lane markings, the time to collision with a lead vehicle, etc. They are estimated using a supervised learning paradigm called direct perception, which serves as a middle ground between end-to-end learning behavior cloning models and modular pipelines. Instead of directly predicting control signals from sensory input in an end-to-end manner, direct perception approaches first predict affordances, and those are then passed to a controller module to output actuation.

Sauer et al. [126] map image inputs to 6 affordance outputs using a CNN feature extractor and a conditional gating mechanism based on the high-level command (see Fig. 10). Those predicted affordances are then leveraged by a low-level controller to output the appropriate steering + brake/throttle actions. Mehta et al. [127] use affordances as auxiliary tasks. Toromanoff et al. [49] similarly use affordances as supervision tasks to train a visual encoder to better understand its environment. The encoder is then used to generate encoded features as input instead of raw images to a model that is trained using RL to output the appropriate controls. Zhao et al. [47] use stop-intention affordances as part of the auxiliary tasks to train a privileged model to drive. The latent vector generated from the encoder of the auxiliary tasks is then used as additional supervision for the sensorimotor agent during its training.



**FIGURE 10.** Affordances predicted by the Conditional Affordance Learning (CAL) model [126] include hazard stops, red traffic light state, speed sign km/h values, distance to the leading vehicle, and relative angle and distance to the road centerline.

## E. MOTION FORECASTING

Motion forecasting is a powerful tool for autonomous urban driving, where many interactive situations arise that require an understanding of other agents' intentions for effective motion planning [128].

Several implementations incorporate motion forecasting in their end-to-end models as an auxiliary task. Chen et al. [72] use a transformer encoder to extract social interactions from the predicted output motion plan and an auxiliary motion forecasting output. Renz et al. [71] achieve motion forecasting by predicting future attributes of other vehicles. The predicted attributes include speed, position, orientation, and size. The authors of [76], [77], and [93] predict a probabilistic object density map of other objects in the scene. Wang et al. [37] decode potential collisions and

the leading vehicle's relative speed to the ego vehicle. Hu et al.'s Model-Based Imitation Learning (MILE) [118] learns a world model alongside motion planning to predict the evolution of the environment and can generate diverse and plausible future states and actions to drive entirely from imagination.

Other implementations incorporate motion forecasting within their end-to-end models rather than treating it as an auxiliary task. Jia et al. use a prediction module that takes previous BEV features along with predicted ego-vehicle outputs to anticipate how surrounding agents would react to the ego-vehicle's actions before it performs them [78]. This allows their model to further refine the output trajectories afterwards based on the prediction module. Hu et al. [43] train a CNN to predict a 2D BEV voxel grid and forecast freespace, where each voxel's state can be either unknown, free, or occupied. Free space forecasting is then used to override waypoints that overlap with occupied locations to stop the ego vehicle. The ST-P3 model implemented by Hu et al. [42] uses a prediction module to obtain a spatio-temporal BEV map that includes the future occupancy of vehicles and pedestrians. The map is then used to score sampled trajectories by a motion planning module. Jia et al. [79] predict the future BEV map of the scene by aligning its features with the Roach expert that uses a privileged BEV input [114]. Shao et al. [56] incorporate a consistency loss from a probabilistic occupancy map to ensure that generated ego-vehicle waypoints do not overlap with occupied locations. Language based models' linguistic output explanations [38], [39], [61], [62] can implicitly forecast motion (see Fig. 11).

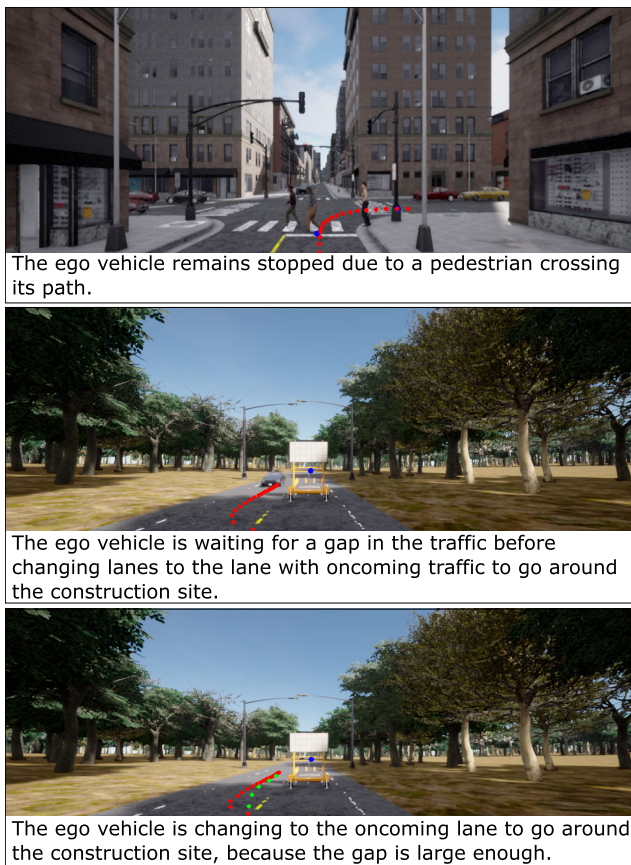
A few methods implement motion forecasting and ego motion planning as joint tasks. The authors of [51] and [129] emphasize learning ego motion planning not just from expert trajectory demonstrations from an ego-vehicle perspective, but also from the trajectories of other vehicles in the scene. This line of thinking resembles how humans might learn by observing others. Learning ego motion planning by also leveraging motion demonstrations from other vehicles further increases the amount of supervision data that can be extracted from training datasets. Zhang and Ohn-bar [129] first learn a motion planning policy from ego-vehicle demonstrations using behavior cloning and use the learned policy to refurbish other vehicles' trajectories in the scene. The ego motion planning model is then retrained using both ego-vehicle demonstrations and the refurbished trajectories from other vehicles. Chen and Krähenbühl [51] apply similar logic, and use partial observability given by the ego-vehicle raw sensors to learn a viewpoint invariant intermediate representation and leverage information from all vehicles. They infer multimodal trajectories of other vehicles afterwards based on possible high-level commands, and optimize their trajectory with the most fitting high-level command. Fu et al. [82] use BEV map features to achieve a comparable effect of viewpoint invariance. Their model considers motion forecasting jointly alongside ego motion



planning rather than treating them as independent tasks, and captures the correlation between the two tasks for safe driving.

### F. LANGUAGE OUTPUTS

Language outputs can potentially enhance end-to-end models' interpretability by commenting the driving behavior [61]. Wang et al. [39] generate textual explanations alongside textual driving decision states that are aligned with an Apollo modular system [130] to obtain control outputs (see Fig. 7). Language has also been used as feedback to supervise and distill knowledge to driving models [38], [62] (more details in Section VIII-C). Language outputs are typically trained to predict the next token using a cross-entropy loss [38], [39].



**FIGURE 11.** CarLLaVa's language explanations comment the current driving behavior [61]. Note that the authors clarify that the commentaries provided are experimental and not always accurate, since they are not always aligned with the actions during training. Red: Space-conditioned predicted path. Green: Time-conditioned predicted waypoints. Blue: Target Point.

## VII. MODEL ARCHITECTURES

Beyond encoders for sensory inputs and decoders for driving outputs, this section explores other aspects related to the architecture of end-to-end models. We examine modular end-to-end architectures, the attention mechanism's application in end-to-end driving, and commonly used end-to-end driving baselines.

### A. MODULAR END-TO-END

While end-to-end models typically follow an encoder-decoder structure with a perception and a planning/control module, some implementations explicitly define additional intermediate modules in their architecture. Hu et al. [42] add a prediction module that receives aggregated features from a perception module and outputs an occupancy cost map for a planning module. Jia et al. [78] propose a larger decoder with look, predict, and refine modules. Wang et al. [68] propose a modular Hierarchical Reinforcement Learning (HRL) model with a perception module and a decision-making/control module that is decomposed for each high-level command subtask. Authors of [56] and [104] incorporate memory bank modules into their models in order to improve temporal reasoning. In contrast to simply concatenating sequential inputs, memory banks learn to selectively store and fuse information from past features that are relevant.

### B. ATTENTION AND TRANSFORMERS

The attention mechanism [131] and transformer architectures [132] have become a staple in recent autonomous driving methods. Initially, the attention mechanism and transformers were most commonly used in natural language processing (NLP) tasks to allow models to selectively focus on relevant parts of an input sequence composed of discrete tokens that represented parts of words. This facilitated more effective learning and understanding of linguistic structures and dependencies. Over time, the use of attention and transformers has expanded beyond NLPs, due to their ability to capture contextual reasoning and relationships between various elements in complex systems and sequence-based tasks, which is particularly useful in end-to-end autonomous driving.

Attention and transformers have been often used for sensor fusion to improve end-to-end models' understanding of the relationships between input sensors. Mehta et al. [127] use a learnable soft-attention layer to make low-dimensional speed and goal conditioning inputs attend to the input image feature vector. Images have a much larger dimension comparatively and contribute more towards the final output, and the usage of soft-attention allows the lower dimension inputs to contribute more towards the output driving action. The TransFuser model [53] use transformer modules to fuse LiDAR and image feature maps from their respective encoders at multiple resolutions (Fig. 12). By stacking the feature maps and treating them as a set of tokens, TransFuser leverages self-attention to better capture the relationships between the perspective 2D image and the 3D BEV LiDAR point clouds. Transformer encoders have also been used to integrate features from multiple sensors *after* they have been fully encoded by features extractors using self-attention [37], [52], [93]. Xu et al. [77] argue that cross-attention can perform better when dealing with multi-modal sensor inputs when compared to self-attention. Cross-attention treats features from different sensors as

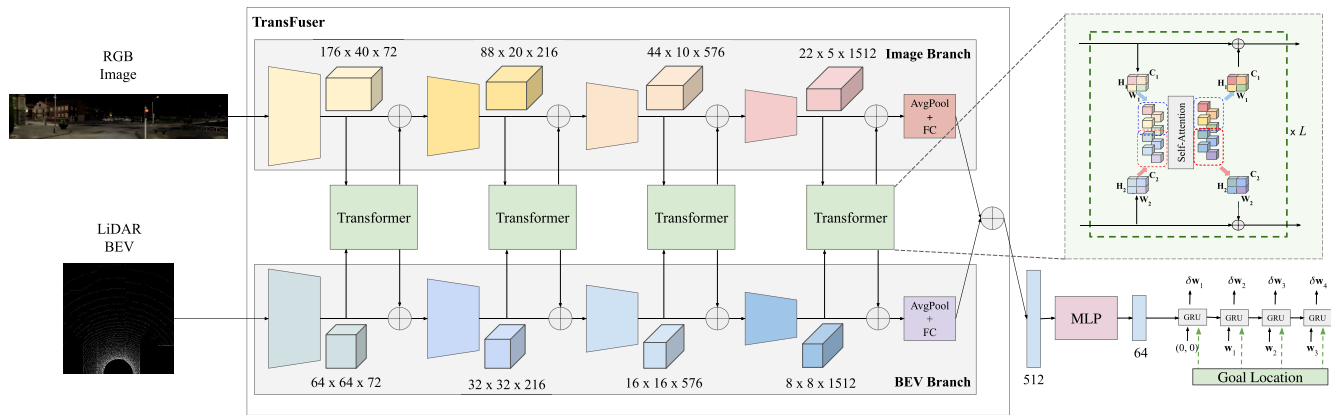


FIGURE 12. The TransFuser architecture [53].

distinct sets to calculate attention scores and find relevant information between modalities. In contrast, self-attention calculates attention scores within a single set, so features from different modalities have to be concatenated before applying self-attention. Xu et al. first concatenate Lidar point clouds, images, and driver attention mask features. Afterwards, they process the concatenated features using both image cross-attention and point cloud cross-attention layers to achieve sensor fusion. Rao et al. [104] fuse a monocular input image with its corresponding depth and semantic predictions using a transformer encoder and a semantic-depth aware decoder to obtain a BEV map. First, each of the image, depth, and semantic features are passed through a global self-attention layer independently to generate their respective embeddings. Afterwards, the embeddings are passed through self-attention, depth, semantic, and visual cross-attention layers to decode a BEV map.

The attention mechanism has also been used to enhance interactions between outputs, inputs, and intermediate layers of end-to-end models. Wu et al. use attention [116] to share encoded features between two branches, one that outputs future waypoints as a trajectory, and another that predicts multiple future control commands. The former's waypoint locations guide the latter's future control signals using attention, highlighting important areas of the encoded input image to output reasonable controls over future time steps. Following similar ideas, they later develop the ThinkTwice model [78], where the model first generates a coarse trajectory in BEV space, then employs a look module followed by a predict module to refine the coarse trajectory and produce the final output. The look module is inspired by how human drivers make sure their target location is safe to drive to before actually executing the necessary maneuvers. This module projects the initial coarse trajectory back to the perspective image plane using the cameras' intrinsics and extrinsics. Multi-scale deformable attention [133] using encoded sensor features and encoded trajectory features is adopted to aggregate information around

the reference reprojected camera pixels, since the information they provide is sparse and might contain reprojection errors. Since those errors are generally local around a small region, deformable attention [133] is precise and outperforms traditional attention from image transformers [134], which have limited spatial feature resolutions. Zhang et al. [41] use a transformer-based architecture to align features from a BEV-based privileged teacher to an image-based student and help the latter learn better image-to-BEV projection. Queries from the teacher's BEV space are mapped to the image space using Inverse Perspective Mapping (IPM) [135] to produce reference points in the image plane. Afterwards, deformable cross-attention based on [133] is employed to attend to regions around the reference points to facilitate image to BEV projection. Ishihara et al. [108] add convolution block attention modules [136] to emphasize auxiliary and driving task feature maps during training. Jaeger et al. [92] study the significance of using a transformer decoder for pooling features before GRU decoders. They show that transformer decoders allow their model to better retain learned spatial information of the scene and lane structures, and ensure that generated waypoints do not exhibit dangerous driving behaviors. Sun et al. [76] employ a TransFuser-inspired backbone to generate a high-dimensional scene feature concatenated from images and point cloud encoders. The concatenated feature is then processed by efficient channel attention (ECA) modules [137], which are applied separately for each auxiliary task and the driving task. The ECA modules add task-specific weights of channels to the high-dimensional concatenated feature and can be used to evaluate how each auxiliary task contributes to the overall scene understanding.

Understanding interactions between agents in the driving scene can be further improved through the usage of transformers and the attention mechanism. Renz et al. [71] use object-level representations instead of raw pixels as inputs to a transformer encoder based on the BERT architecture [138]. The objects can represent the ego vehicle, other vehicles, or the route segments to follow, and they are tokenized

individually as vectors. The model decodes waypoints for the ego vehicle and predicts motion probabilities of other vehicles for the next time-step with the help of the transformer encoder's self-attention. Self-attention allows the model to attend to relevant objects in the scene even when they are geometrically distant, consequently improving interactions between the ego and other vehicles. Chen et al. [72] use a spatial transformer encoder using objects in the environment as input to capture interactions between the ego-vehicle, other dynamic objects, and the reference route. The reference routes and observed objects are first projected into a high-dimensional space using embedding layers before being input into the transformer encoder, where self-attention is employed. Afterwards, the outputs from the transformer encoder and an MLP-encoded offline trajectory tree are attended to using the attention mechanism in order to select a leaf trajectory with the highest score, which is then refined and used for motion planning. Fu et al. [82] use local and global transformers to process BEV map view features of vehicles in the scene to achieve joint motion forecasting and ego planning. The local transformer focuses on distinguishing important regions within the input features for each vehicle, and the global transformer integrates these features across all vehicles to comprehensively consider their interactions. Shao et al. [56] construct a graph using environment and object (vehicles, bicycles, pedestrians) features, and pass them through a graph attention network [139] to model their interactions.

### C. COMMONLY USED BASELINES

#### 1) CONDITIONAL IMITATION LEARNING

The Conditional Imitation Learning (CIL) model (Fig. 5) was the first IL model to be implemented in CARLA [8], and served as a baseline for several future works. Li et al. [107] added output auxiliary tasks to the CIL baseline. Ishihara et al. [108] also output auxiliary tasks, while further adding convolution block attention modules [136] to the CIL network to emphasize task-specific feature maps during training. Liang et al. [140] employ two CIL networks as actor and critic to learn driving using DDPG [141]. Mueller et al. [44] use a binary road segmentation input to CIL, and change the decoder to output waypoints instead of direct control. Prakash et al. [142] experiment with DAgger training algorithms [143] using a CIL baseline. Park et al. [144] employ a CIL baseline in their study to address causal confusion by regularizing imitation learning through a two-stage learning approach. Xiao et al. [94] use CIL to explore the performance of multiple RGB image and depth fusion schemes in urban driving.

Codevilla et al. later proposed CILRS (Conditional Imitation Learning with a ResNet architecture and Speed prediction) [19], which extended the CIL baseline with a deeper convolutional architecture and a speed prediction auxiliary task. Behl et al. [69] studied label efficiency of semantic segmentation visual abstractions in urban driving

using CILRS. Prakash et al. [53] proposed AIM to improve the performance of CILRS by replacing the direct control output with an auto-regressive waypoint prediction network. Later works used the AIM baseline to incorporate auxiliary tasks [52] with AIM-MT, or with a privileged BEV input with AIM-BEV [145]. Kim et al. [109] extend CILRS using temporal inputs and outputs. Zhang et al. [114] use a BEV input to CILRS to train a privileged teacher model using RL, which is then later used to teach and supervise a sensorimotor student agent. Xiao et al. [111] extend CILRS with a wider image Field of View (FOV) using three cameras and a transformer attention mechanism to associate feature maps between them.

#### 2) TRANSFUSER

The TransFuser model (Fig. 12) has been further developed into three additional versions following subsequent improvements [54], [91], [92]. TransFuser+ [91] upgrades TransFuser with an improved expert for data collection and adds depth and semantic segmentation auxiliary tasks. Chitta et al.'s TransFuser [54] uses three cameras instead of one for a wider field of view, an improved encoder, and additional detection and BEV map auxiliary tasks on top of depth and semantic segmentation. Finally, TransFuser++ [92] further improves the expert and the waypoints decoder, and disentangles speed and path predictions.

TransFuser has also been used as a baseline in other implementations. Zhang et al. [75] use the TransFuser architecture and additionally include Radar and HDMap inputs. Sun et al. [76] use the TransFuser architecture to generate an interpretable and high-dimensional scene feature from an image and sequential LiDAR sweeps inputs. References [71], [72] use TransFuser's detection auxiliary task to use object-level inputs to their models. Hanselmann et al. [145] use the TransFuser baseline to study the effect of scenario-generation techniques during data collection on the driving performance during inference.

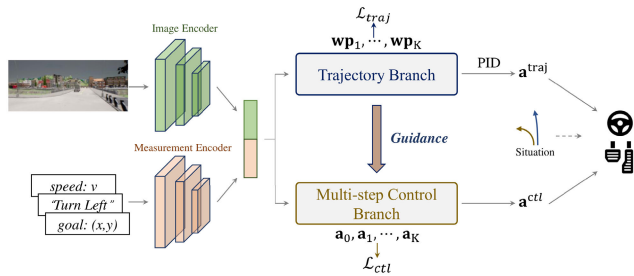
#### 3) TCP

The Trajectory-guided Control and Prediction (TCP) baseline [116] (Fig. 13) implements a situation-based fusion technique based on a dual output from a trajectory and a control branch. Its dual output head has been used by Jia et al. for both the ThinkTwice [78] and DriveAdapter [79] models. The MAGNet model [117] follows a TCP-inspired architecture, and leverages the self-attention mechanism to improve branch interactions. The model also implements control fusion through a learning-based gating network, contrary to the rule-based fusion approach of TCP.

## VIII. TRAINING

Training is the core process that defines deep learning, as it is through training that models develop and learn to solve a given task. Most end-to-end models in CARLA use imitation learning, most notably behavior cloning, as the main training paradigm. Consequently, we explore various





**FIGURE 13. The Trajectory-guided Control and Prediction (TCP) architecture [116].**

aspects that define the imitation learning process in this section, such as dataset collection using experts and dataset augmentation techniques. We additionally dedicate sections towards explaining both the Inverse Reinforcement Learning (IRL) and Reinforcement Learning (RL) paradigms and their application in CARLA.

### A. EXPERT AGENTS

Collecting data in CARLA for supervised learning can be done using autonomous privileged experts that are either rule-based or learning-based. Privileged experts leverage precise, ground-truth information about the environment using the simulator's API. They are more time and cost-efficient than humans for data collection since they can collect data at accelerated speeds and in parallel. However, this advantage comes with the trade-off of comparatively suboptimal performance [91]. This section gives a brief overview of the types of expert drivers that are used for data collection. We refer the reader to [91] and [148] for more details on expert agents.

#### 1) RULE-BASED EXPERTS

Rule-based experts follow a reference global path with dense waypoints using longitudinal and lateral controllers to generate throttle/brake and steering, and avoid obstacles using different emergency braking and motion forecasting techniques (Table 5).

Most implementations follow an A\* [24] reference path with dense waypoints. Some implementations additionally generate a local path that changes dynamically to handle overtaking and obstacle avoidance maneuvers, which is especially useful for handling leaderboards' challenging scenarios. PDM-Lite [148] performs lane switching maneuvers based on the encountered scenario. Kyber-E2E [147] uses sampled trajectories that can deviate from the reference path to handle scenarios.

Early methods define a set of longitudinal velocity targets to follow based on the ego vehicle's situation and location. Situations include stopping when a potential collision is detected, or slowing down near pedestrians and when steering. Otherwise, the ego vehicle is programmed to drive at a constant velocity. Villaflor et al. [146] adapt the constant velocity to match the road speed limit instead. They state that previously defined constant velocities act as a loophole

to handle interactive scenarios by completely avoiding them through slow driving. Wang et al. [37] define additional target speeds based on a chain-of-thought process reasoning with respect to the leading vehicle's speed, time-to-collision, and road speed limit factors. Zhang et al. [147] use an SLT planner [149] (planner using the Frenet frame instead of cartesian coordinates) to sample multiple Bezier curves with different speed profiles (constant accelerations). The trajectory with the least cost is chosen, where cost weights are learned using an inverse reinforcement learning max-margin method [150]. Beißwenger [148] uses the Intelligent Driver Model (IDM) [151] to determine continuous target speeds based on a leading actor. The leading actor does not only represent leading vehicles, but can also represent pedestrians, stop signs, traffic lights, and route obstacles.

Lateral reference targets are mostly defined based on an aim direction towards a waypoint, similar to the example in Fig. 9. Kyber-E2E defines the reference angle based on sampled trajectories.

Most methods employ PID controllers for both lateral and longitudinal control. Jaeger's SEED (Simple and Effective Expert Driver) [91] uses an MLP controller trained with the soft actor-critic algorithm in their learning-based version. PDM-lite uses a linear regression model for longitudinal control. The early expert algorithms (LBC [48], old TransFuser [53]) perform emergency braking if a red light is perceived within a certain distance or if a vehicle or pedestrian is detected within a cone-like area in front of the ego-vehicle. They do not predict other agent's trajectories. Zhang et al. [75] incorporate lane information and time to collision calculations based on velocity and position vector projections to avoid collisions with other cars during lane changing maneuvers. Later methods forecast other agent's motion by unrolling the kinematic bicycle model with an action repeat assumption, i.e. the current control action is going to be repeated in the next time-step.

#### 2) LEARNING-BASED EXPERTS

End-to-end privileged models leverage the simulator's API to obtain ground-truth input values. The LBC approach [48] first trains a privileged model with ground-truth BEV images input to drive using imitation learning from a dataset collected by a rule-based expert. Afterwards, the privileged model itself is used as an expert to collect new data for the sensorimotor model. The Roach (RL Coach) expert [114] similarly uses multiple privileged BEV images as input (Fig. 14) to train a privileged model with PPO [152]. Wu et al. [116] extend the Roach expert with additional rule-based collision detection inspired by the TransFuser expert [53]. Think2Drive [55] employs a model-based RL agent based on Dreamerv3 [153] to drive using BEV input. Contrary to previous methods, the authors of [71] and [146] forego the usage of pixel-level input (such as the rasterized images of the BEV map) and instead directly use object-level representations of the scene as input.



TABLE 5. Rule-based experts.

Method	Reference Path	Local Path	Long Target	Long Ctl	Lat Target	Lat Ctl	Motion Forecasting	Collision Detection
LBC [48]	A*	A*	0, 4, 7 m/s for stop, turn, regular target speeds	PID	Angle to closest waypoint that is >4m away from center of car (changes with lon target speed)	PID	None	Triangular sector
TransFuser (old) [53]	A*	A*	0, 4, 7 m/s for stop, turn, regular target speeds	PID	Angle to closest waypoint that is >4m away from center of car	PID	None	Triangular sector
MMFN [75]	A*	A*	0, 4, 7 m/s for stop, turn, regular target speeds. Lead vehicle speed when following.	PID	Angle to closest waypoint that is >4m away from center of car	PID	Position and velocity vectors	Triangular sector + other lanes during lane switching
SEED [91]	A*	A*	0, 3, 4 m/s for stop, at intersection, regular target speeds	PID	Angle to closest waypoint that is >4m away from center of car	PID	Bicycle model with action repeat	Intersection tests using future predictions of bounding boxes
SEED SAC [91]	A*	A*	0, 3, 4 m/s for stop, turn, regular target speeds	MLP	Angle to closest waypoint that is >4m away from center of car	MLP	Bicycle model with action repeat	Intersection tests using future predictions of bounding boxes
TJPP [146]	A*	A*	0, speed_limit m/s for stop and regular target speeds	PID	Angle to closest waypoint that is >35m away from center of car	PID	Bicycle model with action repeat	Intersection tests using future predictions of bounding boxes
Trans-Fuser++ [92]	A*	A*	0, 2, 5, 8 m/s for stop, near pedestrians, at intersection, regular target speeds	PID	Angle to closest waypoint that is >3.5m away from center of car	PID	Bicycle model with action repeat	Intersection tests using future predictions of bounding boxes
Drive-COT [37]	A*	A*	5 different target speeds based on chain-of-thought reasoning	PID	Angle to waypoint that is a dynamic distance away from center of car based on velocity	PID	Bicycle model with action repeat	Intersection tests using future predictions of bounding boxes
Kyber-E2E [147]	Unspecified	Sampled trajectory	Acceleration given by speed profile	PID	Heading given by sampled Bezier curve	PID	Constant speed and lane path if in lane, constant heading and speed otherwise	Intersection tests using future predictions of bounding boxes
PDM-Lite [148]	A*	A* + Scenario-based planning	Continuous target speed selection from 0 to 24 m/s using the IDM	Linear Regression	Angle to waypoint that is a dynamic distance away from center of car based on velocity	PID	Bicycle model with action repeat	Intersection tests using future predictions of bounding boxes

## B. DATASETS

Imitation learning models in CARLA learn from a dataset that has been collected using a privileged automatic expert. Therefore, the data distribution and structure of datasets play an important role in determining the performance of the sensorimotor imitation learning model.

Datasets are commonly augmented to avoid overfitting and increase the variance in the demonstration examples seen during training to help models generalize better during inference [154]. Directly manipulating the dataset's image features through augmentation techniques such as changing contrast, brightness and tone, adding blur and noise, or removing pixels has been used to add more variance to the dataset [44], [45], [56], [68], [144], [155]. Additional variance can also be achieved by changing weather conditions for each collected frame. Viewpoint augmentation adds more viewpoints to the dataset by shifting camera sensors during data collection or by using additional cameras [48], [49], [54], [59], [92]. These techniques help models learn to identify the essential features of the images, making them less sensitive to variations and distortions and generalize better to new,

unseen data. Augmenting the dataset by simply scaling its size has also been shown to be correlated with improved driving performance [71], [78], [79].

Since driving straight is the most common action during driving, dataset balancing aims to reduce this bias and sample more frequently from demonstrations involving other actions with more steering and acceleration/deceleration [44], [46], [52], [61], [155]. Zimmerlin et al. [156] explore an alternative dataset balancing and filtering technique by estimating target label changes instead of sampling based on demonstration frequencies.

Control noise injection [66], [127], [142], [157] introduces occasional perturbations to the expert driving during data collection so that it is forced to recover from errors and provide demonstrations to address the co-variate shift problem. DAgger (Dataset Aggregation) techniques [143] aim to tackle this issue by querying online human expert guidance when the learning agent drifts and accumulates errors. By aggregating new expert demonstrations into the base training dataset where the agent has previously failed, the learning agent can better recover from errors. While this

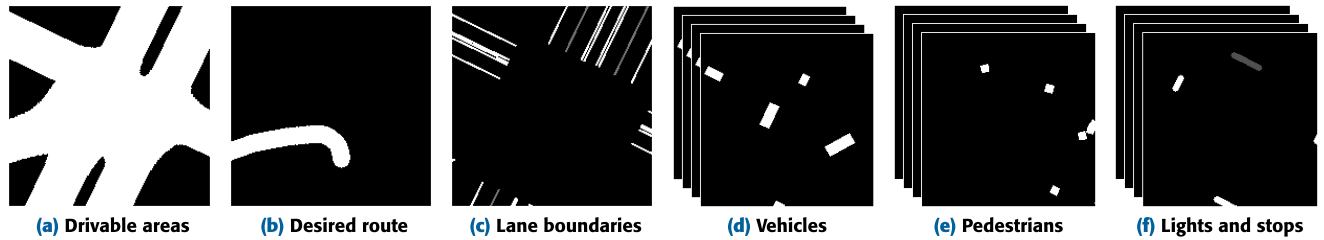


FIGURE 14. The BEV image inputs used for the Roach expert [114].

technique can be applied multiple times iteratively, relying on a human expert for multiple interventions can be costly and inefficient. On the other hand, CARLA's agent and environment states' values can be accessed at will, which makes privileged automatic expert agents an alternative viable option. Prakash et al. [142] explore DAGger techniques applied to vision-based urban autonomous driving using a CIL model. They experiment with a "critical-state-aware" DAGger technique and use a fixed-size replay buffer that focuses on improving the policy where the learning agent exhibits weak performance and behavior. Li et al. [158] train an agent to drive using reinforcement learning with no environmental rewards, but with proxy value functions from human demonstration queries during dangerous situations.

The traffic behavior generated by CARLA is rule-based and model-driven, and is less complex than real-life or data-driven simulations that are more stochastic in nature [159]. Hanselmann et al. [145] tackle this issue by implementing gradient-based optimization techniques to incorporate adversarial scenarios during training. The imitation learning model that learns off of the dataset with such scenarios gains improved collision avoidance performance (Fig. 15).

### C. POLICY DISTILLATION

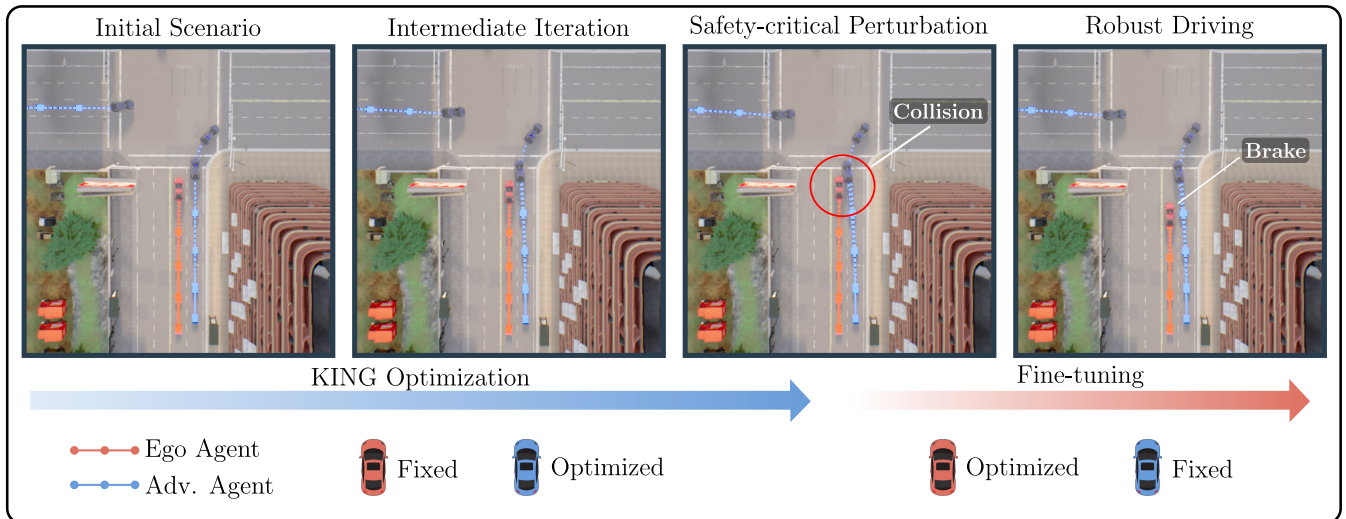
Chen et al. [48] argued that sensorimotor agents struggle to learn using a dataset collected by a rule-based planner, because they have to learn how to see and how to act simultaneously. They proposed Learning By Cheating (LBC), an approach that decomposes imitation learning into two stages. First by training a conditional end-to-end model to focus on how to act by giving it privileged BEV input of the environment. Afterwards, the privileged agent is used to distill its policy and train the sensorimotor model. The privileged agent can give on-policy queries at any state of the environment and provides denser supervision, since the supervision signal can be applied to all conditional high-level commands. Chen et al.'s Learning from All Vehicles (LAV) implementation [51] first trains a perception module and a motion planner with privileged BEV input that learns from both ego and other vehicle trajectories. Both the perception module and the motion planner are then used to distill their policies to a sensorimotor agent. Zhang et al.'s squeeze and mimic (SAM) implementation [47] first trains a squeeze network that takes a privileged ground-truth

semantic segmentation mask input and stop intentions values which indicate how urgent it is to stop to avoid dangerous situations. The network encodes and "squeezes" the inputs into a latent representation that provides additional supervision for the sensorimotor "mimic" network's vision encoder with a normal image input.

Zhang et al. use an RL agent, Roach (RL coach), with privileged BEV input to teach a sensorimotor imitation learning model to drive [114]. Once Roach is fully trained, it is used to collect driving trajectories to generate a dataset with 2D images and corresponding action and value outputs. The imitation learning agent then learns to drive supervised by the collected dataset's outputs and by employing DAGger [142] with on-policy supervision from Roach. Chen et al. [110] teach a privileged agent to estimate discretized action-value functions of the ego vehicle's surrounding environment using Bellman equations. The action-values are then used for policy distillation to supervise a sensorimotor agent.

Reference [41] propose Coaching a Teachable Student (CaT) to further reduce the gap of knowledge that can be found between the privileged teacher and the sensorimotor agent. They use an alignment module with an Inverse Perspective Mapping (IPM) [135] based transformer to map the 3D privileged BEV space to the 2D image features. The IPM facilitates the student's learning process since it reduces the distribution gap between perspective 2D and BEV 3D images. Jia et al.'s DriveAdapter [79] employs alignment modules between student and teacher to reduce the distribution gap in perception. They use Roach as a teacher and freeze its parameters when the training is done. Afterwards, instead of letting the student learn planning again like other policy distillation implementations, the student instead learns to output BEV semantic segmentation using camera and LiDAR inputs. The BEV output is used as input to the Roach model and is supervised using both the ground-truth BEV semantic segmentation loss and the action loss.

Zhang et al. [38] train a camera-only sensorimotor model to drive with language in two stages. First, using feature distillation from a privileged MLLM driving agent, and afterwards by fine-tuning the sensorimotor model with language feedback from the privileged agent. Mei et al. [62] use GPT-4 as an analytic process to reflect on traffic accidents by caused by a lightweight heuristic language model. The



**FIGURE 15.** KING scenario generation image [145]. Collision avoidance improves with the incorporation of adversarial scenarios induced by safety-critical perturbations during training.

analytic process provides improved reasoning and decisions regarding the accidents and adds insights to a memory bank. The samples from the memory bank are then used to distill improved knowledge through supervised fine-tuning.

#### D. VISUAL PRE-TRAINING

Visual pre-training training can simplify learning the driving task by decomposing the training into two steps, where the end-to-end model first learns how to see, and afterwards how to act. This is employed by many reinforcement learning end-to-end models (see Section VIII-E) where the vision encoder is trained using supervised learning, and motion planning is learned using RL with a frozen encoder [49], [66], [67], [68]. The authors of [44], [69], and [70] first train a separate perception module based on semantic segmentation tasks, and afterwards use the encoder's semantic predictions as input to an end-to-end driving model. Wu et al. [40] propose a two-step self-supervised vision pre-training method that can be applied to downstream driving tasks. In the first step, the vision encoder is trained using temporal inputs and two sub-networks (pose and depth) to infer ego motion. Afterwards, the encoder learns to predict ego motion based on a single frame input by focusing on driving relevant information learned from the first step. Finally, the vision encoder can be fine-tuned by supervising the agent with expert driving demonstrations [70].

#### E. REINFORCEMENT LEARNING AND REWARDS

Reinforcement learning has been used in CARLA either as the main training paradigm [49], [55], [66], [67], [101], [102] or combined with imitation learning [110], [114], [140].

The first RL baseline in CARLA [8] used A3C [160] to learn goal-directed urban driving. Episodes were designed to terminate when the ego-vehicle reaches the goal, collides with an obstacle, or runs out of a fixed time limit. The reward

consisted of a weighted sum of five terms. Positive rewards were given for higher speeds and for progression towards the goal destination. Punishments, i.e. negative rewards, were given for collision damage, overlaps with sidewalks, and overlaps with opposite lanes. Despite training on 12 days' worth of data, the RL agent significantly underperformed when compared to Conditional Imitation Learning (CIL). The authors argued that poor results were likely due to RL being brittle [161]. Hyperparameters play a big role in deep RL's performance, and need to be carefully selected with extensive trials and searches. Another possible reason for poor performance is that RL signals (rewards) are too weak to train the entirety of an end-to-end model [49]. The raw input image dimensions, the size of neural networks, and the complexity of urban driving are too big to be trained with RL only, as RL is not very sample efficient [162].

Liang et al. [140] tackle the sample efficiency problem by first pre-training an end-to-end model's parameters with imitation learning. It is then fine-tuned using another network with a similar architecture in an actor-critic setup using Deep Deterministic Policy Gradient (DDPG) [141]. Toromanoff et al. [49] first train a visual encoder to learn to see using semantic segmentation, depth, and affordance auxiliary tasks. Afterwards, the encoder is frozen and the encoded features are used as input to a Rainbow-IQN [163] that is trained using rewards based on desired speed, position, and rotation. Rainbow-IQN is based on DQN [164] and uses a replay buffer to train the network. Chekroun et al. use a similar two stage approach [66], and augment the reinforcement learning replay buffer with offline expert demonstrations that are considered optimal. The model thus learns from both exploration and demonstration data.

Zhang et al. [114] use privileged BEV input to output steering and acceleration distributions by using PPO [152] with clipping to train the network. They use the same rewards

**TABLE 6. Overview of sensorimotor CARLA end-to-end autonomous driving methods. Method: Name and year of end-to-end method. Description: Brief description of end-to-end method. Sensor: Types of sensors used for perception input. 'C' stands for Camera, 'L' for LIDAR, 'R' for Radar, 'M' for HDMap, and 'T' for temporal input. Goal: Goal-conditioning input. 'TP' stands for Target Point (TP) and 'HLC' for High-level Command (HLC). Other: Other types of inputs used. 'Sp' denotes Speed, and 'Steer' denotes steering angle. Enc: Encoder used for sensor feature extraction. Dec: Decoder used to produce the final output. In this table, MLP represents both MLPs and fully-connected layers for the purposes of simplicity. Att: The attention mechanism is used in the model. Type: Output type. 'DC' denotes direct control, and 'WP' denotes waypoints. 'WP(dis)' indicates that steering and velocities from waypoints are disentangled and calculated separately. Aux Tasks: Auxiliary tasks outputs. 'D' stands for depth prediction, 'SS' stands for semantic segmentation, 'Sp' stands for speed, 'BEV' stands bird's-eye view map, 'BB' stands for bounding box (for object detection), 'Traff' stands for traffic light state, and 'RL' stands for reinforcement learning value (when supervised with an RL agent). Forecast: Motion forecasting of other agents in the scene. Training: Training techniques used. 'BC' stands for behavior cloning, 'RL' stands for reinforcement learning, 'IRL' stands for inverse reinforcement learning, 'VPT' stands for visual pre-training, 'PD' stands for policy distillation, and 'CE' stands for a cross-entropy loss for next token prediction. Bench: Benchmark used.**

Method	Description	Input			Architecture			Output				
		Sensor	Goal	Other	Enc	Dec	Att	Type	Aux Tasks	Forecast	Training	Bench
CIL [45] (2018)	Trains a model to drive with different policies conditioned on high-level commands	C	HLC	Sp	CNN	MLP	-	DC	-	-	BC	CoRL
CAL [126] (2018)	Predicts affordances and conditional affordances based on high level commands	CIT	HLC	-	CNN	MLP	-	Aff	-	-	BC	CoRL
MT [107] (2018)	Adds auxiliary tasks to a CIL model.	C	HLC	-	CNN	MLP	-	DC	D+SS+Sp	-	BC	CoRL
CIRL [140] (2018)	Finetunes a CIL model with RL using a two-stage training approach	C	HLC	Sp	CNN	MLP	-	DC	-	-	BC, RL	CoRL
Driving Policy Transfer [44] (2018)	Trains a CIL model with abstract road segmentation input to output waypoints	C	HLC	-	CNN	MLP	-	WP	-	-	BC, VPT	CoRL
DIM [73] (2019)	Combines elements from IL and model-based RL to teach an agent to drive with desirable behavior and the flexibility to achieve complex, unseen goals	LIT	TP	-	CNN	GRU	-	WP	-	-	BC	CoRL
CILRS [19] (2019)	Extends CIL with a deeper image encoder and a speed prediction auxiliary task	C	HLC	Sp	CNN (ResNet34)	MLP	-	DC	Sp	-	BC	CoRL, NoCrash, LBv1
MM-CIL [94] (2020)	Explores early, middle, and late fusion of RGB channels with depth channels using a CIL model.	C	HLC	Sp	CNN	MLP	-	DC	-	-	BC	CoRL
LSD [46] (2020)	Trains an agent to drive using a mixture model of situation-specific policies to better handle diverse driving scenarios.	C	HLC	Sp	CNN (ResNet50)	MLP	-	DC	Sp+Reconstruction	-	BC	CoRL, NoCrash
DA-RB [142] (2020)	Trains a CIL model with an on-policy DAgger-based training algorithm with critical state awareness and a replay buffer	C	HLC	Sp	CNN (ResNet34)	MLP	-	DC	-	-	BC, DAgger	NoCrash
IARL [49] (2020)	Trains a vision encoder to predict affordances and auxiliary tasks. The encoded features are used as input to a deep RL model.	CIT	HLC	Sp+Steer	CNN (Resnet18)	MLP	-	DC	SS+Aff+Traff	-	RL, VPT	CoRL, NoCrash, LBv1
LBC [48] (2020)	First trains a privileged model to drive, and afterwards uses it to teach another sensorimotor model through policy distillation.	C	HLC	Sp	CNN (ResNet34)	Heatmap	-	WP	-	-	BC, PD, DAgger	CoRL, NoCrash, LBv1
LBC-FF [43] (2021)	Learns to forecast a future freespace occupancy map. The map is then used as input to an LBC baseline.	LIT	HLC	-	-	Heatmap	-	WP	-	✓	BC	NoCrash
FASNet [109] (2020)	Changes CILRS to handle temporal inputs and predict a sequence of action outputs	CIT	HLC	Sp	CNN (ResNet34)	RNN	-	DC	Sp+Pose	-	BC	CoRL, NoCrash, Any-Weather
Label Efficient [69] (2020)	Studies the role of semantic segmentation in autonomous driving. Uses CILRS with semantic segmentation input.	C	HLC	Sp	CNN	MLP	-	DC	Sp+SS	-	BC	NoCrash
OREO [144] (2021)	Addresses causal confusion in imitation learning by regularizing policies through extracting and randomly masking semantic objects during training.	C	HLC	Sp	CNN	MLP	-	DC	Reconstruction	-	BC	CoRL
SAM [47] (2021)	Trains a privileged agent to drive using ground-truth representations of the environment. A sensorimotor agent learns to mimic using the privileged network's latent representation.	C	HLC	Sp	CNN (ResNet34)	MLP	-	DC	SS+Aff	-	BC, PD	NoCrash, Traffic-school
TransFuser (old) [53] (2021)	Uses transformers to fuse image and LIDAR feature encoders at multiple resolutions to drive with global contextual reasoning.	CIL	TP	-	CNN (ResNet34, 18), Transformer	GRU	✓	WP	-	-	BC	Town05, LBv1
AIM [53] (2021)	Changes the CILRS output to predict waypoints auto-regressively conditioned on TPs.	C	TP	Sp	CNN (ResNet34)	GRU	-	WP	-	-	BC	Town05
MT-Attention-CIL [108] (2021)	Adds auxiliary tasks and attention modules to a CIL model.	C	HLC	Sp	CNN (ResNet34)	MLP	✓	DC	D+SS+Traff	-	BC	CoRL, NoCrash
WoR [110] (2021)	Trains a privileged agent to learn discrete action-values of the environment. The action-value functions are then used to supervise a sensorimotor agent.	C	HLC	Sp	CNN (ResNet34)	MLP	-	DC	SS	-	BC, RL, PD	NoCrash, LBv1
Rosch [114] (2021)	Trains a privileged model with ground-truth BEV input using RL. It is then used to distill its policy and supervise a sensorimotor agent.	C	HLC+TP	Sp	CNN (ResNet34)	MLP	-	DC	Sp+RL	-	BC, RL, PD	NoCrash, LBv1
GRI [66] (2021)	Trains an end-to-end model to drive using off-policy RL with a replay buffer that contains online exploration and offline expert demonstration data.	C3T	HLC	-	CNN (Efficientnet-b1)	MLP	-	DC	SS+Traff	-	RL, BC, VPT	NoCrash, LBv1
NEAT [52] (2021)	Uses a transformer to integrate features from multi-camera inputs and recursively attends to relevant features to associate them to outputs in BEV space.	C3	TP	Sp	CNN (ResNet34) → Transformer	NEAT	✓	WP	SS+BEV	-	BC	NEAT, LBv1
AIM-MT [52] (2021)	Adds auxiliary tasks to the AIM model.	C	TP	Sp	CNN (ResNet34)	GRU	-	WP	D+SS	-	BC	Town05



**TABLE 6. (Continued.) Overview of sensorimotor CARLA end-to-end autonomous driving methods. Method:** Name and year of end-to-end method. **Description:** Brief description of end-to-end method. **Sensor:** Types of sensors used for perception input. ‘C’ stands for Camera, ‘L’ for LIDAR, ‘R’ for Radar, ‘M’ for HDMap, and ‘T’ for temporal input. **Goal:** Goal-conditioning input. ‘TP’ stands for Target Point (TP) and ‘HLC’ for High-level Command (HLC). **Other:** Other types of inputs used. ‘Sp’ denotes Speed, and ‘Steer’ denotes steering angle. **Enc:** Encoder used for sensor feature extraction. **Dec:** Decoder used to produce the final output. In this table, MLP represents both MLPs and fully-connected layers for the purposes of simplicity. **Att:** The attention mechanism is used in the model. **Type:** Output type. ‘DC’ denotes direct control, and ‘WP’ denotes waypoints. ‘WP(dis)’ indicates that steering and velocities from waypoints are disentangled and calculated separately. **Aux Tasks:** Auxiliary tasks outputs. ‘D’ stands for depth prediction, ‘SS’ stands for semantic segmentation, ‘Sp’ stands for speed, ‘BEV’ stands bird’s-eye view map, ‘BB’ stands for bounding box (for object detection), ‘Traff’ stands for traffic light state, and ‘RL’ stands for reinforcement learning value (when supervised with an RL agent). **Forecast:** Motion forecasting of other agents in the scene. **Training:** Training techniques used. ‘BC’ stands for behavior cloning, ‘RL’ stands for reinforcement learning, ‘IRL’ stands for inverse reinforcement learning, ‘VPT’ stands for visual pre-training, ‘PD’ stands for policy distillation, and ‘CE’ stands for a cross-entropy loss for next token prediction. **Bench:** Benchmark used.

TransFuser+ [91] (2021)	Adds auxiliary tasks and improves TransFuser with an improved expert for data collection.	C1L1	TP	-	CNN (ResNet34, 18), Transformer	GRU	✓	WP	D+SS	-	BC	NEAT, LBv1
HHL [70] (2022)	Uses a pre-trained BEV model’s latent features and pure-pursuit steering angles to supervise an end-to-end model.	C1	HLC+TP	Sp	CNN	MLP	✓	DC	SS+BEV	-	BC	CoRL
CNN-Planner [85] (2022)	Trains a learning-based motion planner to output waypoints for lateral steering, and outputs longitudinal controls using a Finite State Machine.	C2L1	HLC	-	CNN (ResNet18)	MLP	-	WP	Traff	-	BC	LBv1
CADRE [67] (2022)	Trains an end-to-end model with attention modules and auxiliary tasks using behavior cloning. Latent features from the behavior cloning model are used as input to an RL model to learn a driving policy.	C1T	TP	Sp+Meas	CNN (DANet)	MLP	✓	DC	Route+SS+Traff	-	RL, BC, VPT	NoCrash, LBv1
MMFN [75] (2022)	Uses a Transfer-inspired architecture to merge HDMap, RGB, LiDAR, and Radar inputs to output waypoints.	C1L1 M1R1	HLC	-	CNN (ResNet34, 18), Transformer	GRU	✓	WP	-	-	BC	LBv1
ST-P3 [42] (2022)	Trains an end-to-end model with interpretable perception, prediction, and planning modules for spatial-temporal feature learning	C4T	HLC	-	CNN (EfficientNet-b4)	GRU	✓	Traj	D+SS+BEV	✓	IRL	Town05
MILE [118] (2022)	Trains a model-based rl approach to learn a wormd model and a driving policy.	C1	TP	Sp	CNN (ResNet18)	MLP	-	DC	SS+BEV	✓	BC	LAV, Town05
LAV [51] (2022)	Learns a viewpoint-invariant representation in order to learn end-to-end driving from ego and other vehicles’ trajectories.	C3L1	HLC+TP	-	CenterPoint, PointPillars, PointPainting	Multiple GRUs	-	WP	SS+BEV+BB	✓	BC, PD	LAV, LBv1
TCP [116] (2022)	Trains an end-to-end model with a control and a trajectory branch. The model uses the attention mechanism to allow the trajectory to guide control signals. The model uses a situation-based fusion scheme to fuse both branches’ outputs	C1	HLC+TP	Sp	CNN (ResNet34)	GRU	✓	WP+DC	Sp+RL	-	BC, PD	LBv1
TransFuser [54] (2022)	Extends the original TransFuser model with more sensors, an improved backbone, and auxiliary tasks.	C3L1	TP	-	CNN (RegNetY-3.2GF), Transformer	GRU	✓	WP	D+SS+BEV+BB	-	BC	Longest6, LBv1
Latent TF [54] (2022)	Uses the TransFuser architecture under a vision-only setup, with the LiDAR replaced with positional embeddings.	C3	TP	-	CNN (RegNetY-3.2GF), Transformer	GRU	✓	WP	D+SS+BEV+BB	-	BC	Longest6, LBv1
Perception Plant [71] (2022)	Uses object level representations as input tokens instead of pixel-based input to a transformer-based model.	C1L1	TP	-	TransFuser backbone → Transformer	GRU	✓	WP	-	✓	BC	Longest6
Vision-Based HRL [68]	Trains a modular end-to-end model with a perception and a decision making and control module using HRL.	C1	HLC	-	CNN (Enet, YOLOv4)	MLP	-	DC	SS+BB	-	RL, VPT	CoRL
CIL++ [111] (2023)	Modifies CILRS by using a multi-camera setup and by adding a transformer-based attention mechanism to associate feature maps across the images.	C3	HLC	Sp	CNN (ResNet34)	MLP	✓	DC	Sp	-	BC	NoCrash, Town05
PPGeo [40] (2023)	Pre-trains a visual encoder to focus on relevant features only for the downstream driving task by prediction ego-motion, depth, and camera intrinsics using vision only.	C1	HLC	-	CNN (ResNet34)	MLP / GRU	-	DC	D+Camera Intrinsics+Ego Motion	-	BC, VPT	CoRL, Town05
CsT [41] (2023)	Uses BEV alignment, attention, and gradual teaching techniques to bridge the distribution gap between teacher and student during policy distillation.	C3	HLC+TP	-	CNN (ResNet34)	Multiple GRUs	✓	WP	SS+BEV+HLC	✓	BC, PD	Longest6
ThinkTwice [78] (2023)	Predicts an initial coarse trajectory, checks the future scene conditioned on the initial action and the destination, then refines the trajectory accordingly.	C4L1T	HLC+TP	-	CNN (ResNet50), LSS, SECOND	Look, Predict, Refine	✓	WP+DC	D+SS+BEV+Sp+RL	✓	BC, PD	Town05, Longest6
InterFuser [93] (2023)	Uses a transformer-based model to fuse multi-view features, and generates waypoints based on a density map and consequent safety-critical actions.	C31L	TP	-	CCNN(ResNet50, 18) → Transformer	Transformer → GRU	✓	WP(dis)	Traff+BEV+BB	✓	BC	Town05, NEAT, LBv1
ReasonNet [56] (2023)	Builds on the InterFuser architecture and adopts global and temporal reasoning modules to improve understanding of scene relationships and interactions.	C4L1T	TP	-	CNN (ResNet50), PointPillars, Transformer	Transformer → GRU	✓	WP	Traff+BEV+BB	✓	BC, VPT	Town05, NEAT, LBv1
TransFuser++ [92] (2023)	Explores the hidden biases of end-to-end learning that explain their improved performance. The findings are used to improve the performance of TransFuser.	C1L1	TP	-	CNN (RegNetY-3.2GF), Transformer	Transformer → GRU	✓	WP(dis)	D+SS+BEV+BB	-	BC	Longest6, LAV, LBv1
RLAD [102] (2023)	Trains a model end-to-end using purely RL signals by adopting a lightweight vision encoder.	C1T	TP	Sp + Steer	CNN	MLP	-	DC	Traff	-	RL	NoCrash
DriveAdapter [79] (2023)	Uses adapter modules during policy distillation to reduce the distribution gap between ground-truth and image projected BEV maps.	C4L1T	TP	-	CNN, LSS, SECOND	MLP	-	DC	-	✓	BC, PD	Town05, Longest6
II-DSU [76] (2023)	Uses the TransFuser backbone to generate a high-dimensional feature representation of the scene that is supervised by planning and auxiliary tasks for the final driving task.	C1L1T	TP	-	CNN (ConvNeXt-Tiny), Transformer	Multiple GRUs	✓	WP	SS+Traff+BEV+BB+Weather	✓	BC	Town05, NEAT
FAST [72] (2023)	Uses an offline feasible and smooth trajectory tree, combined with object-based scene input and a spatial transformer, to generate trajectory outputs that incorporate interactions with predicted movements of other vehicles	C3L1	TP	Objects, Trajectory tree	TransFuser backbone → Spatial Transformer	GRU	✓	WP	Sp	✓	BC	Town05, Longest6

**TABLE 6. (Continued.) Overview of sensorimotor CARLA end-to-end autonomous driving methods. Method:** Name and year of end-to-end method. **Description:** Brief description of end-to-end method. **Sensor:** Types of sensors used for perception input. 'C' stands for Camera, 'L' for LIDAR, 'R' for Radar, 'M' for HDMap, and 'T' for temporal input. **Goal:** Goal-conditioning input. 'TP' stands for Target Point (TP) and 'HLC' for High-level Command (HLC). **Other:** Other types of inputs used. 'Sp' denotes Speed, and 'Steer' denotes steering angle. **Enc:** Encoder used for sensor feature extraction. **Dec:** Decoder used to produce the final output. In this table, MLP represents both MLPs and fully-connected layers for the purposes of simplicity. **Att:** The attention mechanism is used in the model. **Type:** Output type. 'DC' denotes direct control, and 'WP' denotes waypoints. 'WP(dis)' indicates that steering and velocities from waypoints are disentangled and calculated separately. **Aux Tasks:** Auxiliary tasks outputs. 'D' stands for depth prediction, 'SS' stands for semantic segmentation, 'Sp' stands for speed, 'BEV' stands bird's-eye view map, 'BB' stands for bounding box (for object detection), 'Traff' stands for traffic light state, and 'RL' stands for reinforcement learning value (when supervised with an RL agent). **Forecast:** Motion forecasting of other agents in the scene. **Training:** Training techniques used. 'BC' stands for behavior cloning, 'RL' stands for reinforcement learning, 'IRL' stands for inverse reinforcement learning, 'VPT' stands for visual pre-training, 'PD' stands for policy distillation, and 'CE' stands for a cross-entropy loss for next token prediction. **Bench:** Benchmark used.

InteractionNet [82] (2023)	Uses a transformer to integrate planning and motion forecasting simultaneously instead of separately to learn to drive with better contextual reasoning.	C3LIT	HLC+TP	-	CenterPoint, PointPainting	GRU	✓	WP	SS+BEV+BB	✓	BC	Town05, Longest6
DriveMLM [39] (2023)	Uses an MLLM to decode driving decisions and explanations based on language and sensory inputs	C7LIT	HLC	Lang	ViT, SPT	MLLM	✓	Decision States	-	✓	CE	Town05
CNN-Planner++ [86] (2024)	Increases the density of CNN-Planner's output waypoints.	C2L1	HLC	-	CNN (ResNet 18)	MLP	-	WP	Traff	-	BC	LBv1, LBv2
Enhancing [104] (2024)	Uses a transformer-based architecture with perception, memory, and planning modules to drive end-to-end.	C1T	TP	Sp	CNN (RegNety-32), Transformer	Transformer → GRU	✓	WP(dis)	D+SS+BEV+BB	-	BC	Town05, Longest6
M2DA [77] (2024)	Incorporates human driver attention for scene understanding alongside LIDAR and image inputs using a novel fusion module	C3L1	TP	Driver Attention	CNN (ResNet50, 18), Transformer	Transformer → GRU	✓	WP	BB+Traffic	✓	BC, VPT	Town05, Longest6
RLFOLD [101] (2024)	Uses imitation learning during RL training to provide online on-policy demonstrations when the RL policy contains uncertainty.	C1T	TP	Sp + Steer	CNN	MLP	-	DC	-	-	RL, BC, DAgger	NoCrash
FeD [38] (2024)	Uses language-based feedback to fine-tune an MLLM sensorimotor driving model after initial behavior cloning and policy distillation training stage	C1	HLC+TP	Sp+Lang	VIT (CLIP)	MLLM → MLP	✓	WP	-	✓	BC, PD, CE	LAV
CarLLaVA [61] (2024)	Uses a vision language model to encode monocular images and passes them to a LLaMa backbone to decode output waypoints.	C1	TP	Sp	VLM (LLaVA)	LLM(LLaMA) → MLP	✓	WP(dis)	-	-	BC, (CE)	LBv2
LeapAD [62] (2024)	Uses a VLM for scene understanding and a dual-process decision-making system, where an analytical process continuously tunes a lightweight heuristic process to drive from experience.	C1	TP	-	VLM	LLM (Qwen)	✓	WP	-	✓	Continuous Learning	Town05
MAGNet [117] (2024)	Uses a TCP-inspired architecture to fuse action and trajectory branch outputs, and leverages the self-attention mechanism to improve branch interactions.	C1	TP+HLC	Sp	CNN (ResNet)	GRU	✓	WP+DC	Sp+RL	-	BC	Town05, Longest6
DriveCot [37] (2024)	Fuses multi-view video inputs and outputs waypoints and chain-of-thought driving aspects which are used to obtain the target speed	C6T	TP+HLC	Sp	VIT (Video SwinTransformer)	Transformer → GRU	✓	WP(dis)	Chain-of-thought	✓	BC	Town05

as Toromanoff et al. [49] and also penalize large steering changes to prevent oscillations. Their privileged RL agent, named Roach (Reinforcement Learning Coach) is then used to distill its policy to an imitation learning agent.

Zhao et al. [67] also use a two-stage training approach, where they first train a perception module with a DANet [165] backbone that leverages the attention mechanism using offline driving demonstrations and behavior cloning. The perception module is then frozen, and its latent features are used as input to a PPO agent [152] that learns to drive using a combination of sparse and dense rewards. The sparse rewards include collisions, vehicle blocked infractions, and route deviation infractions. The dense rewards included a deviation degree reward, a deviation distance reward, and a velocity reward.

Contrary to previous works, Coelho et al. [102] train both the vision encoder and the control output using RL signals only. They opt for a smaller size image encoder with around 1M parameters to avoid the issue where the RL signal is too weak to train a deep vision network, such as ResNet50 [166]. They use A-LIX layers [167] to maintain spatial consistency between the encoder's feature maps and prevent self-overfitting, and use the soft actor-critic algorithms [168]

with the same rewards as [114] to train their model end-to-end. Coelho et al. later proposed Reinforcement Learning from Online Demonstrations (RLFOLD) [101] to address the distribution gap found between offline expert demonstrations and online agent exploration data in the replay buffer when combining RL with IL. In addition to the RL soft actor-critic loss [168] that maximizes return and entropy, they incorporated an additional imitation learning loss that aims to maximize the log-likelihood of actions given by a privileged expert rule-based agent. The expert agent is also used to provide demonstrations during the online exploration phase, where its policy can take over and provide actions when the RL's policy uncertainty exceeds a certain threshold.

Chen et al. [110] first learn a forward model to understand how control actions, i.e., steering and throttle/brake, change the next world state. They assume that the world is on rails and their actions only influence the ego-vehicle state, allowing them to discretize the world into a tabular format using only the ego forward model and world states from an offline dataset. This significantly simplifies the estimation of action-value functions through the use of Bellman equations. The value function is represented as a 4D tensor with discrete BEV position (horizontal and vertical), velocity,

and orientation bins. The value function is estimated using positive rewards that are given when the ego-vehicle stays in the desired position, orientation, and speed. It is also rewarded when breaking or having zero velocity in zero-speed regions such as red lights and penalized otherwise. The action-values are then used to supervise a sensorimotor agent using policy distillation.

Li et al. [55] employ the Dreamerv3 model-based RL structure [153] to learn a world model and a planner. They use a recurrent state-space model (RSSM) to encode privileged BEV input into a latent space that is modeled using stochastic and deterministic components. The RSSM is trained to predict a rollout of the world model in latent space so that the planner can learn from its imagination and reduce the amount of interactions necessary with the simulator itself. Their reward function combines additive speed and progression rewards, and penalizes lane center deviation and steering. Steering is penalized when the current steering is different from the previous steering action, in order to avoid oscillations.

#### F. INVERSE REINFORCEMENT LEARNING

A few implementations have explored using Inverse Reinforcement Learning (IRL) for end-to-end autonomous driving using the CARLA simulator [42], [147], [169], [170]. Couto et al. [169] combine Generative Adversarial Imitation Learning (GAIL) [35] with a Behavior Cloning (BC) loss. The BC loss offers stronger supervision at the beginning of training when the discriminator is not yet fully trained. As training progresses, the influence of the BC loss gradually decreases, allowing the discriminator to take on a more dominant role in guiding the policy training. Lee et al. [170] learn a BEV costmap for highway driving by proposing an improved version of maximum entropy deep inverse reinforcement learning [171] with reduced costmap noise and temporal information. The spatiotemporal costmap is used by a Model Predictive Control (MPC) to generate steering and throttle commands. Hu et al. [42] also learn a spatiotemporal BEV costmap function, which is based on learnable subcosts that include safety, comfort, progress, future occupancy, and traffic rules. The cost map is learned using a max-margin loss and is used to choose the best trajectory from a sampled set. The trajectory is further refined using a GRU network with a simple behavior cloning loss. Zhang et al.'s modular implementation [147] similarly uses inverse reinforcement learning for trajectory scoring and sampling in the motion planning module. The optimal trajectory is chosen using a combination of costs such as swiftness, comfort, and safety that are learned using a maximum margin planning algorithm from [150].

#### IX. EVALUATIONS AND DISCUSSION

We provide Longest6, Town05 Long, leaderboardv1, and leaderboardv2 benchmark evaluations of sensorimotor agents in Tables 7, 9, 10, and 8 respectively. Since non-leaderboard evaluations (Tables 9 and 10) are obtained from papers rather

than the leaderboard website, results vary in terms of the significant figures' format. Therefore, we choose to keep the obtained evaluations as they are presented in their respective papers. As for leaderboard evaluations, we mainly focus on the sensors track, where the usage of an HD map is not allowed. This is due to the map track being mostly dominated by modular implementations, which is beyond the scope of this survey.

In this section, we begin by sharing a few insights regarding implementation choices, their ablation studies, and their evaluations. We mostly focus the discussion around leaderboardv1 and variants, since results in leaderboardv2 are drastically lower (highest DS  $\approx$  80, 73, 76, and 7 for leaderboardv1, Longest6, Town05 Long, and leaderboardv2 respectively). Afterwards, we explain factors that result in much lower scores in leaderboardv2. We emphasize that the insights provided in this section are mainly for observation, and are not necessarily indicative of a direct correlation to an improvement in performance. It should also be noted that online leaderboard results are difficult to analyze since the runs are hidden, and the results are highly variable, showing significant disparities when compared with tests using community-created benchmarks [61], [92].

#### A. LEADERBOARDV1 AND VARIANTS EVALUATIONS

##### 1) MODEL INPUT CHOICES

While most top-performing implementations use multi-modal sensors for their models, there have been a few camera-only implementations (DriveCot, MAGNet, Enhancing, TCP) that achieved decent success (see Fig. 16). M2DA, Interfuser, and TransFuser study how incorporating LiDARs can lead to an improvement in their models. DriveMLM conducts a similar study for sensor modality, although the authors notice that incorporating LiDAR point clouds has little impact on performance.

As mentioned previously in Section V, implementations saw varying results regarding the usage of temporal inputs. It could be possible that incorporating temporal inputs is mostly beneficial when the output task explicitly relies on temporal aspects, since most of the methods that saw an improvement with temporal inputs in their ablation studies (ST-P3, ThinkTwice, ReasonNet, Enhancing, CarLLaVa, DriveMLM, DriveCot) also forecast the motion of other vehicles as an additional task. For example, [61] saw an improvement in avoiding rear-end collisions when experimenting with temporal inputs to their CarLLaVa model.

As for goal conditioning inputs, models that use Target Points (TPs) significantly outperform those that don't. The highest DS achieved for methods using only High-level Commands (HLCs) does not pass the 50% mark in any benchmark (highest being GRI [66]  $\approx$  37 for leaderboardv1, WOR [110]  $\approx$  24 for Longest6, and WOR  $\approx$  45 for Town05 Long).

Although all leaderboard methods discussed in this section were tested on the sensors track, it is important to highlight

**TABLE 7.** Leaderboardv1 results, sensor track (accessed Jun2024). DS : Driving Score | RC: Route Completion | IP: Infraction Penalty | Coll Ped: Collisions Pedestrians | Coll Veh: Collision Vehicles | Coll lay: Collisions Layout | Red: Red Light Infractions | Stop: Stop Sign Infractions | Off: Off-road Infractions | Dev: Route Deviations | Time: Route timeouts | Blocked: Agent Blocked.

Method	DS	RC	IP	Coll Ped	Coll Veh	Coll Lay	Red	Stop	Off-road	Dev	Time	Blocked
	Percentage % ↑		[0,1] ↑	Infractions / km ↓								
ReasonNet [56]	79.95	89.89	0.89	0.02	0.13	0.01	0.08	0.00	0.04	0.00	0.01	0.33
InterFuser [93]	76.18	88.23	0.84	0.54	0.37	0.14	0.22	0.00	0.13	0.00	0.01	0.43
TCP [116]	75.14	85.63	0.87	0.00	0.32	0.00	0.09	0.00	0.04	0.00	0.00	0.54
TransFuser ++ [92]	66.32	78.57	0.84	0.00	0.50	0.00	0.01	0.00	0.12	0.00	0.00	0.71
LAV [51]	61.85	94.46	0.64	0.04	0.70	0.02	0.17	0.00	0.25	0.09	0.04	0.10
TransFuser [54]	61.18	86.69	0.71	0.04	0.81	0.01	0.05	0.00	0.23	0.00	0.01	0.43
Latent TF [54]	45.2	66.31	0.72	0.02	1.11	0.02	0.05	0.00	0.16	0.00	0.04	1.82
GRI [66]	36.79	61.85	0.60	0.00	2.77	0.41	0.48	0.00	1.39	1.11	0.34	0.84
TransFuser+ [91]	34.58	69.84	0.56	0.04	0.70	0.03	0.75	0.00	0.18	0.00	0.00	2.41
WOR [110]	31.37	57.65	0.56	0.61	1.35	1.02	0.79	0.00	0.96	1.69	0.00	0.47
IARL [49]	24.98	46.97	0.52	0.00	2.33	2.47	0.55	0.00	1.82	1.44	0.79	0.94
NEAT [52]	21.83	41.71	0.65	0.04	0.74	0.62	0.70	0.00	2.68	0.00	0.00	5.22
AIM-MT [52]	19.38	67.02	0.39	0.18	1.53	0.12	1.55	0.00	0.35	0.00	0.01	2.11
CNN-Planner [85]	15.4	50.05	0.41	0.08	4.67	0.42	0.35	0.00	2.78	0.12	0.00	4.63
LBC [48]	10.86	21.32	0.55	0.00	0.48	0.03	0.85	0.20	0.47	0.24	0.00	0.78
CILRS [19]	5.37	14.4	0.55	2.69	1.48	2.35	1.62	0.00	4.55	4.14	0.00	4.28
Cadre v1 [67]	2.77	65.66	0.07	1.18	6.45	1.29	2.96	0.00	0.90	0.10	0.00	0.99

**TABLE 8.** Leaderboardv2 results, sensor track (accessed Jun2024). DS : Driving Score | RC: Route Completion | IP: Infraction Penalty | Coll Ped: Collisions Pedestrians | Coll Veh: Collision Vehicles | Coll lay: Collisions Layout | Red: Red Light Infractions | Stop: Stop Sign Infractions | Off: Off-road Infractions | Dev: Route Deviations | Time: Route timeouts | Blocked: Agent Blocked. | Yield: Yield Emergency infractions. | Sc Time: Scenario Timeouts. | Min Speed: Minimum Speed Infractions.

Method	DS	RC	IP	Coll Ped	Coll Veh	Coll Lay	Red	Stop	Off-road	Dev	Time	Blocked	Yield	Sc Time	Min Speed
	Percentage % ↑		[0,1] ↑	Infractions / km ↓											
CarLLaVA [61]	6.87	18.08	0.42	0.05	1.17	0.05	0.00	0.11	0.01	0.08	0.00	0.45	0.00	0.13	0.00
TransFuser++ [156]	5.18	11.34	0.48	0.00	1.24	0.04	0.04	0.13	0.03	0.09	0.00	0.73	0.00	0.39	0.00
CNN-Planner++ [86]	1.23	9.55	0.31	0.25	1.64	0.25	0.25	0.40	0.43	0.10	0.30	0.60	0.10	1.20	0.15

the role of HD maps in the context of end-to-end model inputs. MMFN [75] incorporated an HD map from the leaderboard's map track in their end-to-end implementation, experimenting with both rasterized and vectorized map formats. The latter yielded better results in their ablation studies, and achieved a DS of 22.8 in the online leaderboard. Although this score was competitive at the time, it has since been surpassed by more recent sensors track implementations, despite the additional HD map input. This could therefore be attributed to factors beyond the input data itself, such as advances in model outputs and decoder architectures, which are further explained below.

## 2) MODEL OUTPUT CHOICES

Only DriveAdapter achieves decent results (DS  $\approx$  71, 72 for Longest6 and Town05 Long respectively) with direct control outputs. Most recent implementations use waypoints

since they plan towards the future as well. TransFuser++, Enhancing, and CarLLaVa show how disentangling waypoints, where speeds and headings are calculated separately, can lead to improved performance. Auxiliary tasks have been adopted by several methods in the state-of-the-art. Ablation studies by MILE, ThinkTwice, TransFuser, and II-DSU show a noticeable drop in the DS when excluding auxiliary tasks, especially the BEV, from their models. Motion forecasting of other agents improved driving performance as per the ablation studies of ThinkTwice, II-DSU, FAST, and InteractionNet.

## 3) MODEL ARCHITECTURE CHOICES

A few implementations have experimented with different architectural designs for their models to gauge their impact on driving performance. Chitta et al. [54] experiment with various CNN encoder designs, such as ResNet [166],



**TABLE 9.** Longest6 results. DS : Driving Score | RC: Route Completion | IP: Infraction Penalty. Methods with “\*” have been evaluated by other papers and not by the paper itself. Since those methods have been evaluated multiple times by others with varying results, we choose to keep the highest evaluation and cite the paper that obtained the evaluation in brackets “()”. It should be noted that not all methods use the same amount of training data.

Method	DS↑	RC	IP
Enhancing [104]	73.4	93.5	0.78
MAGNet [117]	71.43	84.54	0.87
DriveAdapter [79]	71.4	88.2	0.85
TransFuser++ [92]	69 ± 0	94 ± 2	0.72 ± 0.01
ThinkTwice [78]	66.7	77.2	0.84
M2DA [77]	64.5 ± 3.1	85.2 ± 3.6	0.76 ± 0.01
FAST [72]	59.82 ± 2.14	84.88 ± 2.21	0.73 ± 0.05
CaT [41]	58.36 ± 2.24	78.79 ± 1.50	0.77 ± 0.02
LAV* [51](TransFuser++)	58 ± 1	83 ± 1	0.68 ± 0.02
Perception PlanT [71]	57.66±5.01	88.20±0.94	0.65±0.06
InteractionNet [82]	51.98±1.23	87.32±6.15	0.60±0.01
TCP* (TransFuser++)	48 ± 3	72 ± 3	0.65 ± 0.04
InterFuser* (TransFuser++)	47 ± 6	74 ± 1	0.63 ± 0.07
TransFuser	47.30 ± 5.72	93.38 ± 1.20	0.50 ± 0.06
Latent TransFuser	37.31 ± 5.35	95.18 ± 0.45	0.38 ± 0.05
NEAT* (CaT)	24.08 ± 3.30	59.94 ± 0.50	0.49 ± 0.02
WOR* (ThinkTwice)	23.6	52.3	0.59

RegNet [172], and ConvNeXt [173]. They found that RegNet backbones worked best for the TransFuser model. Renz et al. show how using a Visual Language Model (VLM) image encoder in CarLLava outperforms commonly used ResNet encoders. They also conduct studies on the model’s scale, demonstrating that increasing the number of parameters improves performance up to a certain point, after which further increases result in decreased performance.

The attention mechanism and transformers have seen significant use in recent end-to-end driving models, as they enhance performance through improved reasoning when combining information from multiple sources, which is reflected in Fig. 17. Ablation studies by Interfuser, MAGNet, and M2DA show the value of attention for their models’ driving performance.

Jaeger et al. demonstrate how using transformer decoders for pooling features before waypoint decoding can better retain spatial information, and lead to improved Route Completion (RC) and less collisions with static objects. Jia et al. [78] show how larger and more sophisticated decoders lead to improved driving performance. Chen et al. [51] show how iterative refinement for GRU decoders improves driving performance.

4) TRAINING PARADIGM CHOICES

Imitation Learning (IL) currently outperforms Reinforcement Learning (RL) by a significant margin. Additionally, there have been very few RL implementations in the benchmarks

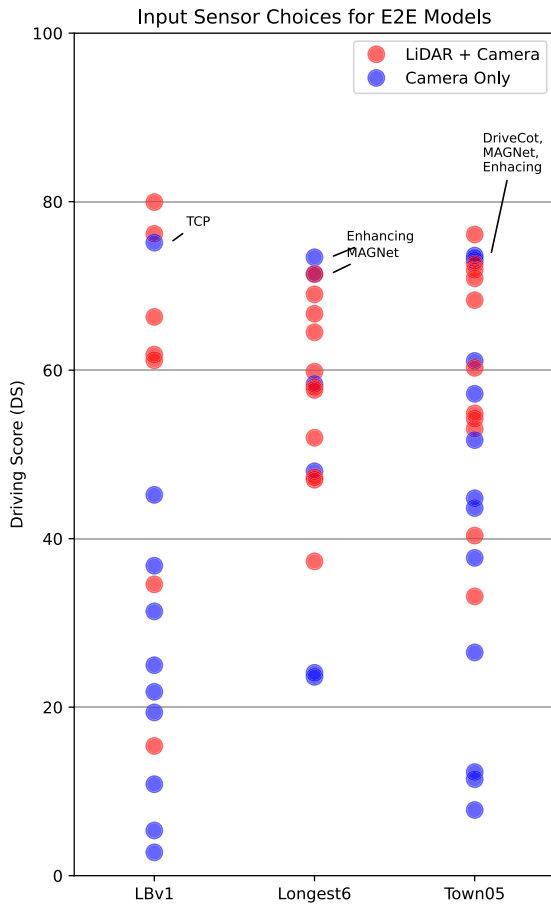
**TABLE 10.** Tonw05 Long results. DS : Driving Score | RC: Route Completion | IP: Infraction Penalty. Methods with “\*” have been evaluated by other papers and not by the paper itself. Since those methods have been evaluated multiple times by others with varying results, we choose to keep the highest evaluation and cite the paper that obtained the evaluation in brackets “()”. It should be noted that not all methods use the same amount of training data.

Method	DS↑	RC	IP
DriveMLM [39]	76.1	98.1	0.78
DriveCot [37]	73.6	96.8	0.76
MAGNet [117]	73.3	98.5	0.79
Enhancing [104]	73.0	98.2	0.73
M2DA [77]	72.6 ± 5.7	89.7 ± 7.8	0.80 ± 0.05
DriveAdapter [79]	71.9	97.3	0.74
ThinkTwice [78]	70.9±3.4	95.5±2.6	0.75±0.05
InterFuser [93]	68.31 ± 1.86	94.97 ± 2.87	-
MILE* [118] (ThinkTwice)	61.1±3.2	97.4±0.8	0.63±0.03
II-DSU [76]	60.28 ± 4.03	91.88 ± 2.60	-
TCP* [116] (ThinkTwice)	57.2±1.5	80.4±1.5	0.73±0.02
FAST [72]	54.87 ± 2.78	95.68 ± 1.96	-
LAV [51] (FAST)	54.24 ± 3.19	68.73 ± 3.80	0.73±0.02
TransFuser* [54](II-DSU)	53.04 ± 3.54	84.83 ± 2.43	-
LeapAD [38]	51.7	100	0.517
WOR* [110] (Interfuser)	44.80 ± 3.69	82.41 ± 5.01	-
Roach* [114] (InterFuser)	43.64 ± 3.95	80.37 ± 5.68	-
InteractionNet [82]	40.37±0.41	72.71±1.65	-
NEAT* [52] (Interfuser)	37.72 ± 3.55	62.13 ± 4.66	-
TransFuser [53](old)	33.15 ± 4.04	56.36 ± 7.14	-
AIM [52] (TransFuser)	26.50 ± 4.82	60.66 ± 7.66	-
LBC* [48] (InterFuser)	12.3±2.0	31.9±2.2	0.66±0.02
ST-P3 [42]	11.45	83.15	-
CILRS* [19] (InterFuser)	7.8±0.3	10.3±0.0	0.75±0.05

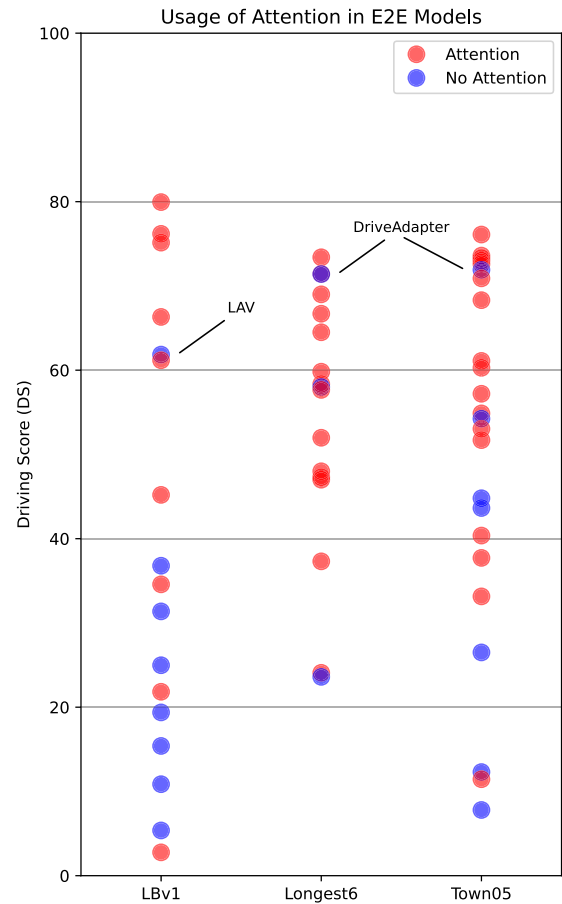
provided by the tables (only IARL, CADRE, and GRI). RL methods mainly shine in CARLA as experts, when they have privileged access to the environment (e.g. Roach expert [114] or Think2Drive [55]). There hasn’t been a sensorimotor RL implementation with a DS above 40 in leaderboardv1, Longest6, or Town05 Long. Roach only uses RL for its expert; the sensorimotor model is trained using behavior cloning.

B. LEADERBOARDV2

There is currently a big disparity between the highest DS in leaderboardv1 and leaderboardv2. As explained by [55], this is primarily due to factors such as much longer route lengths, and significantly more complex scenarios. This results in a much higher accumulation of infractions, causing exponential decay in the Infraction Penalty (IP) due to multiplicative penalty calculation. The authors of [61] and [156] describe that the way the the DS is calculated makes it so a reduction in the Infraction Penalty (IP) score does not linearly correlate with an increase in Route Completion (RC).



**FIGURE 16.** Plot showcasing the type of input modalities used and their corresponding DS, based on data from Tables 7, 9, 10. Most top-performing implementations use multi-modal sensor inputs. High-scoring implementations that use cameras only are annotated.



**FIGURE 17.** Plot showcasing the usage of the attention mechanism in end-to-end models and the corresponding DS, based on data from Tables 7, 9, 10. Most top-performing implementations use attention. High-scoring implementations that do not use attention are annotated.

Therefore, they demonstrate how the usage of a loophole that stops the driving agent early before route completion improves the DS by avoiding the accumulation of infractions.

Even if leaderboardv2’s route lengths were made shorter, the complex scenarios that occur are still significantly more difficult than leaderboardv1’s. Most expert methods implemented for leaderboardv1 simply stop to avoid collisions. This technique cannot deal with scenarios in leaderboardv2, where overtaking maneuvers and exhibiting complex driving behavior are necessary to progress towards the destination goal.

**X. CHALLENGES AND FUTURE DIRECTIONS**

Beyond the considerations of safety, interpretability, and robustness for autonomous driving, we explore more specific challenges that could be tackled in future works.

**A. HIGH-LEVEL COMMANDS VS TARGET POINTS**

Intuitively, a human can drive around using only directional instructions given by High-level Commands (HLCs). While using Target Points (TPs) generally leads to an improvement in the reviewed methods’ performance, they are currently

over-reliant on the GPS positional information this input provides [92]. Future driving models should be able to drive with HLCs just as well. Since HLCs are commonly represented with one-hot vectors that provide little information, using language models can be promising to make HLCs richer with linguistic semantics.

**B. BEHAVIOR PLANNING**

With the increased complexity of scenarios in leaderboardv2, behavior planning and decision-making are essential to handle complex situations and corner cases. Complex situations require maneuvers such as overtaking, yielding, or swerving away from incoming dynamic obstacles. Currently, maneuvers are directly implemented in expert models through a rule-based formulation [148], or indirectly using sampled trajectories [147] or reinforcement learning [55]. Imitation learning sensorimotor agents, on the other hand, learn to maneuver implicitly by observing those expert demonstrations. Barring collision avoidance heuristics [51], [56], [82], [93] and high-level command inputs, imitation learning models do not explicitly train on deciding what maneuvers to execute. This is mainly due to the complexity

of defining such a supervision target. Reasoning through language holds promise in developing such decision making. High-level commands as a prediction target [41], [51] can also be an option to develop further and incorporate maneuver decision-making in end-to-end models.

### C. MOTION FORECASTING

Another essential component that is required to handle complex scenarios is motion forecasting. Currently, most sensorimotor agents leverage other agents' trajectories from training data to learn motion forecasting; the supervision signal for training is exclusively derived from these trajectories. While it is possible to learn multi-modal stochastic predictions using those demonstrations, e.g., probabilistic object density maps [56], [76], [77], [93], additional constraints and supervision labels need to be incorporated to better model multi-modal trajectory probabilities. The LAV model [51] uses HLCs to condition multi-modal trajectory predictions, although they do not include a probabilistic formulation to other agents' stochasticity. An interesting avenue to further explore is explicitly using cues from lane and road networks' geometry to predict other vehicles' intent and trajectories. Previous works employed HD maps to obtain vectorized lane data and used graph networks to model interactions between agents and lanes for motion forecasting [174], [175]. Similarly, Zhang et al. [75] used VectorNet [103] to encode vectorized HD maps for their end-to-end CARLA implementation. The rasterized BEV map auxiliary predictions used in many end-to-end models could also be potentially processed in future works to explicitly model road networks for improved motion forecasting in end-to-end models.

### D. EXPERTS

Evaluations provided in [148] show that the current best experts do not surpass a Driving Score (DS) of 60 for leaderboadv2, despite privileged perception from the simulator. Current rule-based experts rely on a simple action repeat assumption using the kinematic bicycle model for motion forecasting. This assumption might struggle to forecast motion accurately with the increased complexity of leaderboadv2, highlighting a need for more sophisticated techniques. For instance, in highway scenarios, the kinematic bicycle model fails to accurately model high-speed vehicle dynamics, where factors like tire slip and momentum are significant. Learning-based experts [55], [71], [146] on the other hand, can potentially learn better forecasting and world model predictions using data. However, they might fail to handle scenarios that require sophisticated lane-negotiation techniques [55], where rule-based experts can categorize certain scenarios and hand-engineer rules to tackle them. In general, both learning-based and rule-based experts struggle with difficult merging scenarios found in leaderboadv2 [55], [148]. Developing an improved expert can be a valuable contribution to CARLA autonomous

driving research, as such an expert could also be used for data collection and help improve future imitation learning models. A hybrid rule-based and learning-based planner could be a potential avenue to investigate.

### E. DATASETS AND IMITATION LEARNING

The driving performance of imitation learning models is heavily dependent on training datasets. As seen in Section VIII, structuring and balancing datasets can lead to a better training process and can consequently improve driving performance. Dataset distillation [176] can be particularly interesting for autonomous driving applications by creating a synthesized dataset that retains only meaningful samples, eliminating any redundant data. Developing techniques to focus on difficult or safety-critical samples from the dataset to improve sample efficiency, as shown by [142], is also worth further exploration. Another possibility involves providing denser information from data samples, as shown by Chen and Krähenbühl [51] where they train motion planning using both ego and other vehicles' trajectories.

### F. REINFORCEMENT LEARNING

Sensorimotor models based on imitation learning currently outperform those that use reinforcement learning as the main training paradigm. However, IL models lack explicit feedback on what is good or bad driving behavior, contrary to RL where feedback is achieved through rewards [177]. Ideally, an RL model that learns to effectively evaluate state values can adapt better to unseen situations. It can also potentially surpass the performance upper bounds of expert demonstrations. However, problems such as sample efficiency and designing appropriate rewards [178] are significant challenges that hold back RL from achieving its potential. Improvements in model-based reinforcement learning and world models [55], [110], [113] can improve data efficiency by allowing an agent to learn through interactions with a latent "imagined" world instead of the simulator itself. Another possible solution to improve sample efficiency can be found in curriculum learning [179]. This training strategy can facilitate the learning process by providing the learning agent with easier tasks at the beginning, such as lane following and simple turn scenarios [55], before progressively increasing the complexity of tasks. Learning reward functions instead of manually designing them [178], [180], e.g. inverse reinforcement learning, can also be worth further investigation.

### G. CONTROLLERS

While it is understandable that controllers are not heavily researched in the context of end-to-end autonomous driving and deep learning, bad controller design can be detrimental to driving performance. This is especially relevant in the CARLA literature, where waypoints coupled with PID controllers are very commonly used. Liang et al. [181] show that simply tuning PID gains for the TCP baseline can improve its performance. While most crashes observed

during their experiments were attributed to poor trajectories rather than control instability, they still advocate for the potential benefits of more sophisticated control algorithms, such as MPC [121], where end-to-end waypoint outputs can be incorporated into the cost function. Classical control theory is a well-established field that provides numerous tools for autonomous driving control [182], and exploring these methods could be valuable for improving end-to-end models' robustness in complicated leaderboard scenarios.

Another potential avenue for control design lies in learning-based control as part of a model's end-to-end architecture. Methods that used the dual control and trajectory prediction from TCP (ThinkTwice [78], DriveAdapter [79], MAGNet [117]) observed better performance with the inclusion of learning-based control. Incorporating kinematic feasibility and curvature continuity for output trajectories, as done by the FAST model [72], could also simplify the control task and is worth further study.

## H. BENCHMARKS AND METRICS

The Driving Score (DS), Route Completion (RC), and Infraction Penalty (IP) metrics that are commonly used in CARLA benchmarks generally give a decent indication on the quality of driving performance. However, they still have drawbacks that need to be addressed to accurately reflect what constitutes good driving. For instance, as explained in Section IX-B, the exponential nature of the IP score means that minor infractions over long routes can lead to disproportionately large overall score penalties. This has led to strategies that employ early stopping in leaderboard2 to achieve a better DS [61], [156]. Alternative methods for calculating the DS should be considered when designing future benchmarks, in order to ensure reliable performance assessment across varying route lengths, as in [55].

In the event that such benchmarks get solved, new metrics will be needed to capture more nuanced aspects in driving. For example, under the current metrics, an agent might exhibit risky driving behavior—such as getting uncomfortably close to pedestrians—yet still avoid collisions and infractions, achieving a perfect score. Addressing this requires developing metrics that assess risk, which ties into motion forecasting research. Another metric to consider is comfort, where factors such as longitudinal/lateral acceleration and jerk should be accounted for in future work. Such metrics will require a more refined approach to controller design in end-to-end driving to handle the added constraints.

## XI. CONCLUSION

In this survey, we have showcased several CARLA-based autonomous driving end-to-end implementations and explored various inputs, outputs, architectures, and training techniques that were designed to tackle the challenges posed by urban driving. We summarized reviewed methods in a single table to provide a concise, comprehensive overview. We introduced various official and community-created benchmarks, and analysed results and evaluations of

the state-of-the-art. Finally, we discussed a few challenges concerning CARLA and autonomous driving, and explored potential avenues for future research.

## REFERENCES

- [1] S. Tsugawa, T. Yatabe, T. Hirose, and S. Matsumoto, "An automobile with artificial intelligence," in *Proc. 6th Int. Joint Conf. Artif. Intell. (IJCAI)*, San Francisco, CA, USA: Morgan Kaufmann, 1979, pp. 893–895.
- [2] D. A. Pomerleau, "ALVINN: An autonomous land vehicle in a neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, D. Touretzky, Ed., Burlington, MA, USA: Morgan-Kaufmann, 1988, pp. 1–11.
- [3] L. Chen, Y. Li, C. Huang, B. Li, Y. Xing, D. Tian, L. Li, Z. Hu, X. Na, Z. Li, S. Teng, C. Lv, J. Wang, D. Cao, N. Zheng, and F.-Y. Wang, "Milestones in autonomous driving and intelligent vehicles: Survey of surveys," *IEEE Trans. Intell. Vehicles*, vol. 8, no. 2, pp. 1046–1056, Feb. 2023.
- [4] H. Szűcs and J. Hézer, "Road safety analysis of autonomous vehicles: An overview," *Periodica Polytechnica Transp. Eng.*, vol. 50, no. 4, pp. 426–434, Aug. 2022.
- [5] *Road Traffic Injuries*, Fact Sheet, World Health Organization, Geneva, Switzerland, 2023.
- [6] G. Baldini, *Testing and Certification of Automated Vehicles Including Cybersecurity and Artificial Intelligence Aspects*, document Technical Guidance KJ-NA-30472-EN-N, Scientific Analysis or Review, Anticipation and Foresight, Publications Office of the European Union, Luxembourg (Luxembourg), 2020.
- [7] X. Hu, S. Li, T. Huang, B. Tang, R. Huai, and L. Chen, "How simulation helps autonomous driving: A survey of sim2real, digital twins, and parallel intelligence," *IEEE Trans. Intell. Vehicles*, vol. 9, no. 1, pp. 593–612, Jan. 2024.
- [8] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proc. 1st Annu. Conf. Robot Learn.*, vol. 78, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., Nov. 2017, pp. 1–16.
- [9] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "NuScenes: A multimodal dataset for autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11618–11628.
- [10] H. Caesar, "NuPlan: A closed-loop ML-based planning benchmark for autonomous vehicles," in *Proc. CVPR ADP Workshop*, 2021, pp. 1–5.
- [11] A. Amini, T.-H. Wang, I. Gilitschenski, W. Schwarting, Z. Liu, S. Han, S. Karaman, and D. Rus, "VISTA 2.0: An open, data-driven simulator for multimodal sensing and policy learning for autonomous vehicles," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2022, pp. 2419–2426.
- [12] M. Althoff, M. Koschi, and S. Manziinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 719–726.
- [13] C. Gulino, "Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research," in *Proc. Neural Inf. Process. Syst. Track Datasets Benchmarks*, 2023, pp. 7730–7742.
- [14] E. Vinitzky, N. Lichtlé, X. Yang, B. Amos, and J. Foerster, "Nocturne: A scalable driving benchmark for bringing multi-agent learning one step closer to the real world," 2022, *arXiv:2206.09889*.
- [15] P. S. Chib and P. Singh, "Recent advancements in end-to-end autonomous driving using deep learning: A survey," *IEEE Trans. Intell. Vehicles*, vol. 9, no. 1, pp. 103–118, Jan. 2024.
- [16] S. Teng, X. Hu, P. Deng, B. Li, Y. Li, Y. Ai, D. Yang, L. Li, Z. Xuanyuan, F. Zhu, and L. Chen, "Motion planning for autonomous driving: The state of the art and future perspectives," *IEEE Trans. Intell. Vehicles*, vol. 8, no. 6, pp. 3692–3711, Jun. 2023.
- [17] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, "End-to-end autonomous driving: Challenges and frontiers," 2023, *arXiv:2306.16927*.
- [18] D. Coelho and M. Oliveira, "A review of end-to-end autonomous driving in urban environments," *IEEE Access*, vol. 10, pp. 75296–75311, 2022.
- [19] F. Codevilla, E. Santana, A. Lopez, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 9328–9337.
- [20] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 6, pp. 4909–4926, Jun. 2022.



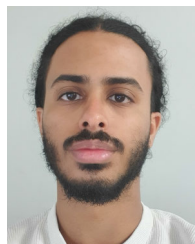
- [21] S. Hagedorn, M. Hallgarten, M. Stoll, and A. Condurache, "The integration of prediction and planning in deep learning automated driving systems: A review," 2023, *arXiv:2308.05731*.
- [22] Z. Yang, X. Jia, H. Li, and J. Yan, "LLM4Drive: A survey of large language models for autonomous driving," 2023, *arXiv:2311.01043*.
- [23] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.
- [24] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. SSC-4, no. 2, pp. 100–107, Jul. 1968.
- [25] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck, "Route planning in transportation networks," in *Algorithm Engineering*. Cham, Switzerland: Springer, 2016, pp. 19–80.
- [26] B. Paden, M. Cáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Vehicles*, vol. 1, no. 1, pp. 33–55, Mar. 2016.
- [27] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.
- [28] CARLA. *Carla Leaderboard*. Accessed: Jun. 11, 2024. [Online]. Available: <https://leaderboard.carla.org/leaderboard/>
- [29] I. Gog, S. Kalra, P. Schaffhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica, "Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2021, pp. 8806–8813.
- [30] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
- [31] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, and J. Peters, "An algorithmic perspective on imitation learning," *Found. Trends Robot.*, vol. 7, nos. 1–2, pp. 1–179, 2018.
- [32] A. Y. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2000, pp. 663–670.
- [33] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proc. 21st Int. Conf. Mach. Learn. (ICML)*, 2004, p. 1.
- [34] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *Proc. 23rd AAAI Conf. Artif. Intell.*, Chicago, IL, USA, vol. 8, Jul. 2008, pp. 1433–1438.
- [35] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1–11.
- [36] I. J. Goodfellow, "Generative adversarial networks," *Commun. ACM*, vol. 63, no. 11, pp. 139–144, 2020.
- [37] T. Wang, E. Xie, R. Chu, Z. Li, and P. Luo, "DriveCoT: Integrating chain-of-thought reasoning with end-to-end driving," 2024, *arXiv:2403.16996*.
- [38] J. Zhang, Z. Huang, A. Ray, and E. Ohn-Bar, "Feedback-guided autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2024, pp. 15000–15011.
- [39] W. Wang, J. Xie, C. Hu, H. Zou, J. Fan, W. Tong, Y. Wen, S. Wu, H. Deng, Z. Li, H. Tian, L. Lu, X. Zhu, X. Wang, Y. Qiao, and J. Dai, "DriveMLM: Aligning multi-modal large language models with behavioral planning states for autonomous driving," 2023, *arXiv:2312.09245*.
- [40] P. Wu, L. Chen, H. Li, X. Jia, J. Yan, and Y. Qiao, "Policy pre-training for autonomous driving via self-supervised geometric modeling," in *Proc. Int. Conf. Learn. Represent.*, 2023, pp. 1–9.
- [41] J. Zhang, Z. Huang, and E. Ohn-Bar, "Coaching a teachable student," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 7805–7815.
- [42] S. Hu, L. Chen, P. Wu, H. Li, J. Yan, and D. Tao, "ST-P3: End-to-end vision-based autonomous driving via spatial-temporal feature learning," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2022, pp. 533–549.
- [43] P. Hu, A. Huang, J. Dolan, D. Held, and D. Ramanan, "Safe local motion planning with self-supervised freespace forecasting," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 12727–12736.
- [44] M. Mueller, A. Dosovitskiy, B. Ghanem, and V. Koltun, "Driving policy transfer via modularity and abstraction," in *Proc. 2nd Conf. Robot Learn.*, vol. 87, 2018, pp. 1–15.
- [45] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 4693–4700.
- [46] E. Ohn-Bar, A. Prakash, A. Behl, K. Chitta, and A. Geiger, "Learning situational driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11293–11302.
- [47] A. Zhao, T. He, Y. Liang, H. Huang, G. Van den Broeck, and S. Soatto, "SAM: Squeeze-and-mimic networks for conditional visual driving policy learning," in *Proc. Conf. Robot Learn.*, 2021, pp. 156–175.
- [48] D. Chen, B. Zhou, V. Koltun, and P. Krähennühl, "Learning by cheating," in *Proc. Conf. Robot Learn.*, 2020, pp. 66–75.
- [49] M. Toromanoff, E. Wirbel, and F. Moutarde, "End-to-end model-free reinforcement learning for urban driving using implicit affordances," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 7151–7160.
- [50] W. G. Najm, J. D. Smith, M. Yanagisawa, and J. A. Volpe, "Pre-crash scenario typology for crash avoidance research," National Highway Traffic Safety Administration, Washington, DC, USA, Tech. Rep. DOT-VNTSC-NHTSA-06-02, Apr. 2007.
- [51] D. Chen and P. Krähennühl, "Learning from all vehicles," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 17201–17210.
- [52] K. Chitta, A. Prakash, and A. Geiger, "NEAT: Neural attention fields for end-to-end autonomous driving," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 15773–15783.
- [53] A. Prakash, K. Chitta, and A. Geiger, "Multi-modal fusion transformer for end-to-end autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 7073–7083.
- [54] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger, "TransFuser: Imitation with transformer-based sensor fusion for autonomous driving," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 11, pp. 1–18, Aug. 2022.
- [55] Q. Li, X. Jia, S. Wang, and J. Yan, "Think2Drive: Efficient reinforcement learning by thinking in latent world model for quasi-realistic autonomous driving (in CARLA-v2)," 2024, *arXiv:2402.16720*.
- [56] H. Shao, L. Wang, R. Chen, S. L. Waslander, H. Li, and Y. Liu, "ReasonNet: End-to-end driving with temporal and global reasoning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 13723–13733.
- [57] X. Jia, Z. Yang, Q. Li, Z. Zhang, and J. Yan, "Bench2Drive: Towards multi-ability benchmarking of closed-loop end-to-end autonomous driving," 2024, *arXiv:2406.03877*.
- [58] U. Müller, J. Ben, E. Cosatto, B. Flepp, and Y. Cun, "Off-road obstacle avoidance through end-to-end learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 18, Y. Weiss, B. Schölkopf, and J. Platt, Eds., Cambridge, MA, USA: MIT Press, 2005, pp. 1–11.
- [59] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Müller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," 2016, *arXiv:1604.07316*.
- [60] P. D. Haan, D. Jayaraman, and S. Levine, "Causal confusion in imitation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 11698–11709.
- [61] K. Renz, L. Chen, A.-M. Marcu, J. Hünemann, B. Hanotte, A. Karnsund, J. Shotton, E. Arani, and O. Sinavski, "CarLLaVA: Vision language models for camera-only closed-loop driving," 2024, *arXiv:2406.10165*.
- [62] J. Mei, Y. Ma, X. Yang, L. Wen, X. Cai, X. Li, D. Fu, B. Zhang, P. Cai, M. Dou, B. Shi, L. He, Y. Liu, and Y. Qiao, "Continuously learning, adapting, and improving: A dual-process approach to autonomous driving," 2024, *arXiv:2405.15324*.
- [63] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.
- [64] J. Zhang, J. Huang, S. Jin, and S. Lu, "Vision-language models for vision tasks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 8, pp. 5625–5644, Aug. 2024.
- [65] A. Radford, "Learning transferable visual models from natural language supervision," in *Proc. Int. Conf. Mach. Learn.*, vol. 139, 2021, pp. 8748–8763.
- [66] R. Chekroun, M. Toromanoff, S. Hornauer, and F. Moutarde, "GRI: General reinforced imitation and its application to vision-based autonomous driving," 2021, *arXiv:2111.08575*.
- [67] Y. Zhao, "CADRE: A cascade deep reinforcement learning framework for vision-based autonomous urban driving," in *Proc. AAAI Conf. Artif. Intell.*, 2022, pp. 3481–3489.
- [68] J. Wang, H. Sun, and C. Zhu, "Vision-based autonomous driving: A hierarchical reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 72, no. 9, pp. 11213–11226, Jun. 2023.

- [69] A. Behl, K. Chitta, A. Prakash, E. Ohn-Bar, and A. Geiger, "Label efficient visual abstractions for autonomous driving," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2020, pp. 2338–2345.
- [70] S. Teng, L. Chen, Y. Ai, Y. Zhou, Z. Xuanyuan, and X. Hu, "Hierarchical interpretable imitation learning for end-to-end autonomous driving," *IEEE Trans. Intell. Vehicles*, vol. 8, no. 1, pp. 673–683, Jan. 2023.
- [71] K. Renz, K. Chitta, O.-B. Mercea, A. S. Koepke, Z. Akata, and A. Geiger, "PlanT: Explainable planning transformers via object-level representations," in *Proc. Conf. Robotic Learn. (CoRL)*, 2022, pp. 459–470.
- [72] W. Chen, Y. Chen, S. Wang, F. Kong, X. Zhang, and H. Sun, "Motion planning using feasible and smooth tree for autonomous driving," *IEEE Trans. Veh. Technol.*, vol. 73, no. 5, pp. 6270–6282, May 2024.
- [73] N. Rhinehart, R. McAllister, and S. Levine, "Deep imitative models for flexible inference, planning, and control," 2018, *arXiv:1810.06544*.
- [74] N. Rhinehart, R. McAllister, K. Kitani, and S. Levine, "PRECOG: PREDiction conditioned on goals in visual multi-agent settings," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 2821–2830.
- [75] Q. Zhang, M. Tang, R. Geng, F. Chen, R. Xin, and L. Wang, "MMFN: Multi-modal-fusion-net for end-to-end driving," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2022, pp. 8638–8643.
- [76] Y. Sun, X. Wang, Y. Zhang, J. Tang, X. Tang, and J. Yao, "Interpretable end-to-end driving model for implicit scene understanding," in *Proc. IEEE 26th Int. Conf. Intell. Transp. Syst. (ITSC)*, Sep. 2023, pp. 2874–2880.
- [77] D. Xu, H. Li, Q. Wang, Z. Song, L. Chen, and H. Deng, "M2DA: Multi-modal fusion transformer incorporating driver attention for autonomous driving," 2024, *arXiv:2403.12552*.
- [78] X. Jia, P. Wu, L. Chen, J. Xie, C. He, J. Yan, and H. Li, "Think twice before driving: Towards scalable decoders for end-to-end autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2023, pp. 21983–21994.
- [79] X. Jia, Y. Gao, L. Chen, J. Yan, P. Langechuan Liu, and H. Li, "DriveAdapter: Breaking the coupling barrier of perception and planning in end-to-end autonomous driving," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2023, pp. 7919–7929.
- [80] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, p. 3337, Oct. 2018.
- [81] Y. Zhou and O. Tuzel, "VoxelNet: End-to-end learning for point cloud based 3D object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4490–4499.
- [82] J. Fu, Y. Shen, Z. Jian, S. Chen, J. Xin, and N. Zheng, "InteractionNet: Joint planning and prediction for autonomous driving with transformers," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2023, pp. 9332–9339.
- [83] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast encoders for object detection from point clouds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 12689–12697.
- [84] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 77–85.
- [85] L. Rosero, J. Silva, D. Wolf, and F. Osório, "CNN-planner: A neural path planner based on sensor fusion in the bird's eye view representation space for mapless autonomous driving," in *Proc. Latin Amer. Robot. Symp. (LARS), Brazilian Symp. Robot. (SBR), Workshop Robot. Educ. (WRE)*, Oct. 2022, pp. 181–186.
- [86] L. A. Rosero, I. P. Gomes, J. A. R. da Silva, C. A. Przewodowski, D. F. Wolf, and F. S. Osório, "Integrating modular pipelines with end-to-end learning: A hybrid approach for robust and reliable autonomous driving systems," *Sensors*, vol. 24, no. 7, p. 2097, Mar. 2024.
- [87] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Proc. Asian Conf. Comput. Vis. (ACCV)*, 2010, pp. 25–38.
- [88] T. Yin, X. Zhou, and P. Krähnenbühl, "Center-based 3D object detection and tracking," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 11779–11788.
- [89] S. Vora, A. H. Lang, B. Helou, and O. Beijbom, "PointPainting: Sequential fusion for 3D object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 4603–4611.
- [90] J. Phillion and S. Fidler, "Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3D," in *Proc. 16th Eur. Conf. Comput. Vis.*, Aug. 2020, pp. 194–210.
- [91] B. Jaeger, "Expert drivers for autonomous driving," M.S. thesis, Mathematisch-Naturwissenschaftliche Fakultät, Wilhelm-Schickard-Institut für Informatik, Tübingen, Germany, 2021.
- [92] B. Jaeger, K. Chitta, and A. Geiger, "Hidden biases of end-to-end driving models," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2023, pp. 8206–8215.
- [93] H. Shao, L. Wang, R. Chen, H. Li, and Y. Liu, "Safety-enhanced autonomous driving using interpretable sensor fusion transformer," in *Proc. Conf. Robot Learn.*, 2023, pp. 726–737.
- [94] Y. Xiao, F. Codevilla, A. Gurram, O. Urfalioglu, and A. M. López, "Multimodal end-to-end autonomous driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 1, pp. 537–547, Jan. 2022.
- [95] P. Polack, F. Althé, B. d'Andréa-Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 812–818.
- [96] E. A. Wan and R. Van Der Merwe, "The unscented Kalman filter for nonlinear estimation," in *Proc. IEEE Adapt. Syst. Signal Process., Commun., Control Symp.*, Oct. 2000, pp. 153–158.
- [97] R. Van Der Merwe, *Sigma-Point Kalman Filters for Probabilistic Inference Dynamic State-Space Models*. Portland, OR, USA: Oregon Health & Science Univ., 2004.
- [98] S. Yin, C. Fu, S. Zhao, K. Li, X. Sun, T. Xu, and E. Chen, "A survey on multimodal large language models," 2023, *arXiv:2306.13549*.
- [99] C. Cui, "A survey on multimodal large language models for autonomous driving," in *Proc. IEEE/CVF Winter Conf. Appl. Comput. Vis. Workshops (WACVW)*, Jan. 2024, pp. 958–979.
- [100] H. Shao, Y. Hu, L. Wang, G. Song, S. L. Waslander, Y. Liu, and H. Li, "LMDrive: Closed-loop end-to-end driving with large language models," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2024, pp. 15120–15130.
- [101] D. Coelho, M. Oliveira, and V. Santos, "RLfOLD: Reinforcement learning from online demonstrations in urban autonomous driving," in *Proc. AAAI Conf. Artif. Intell.*, vol. 38, 2024, pp. 11660–11668.
- [102] D. Coelho, M. Oliveira, and V. Santos, "RLAD: Reinforcement learning from pixels for autonomous driving in urban environments," *IEEE Trans. Autom. Sci. Eng.*, pp. 1–9, 2004.
- [103] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, "VectorNet: Encoding HD maps and agent dynamics from vectorized representation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11522–11530.
- [104] Z. Rao, Y. Cai, H. Wang, Y. Lian, Y. Zhong, L. Chen, and Y. Li, "Enhancing autonomous driving: A low-cost monocular end-to-end framework with multi-task integration and temporal fusion," *IEEE Trans. Intell. Vehicles*, pp. 1–14, 2024.
- [105] C. Wen, J. Lin, T. Darrell, D. Jayaraman, and Y. Gao, "Fighting copycat agents in behavioral cloning from observation histories," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 2564–2575.
- [106] D. Wang, C. Devin, Q.-Z. Cai, P. Krähnenbühl, and T. Darrell, "Monocular plan view networks for autonomous driving," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Nov. 2019, pp. 2876–2883.
- [107] Z. Li, T. Motoyoshi, K. Sasaki, T. Ogata, and S. Sugano, "Rethinking self-driving: Multi-task knowledge for better generalization and accident explanation ability," 2018, *arXiv:1809.11100*.
- [108] K. Ishihara, A. Kanervisto, J. Miura, and V. Hautamäki, "Multi-task learning with attention for end-to-end autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2021, pp. 2896–2905.
- [109] I. Kim, H. Lee, J. Lee, E. Lee, and D. Kim, "Multi-task learning with future states for vision-based autonomous driving," in *Proc. Asian Conf. Comput. Vis. (ACCV)*, Nov. 2020, pp. 654–669.
- [110] D. Chen, V. Koltun, and P. Krähnenbühl, "Learning to drive from a world on rails," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 15570–15579.
- [111] Y. Xiao, F. Codevilla, D. Porres, and A. M. Lopez, "Scaling vision-based end-to-end driving with multi-view attention learning," 2023, *arXiv:2302.03198*.
- [112] C. M. Bishop, "Mixture density networks," Aston Univ., Birmingham, U.K., Working Paper NCRG/94/004, 1994.
- [113] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 1–11.

- [114] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. Van Gool, "End-to-end urban driving by imitating a reinforcement learning coach," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 15202–15212.
- [115] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, *arXiv:1503.02531*.
- [116] P. Wu, X. Jia, L. Chen, J. Yan, H. Li, and Y. Qiao, "Trajectory-guided control prediction for end-to-end autonomous driving: A simple yet strong baseline," in *Proc. NeurIPS*, 2022, pp. 6119–6132.
- [117] S. Azam and V. Kyrki, "Multi-task adaptive gating network for trajectory distilled control prediction," *IEEE Robot. Autom. Lett.*, vol. 9, no. 5, pp. 4862–4869, May 2024.
- [118] A. Hu, "Model-based imitation learning for urban driving," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2022, pp. 20703–20716.
- [119] 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," 2014, *arXiv:1406.1078*.
- [120] K. Heong Ang, G. Chong, and Y. Li, "PID control system analysis, design, and technology," *IEEE Trans. Control Syst. Technol.*, vol. 13, no. 4, pp. 559–576, Jul. 2005.
- [121] E. Carlos Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—A survey," *Automatica*, vol. 25, no. 3, pp. 335–348, May 1989.
- [122] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed., Cambridge, U.K.: Cambridge Univ. Press, 2004.
- [123] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2366–2374.
- [124] P. Wang, P. Chen, Y. Yuan, D. Liu, Z. Huang, X. Hou, and G. Cottrell, "Understanding convolution for semantic segmentation," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2018, pp. 1451–1460.
- [125] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning affordance for direct perception in autonomous driving," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 2722–2730.
- [126] A. Sauer, N. Savinov, and A. Geiger, "Conditional affordance learning for driving in urban environments," in *Proc. Conf. Robot Learn.*, 2018, pp. 237–252.
- [127] A. Mehta, A. Subramanian, and A. Subramanian, "Learning end-to-end autonomous driving using guided auxiliary supervision," in *Proc. 11th Indian Conf. Comput. Vis., Graph. Image Process.*, Dec. 2018, pp. 1–8.
- [128] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. Qi, Y. Zhou, Z. Yang, A. Chouard, P. Sun, J. Ngiam, V. Vasudevan, A. McCauley, J. Shlens, and D. Anguelov, "Large scale interactive motion forecasting for autonomous driving: The Waymo open motion dataset," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 9690–9699.
- [129] J. Zhang and E. Ohn-Bar, "Learning by watching," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 12706–12716.
- [130] Baidu Apollo Team. *Apollo: Open Source Autonomous Driving*. Accessed: Feb. 11, 2019. [Online]. Available: <https://github.com/ApolloAuto/apollo>
- [131] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*.
- [132] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 6000–6010.
- [133] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable transformers for end-to-end object detection," 2020, *arXiv:2010.04159*.
- [134] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *Proc. Eur. Conf. Comput. Vis. Cham, Switzerland: Springer*, 2020, pp. 213–229.
- [135] H. A. Mallot, H. H. Bülthoff, J. J. Little, and S. Bohrer, "Inverse perspective mapping simplifies optical flow computation and obstacle detection," *Biol. Cybern.*, vol. 64, no. 3, pp. 177–185, Jan. 1991.
- [136] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "CBAM: Convolutional block attention module," in *Proc. Eur. Conf. Comput. Vis.*, Sep. 2018, pp. 3–19.
- [137] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo, and Q. Hu, "ECA-Net: Efficient channel attention for deep convolutional neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11531–11539.
- [138] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
- [139] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*.
- [140] X. Liang, T. Wang, L. Yang, and E. Xing, "CIRL: Controllable imitative reinforcement learning for vision-based self-driving," in *Proc. 15th Eur. Conf. Comput. Vis. (ECCV)*, Munich, Germany. Berlin, Germany: Springer, Sep. 2018, pp. 604–620.
- [141] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," In Yoshua Bengio and Yann LeCun, editors, *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, San Juan, Puerto Rico, May 2016.
- [142] A. Prakash, A. Behl, E. Ohn-Bar, K. Chitta, and A. Geiger, "Exploring data aggregation in policy learning for vision-based urban autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 11760–11770.
- [143] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. 14th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 627–635.
- [144] J. Park, Y. Seo, C. Liu, L. Zhao, T. Qin, J. Shin, and T.-Y. Liu, "Object-aware regularization for addressing causal confusion in imitation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34. Red Hook, NY, USA: Curran Associates, 2021, pp. 3029–3042.
- [145] N. Hanselmann, K. Renz, K. Chitta, B. Bhattacharyya, and A. Geiger, "King: Generating safety-critical driving scenarios for robust imitation via kinematics gradients," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2022, pp. 335–352.
- [146] A. Villaflo, B. Yang, H. Su, K. Fragkiadaki, J. Dolan, and J. Schneider, "Tractable joint prediction and planning over discrete behavior modes for urban driving," 2024, *arXiv:2403.07232*.
- [147] W. Zhang, M. Elmaghiubi, K. Rezaee, B. Khamidehi, H. Mirkhani, F. Arasteh, C. Li, M. Ahsan Kaleem, E. R. Corral-Soto, D. Sharma, and T. Cao, "Analysis of a modular autonomous driving architecture: The top submission to CARLA leaderboard 2.0 challenge," 2024, *arXiv:2405.01394*.
- [148] J. Reißwenger, "PDM-lite: A rule-based planner for Carla leaderboard 2.0," Univ. Tübingen, 2024. [Online]. Available: <https://github.com/OpenDriveLab/DriveLM/blob/DriveLM-CARLA/docs/report.pdf>
- [149] W. Zhang, P. Yadmellat, and Z. Gao, "Spatial optimization in spatio-temporal motion planning," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2022, pp. 1248–1254.
- [150] N. D. Ratliff, D. Silver, and J. A. Bagnell, "Learning to search: Functional gradient techniques for imitation learning," *Auto. Robots*, vol. 27, no. 1, pp. 25–53, Jul. 2009.
- [151] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 62, no. 2, pp. 1805–1824, Aug. 2000.
- [152] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
- [153] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse domains through world models," 2023, *arXiv:2301.04104*.
- [154] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *J. Big Data*, vol. 6, no. 1, pp. 1–48, Dec. 2019.
- [155] F. Codevilla, A. M. Lopez, V. Koltun, and A. Dosovitskiy, "On offline evaluation of vision-based driving models," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, Sep. 2018, pp. 246–262.
- [156] J. Reißwenger, "PDM-lite: A rule-based planner for carla leaderboard 2.0," Univ. Tübingen, 2024. [Online]. Available: <https://github.com/OpenDriveLab/DriveLM/blob/DriveLM-CARLA/docs/report.pdf>
- [157] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, "DART: Noise injection for robust imitation learning," in *Proc. Conf. Robot Learn.*, 2017, pp. 143–156.
- [158] Q. Li, Z. Peng, and B. Zhou, "Efficient learning of safe driving policy via human-AI copilot optimization," in *Proc. Int. Conf. Learn. Represent.*, 2022, pp. 1–9.
- [159] F. Mütsch, H. Gremmelmaier, N. Becker, D. Bogdoll, M. R. Zofka, and J. M. Zöllner, "From model-based to data-driven simulation: Challenges and trends in autonomous driving," 2023, *arXiv:2305.13960*.
- [160] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.



- [161] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 3207–3214.
- [162] J. Hao, T. Yang, H. Tang, C. Bai, J. Liu, Z. Meng, P. Liu, and Z. Wang, "Exploration in deep reinforcement learning: From single-agent to multiagent domain," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 7, pp. 1–21, Jul. 2023.
- [163] M. Toromanoff, E. Wirbel, and F. Moutarde, "Is deep reinforcement learning really superhuman on Atari? Leveling the playing field," 2019, *arXiv:1908.04683*.
- [164] V. Mnih, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.
- [165] J. Fu, J. Liu, H. Tian, Y. Li, Y. Bao, Z. Fang, and H. Lu, "Dual attention network for scene segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 3141–3149.
- [166] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
- [167] E. Cetin, P. J. Ball, S. Roberts, and O. Celiktutan, "Stabilizing off-policy deep reinforcement learning from pixels," 2022, *arXiv:2207.00986*.
- [168] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.
- [169] G. C. Karl Couto and E. A. Antonelo, "Generative adversarial imitation learning for end-to-end autonomous driving on urban environments," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2021, pp. 1–7.
- [170] K. Lee, D. Isele, E. A. Theodorou, and S. Bae, "Spatiotemporal costmap inference for MPC via deep inverse reinforcement learning," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 3194–3201, Apr. 2022.
- [171] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner, "Large-scale cost function learning for path planning using deep inverse reinforcement learning," *Int. J. Robot. Res.*, vol. 36, no. 10, pp. 1073–1087, Sep. 2017.
- [172] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 10425–10433.
- [173] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 11966–11976.
- [174] J. Wang, T. Ye, Z. Gu, and J. Chen, "LTP: Lane-based trajectory prediction for autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 17134–17142.
- [175] D. Sierra-Gonzalez, A. Paigwar, O. Erkent, and C. Laugier, "MultiLane: Lane intention prediction and sensible lane-oriented trajectory forecasting on centerline graphs," in *Proc. IEEE 25th Int. Conf. Intell. Transp. Syst. (ITSC)*, Oct. 2022, pp. 3657–3664.
- [176] S. Lei and D. Tao, "A comprehensive survey of dataset distillation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 1, pp. 17–32, Jan. 2024.
- [177] Y. Lu, J. Fu, G. Tucker, X. Pan, E. Bronstein, R. Roelofs, B. Sapp, B. White, A. Faust, S. Whiteson, D. Anguelov, and S. Levine, "Imitation is not enough: Robustifying imitation with reinforcement learning for challenging driving scenarios," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2023, pp. 7553–7560.
- [178] W. B. Knox, A. Allievi, H. Banzhaf, F. Schmitt, and P. Stone, "Reward (Mis)design for autonomous driving," *Artif. Intell.*, vol. 316, Mar. 2023, Art. no. 103829.
- [179] Y. Bengio, J. Louradour, and R. Collobert, "Curriculum learning," in *Proc. Int. Conf. Mach. Learn.*, Aug. 2009, pp. 41–48.
- [180] J. Leike, D. Krueger, T. Everitt, M. Martic, V. Maini, and S. Legg, "Scalable agent alignment via reward modeling: A research direction," 2018, *arXiv:1811.07871*.
- [181] W. Liang, P. R. Balddivieso, R. Drummond, and D. Shin, "Tuning the feedback controller gains is a simple way to improve autonomous driving performance," 2024, *arXiv:2402.05064*.
- [182] Q. Yao, Y. Tian, Q. Wang, and S. Wang, "Control strategies on path tracking for autonomous vehicle: State of the art and future challenges," *IEEE Access*, vol. 8, pp. 161211–161222, 2020.



**YOUSSEF AL OZAIBI** received the B.S. degree in computer science from Université Paris Cite, Paris, France, in 2021, and the M.S. degree in robotics from Sorbonne Université, Paris, in 2023. He is currently pursuing the Ph.D. degree in autonomous vehicle motion planning with the ECE Paris School of Engineering, LISV Laboratory, Université de Versailles Paris-Saclay. His research interests include motion planning, mobile robotics, deep learning, and autonomous driving.



**MANOLO DULVA HINA** (Member, IEEE) received the bachelor's degree in computer engineering from Mapua University (formerly Mapua Institute of Technology), Manila, Philippines, the master's degree in computer science from Concordia University, Montreal, Canada, the Ph.D. degree in computer science from the Université de Versailles St-Quentin-en-Yvelines, in 2011, and the Ph.D. degree in engineering from the Université du Québec, École de Technologie

Supérieure, in 2010. He is currently working as an Associate Professor of computer science with the ECE Paris School of Engineering. His research interests include artificial intelligence, machine learning, intelligent transportation, autonomous machines, multimodal computing, and formal specification.



**AMAR RAMDANE-CHERIF** received the Ph.D. degree from Pierre and Marie Curie University, Paris, in 1998, and the HDR degree from the University of Versailles, in 2007. From 2000 to 2007, he was an Associate Professor. Since 2008, he has been a Full Professor with the LISV Laboratory, Université de Versailles Paris-Saclay. His research interests include software ambient intelligence, semantic knowledge representation, modelling of ambient environment, multimodal interaction

between person, machine and environment, fusion and fission of events, ambient assistance, and software architecture. He is currently a member of the Council Board of the Graduate School of Computer Science at the Paris-Saclay University.

• • •