EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

**Mathematisch-
Naturwissenschaftliche
Fakultät**

Autonomous Vision

Master's thesis

LiDAR-based Object Detection for Planning Transformers

Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik
Autonomous Vision
Luis Winckelmann, `luis.winckelmann@uni-tuebingen.de`, 2022-2023

Bearbeitungszeitraum:	01.12.2022-28.06.2023

Betreuer/Gutachter:	Prof. Dr. Andreas Geiger, Universität Tübingen
Zweitgutachter:	Prof. Dr. Zeynep Akata, Universität Tübingen

Selbstständigkeitserklärung

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

Luis Winckelmann (Matrikelnummer 5615177), June 26, 2023

Abstract

This thesis aims to resolve the apparent disparity in performance between the two primary methodologies in autonomous driving: end-to-end learning and modular pipelines. In simulated driving environments like CARLA, the predominant method for achieving success has been end-to-end learning, which directly learns the mapping between sensor data and vehicle controls. This approach simplifies the process of developing driving systems. On the other hand, the autonomous vehicle industry tends to lean more towards modular pipeline-based strategies. These approaches break down the driving task into smaller components such as detection and planning, and are appealing for their interpretability and safety assurances. Despite these advantages, contemporary modular pipelines exhibit notable shortcomings in their perception modules and are unable to compete with end-to-end methods.

Our research primarily focuses on devising a modular pipeline with improved perception and serves to bridge this performance gap. With this intention, we introduce a novel modular pipeline that capitalizes on recent advances in LiDAR-based 3D object detection and planning. Our method, named SECOND PlanT, achieves state-of-the-art performance on the LAV benchmark, proving its capability to match and even surpass the performance of leading end-to-end models while simultaneously accelerating the speed of inference during deployment. We additionally introduce and provide insights into E2E PlanT, a promising future direction for leveraging the advantages of both paradigms.

Acknowledgments

Firstly, I am sincerely grateful to my supervisors, Kashyap Chitta and Katrin Renz, for their valuable support and extensive guidance throughout this research. I am truly thankful for the time they spent discussing various aspects, providing feedback, addressing code errors, and comforting me during stressful moments. Their advice and expertise have been instrumental in shaping this work.

I would also like to thank my friends. Especially Leo, Sebastian, Ahmet, Jonny, Tim, and Lucas that provided me with support and feedback during my thesis.

Thank you Sophia for always being there for me.

Lastly, I am deeply thankful to my parents for their love and support throughout my life and studies. Their encouragement and unwavering belief in me have been a constant source of motivation throughout my academic journey.

Contents

1 Introduction

In recent years, autonomous driving (AD) has gained significant attention as a promising technology with the potential to revolutionize the transportation system and to enhance road safety. According to the World Health Organization, traffic crashes result in approximately 1.35 million fatalities annually [73], with human errors accounting for 94% of accidents in the USA [92]. Achieving reliable and robust autonomous driving capabilities remains a challenging task due to the complex nature of real-world environments. Researchers and engineers have explored various methodologies to develop effective autonomous driving systems, with different approaches being employed in closed-loop settings and real-world industry applications.

In real-world industry applications, the dominant approach has been the adoption of modular pipeline architectures [111, 103, 98, 127]. These architectures decompose the autonomous driving system into distinct modules responsible for perception, planning, and control. Each module focuses on a specific task, allowing for modularity, interpretability, and fine-grained improvements. This approach has been favored in industry, due to its interpretability, safety guarantees, verifiability, seamless integration with existing systems, and its ability to provide and facilitate algorithmic improvements on individual modules. Within the realm of non-industry research, the evaluation of driving models primarily relies on simulators. Notably, a prevailing trend observed in simulation techniques is the adoption of end-to-end approaches. These approaches aim to learn a mapping directly from sensor inputs to driving actions, effectively bypassing the modular decomposition of the perception, planning, and control components. Noteworthy achievements in this domain have showcased the advantages of end-to-end learning, allowing for end-to-end optimization and potentially capturing complex dependencies in the driving task.

Figure 1.1 shows the entries of the CARLA Leaderboard [10] split into modular pipeline and end-to-end models. In this work, we aim to bridge the clearly visible gap between the success of end-to-end approaches in closed-loop settings and the prevalent use of modular pipeline architectures in real-world industry applications. To this end, we adopt the most successful modular pipeline approach used in the CARLA simulator [30] benchmark, known as Perception PlanT [82]. The Perception PlanT framework comprises three key components: an object detector, a planner called PlanT, and a Controller. However, we focus on enhancing the object detector as previous studies have shown that the planner performs well when provided with ground truth detections [82], while the previous object detector has clear

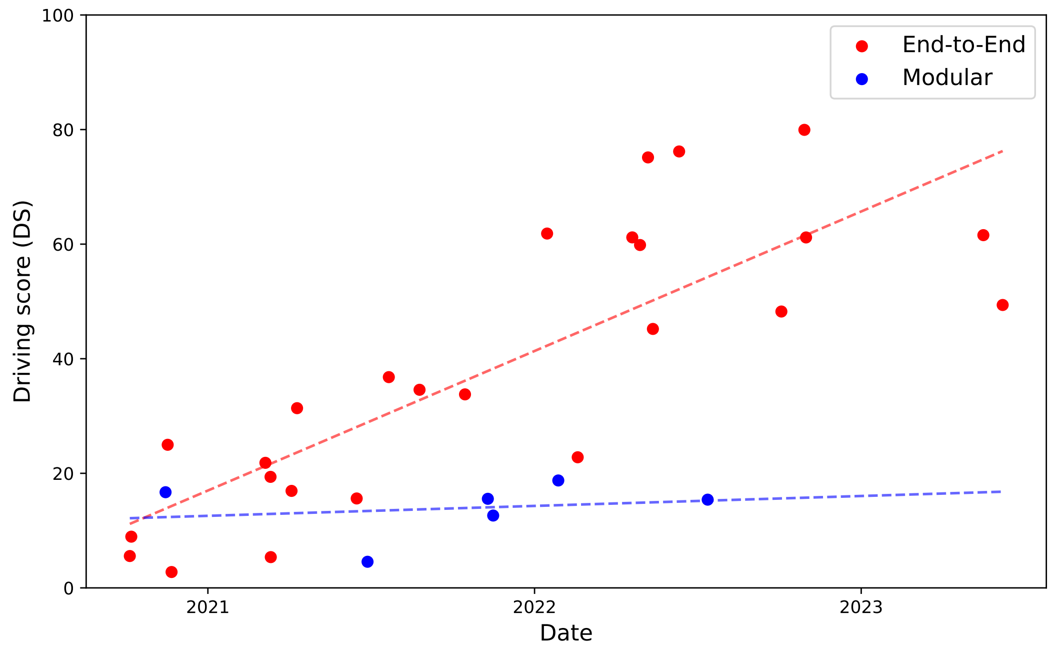


Figure 1.1: **Performance on the CARLA Leaderboard.** In recent years, there has been a steady improvement in performance for end-to-end driving stacks, while the development of modular pipeline models has lagged behind.

problems such as limited LiDAR range, slow inference, and a generally unoptimized architecture. Specifically, we increase the LiDAR range from 180° to 360°, employ more advanced data augmentation, improve inference time significantly, and utilize a more optimized LiDAR-based detection architecture SECOND [115].

Our primary objectives are twofold: (1) closing the performance gap between ground truth detections and the usage of an object detector, and (2) demonstrating performance on par with state-of-the-art end-to-end driving stacks. To achieve these objectives, we integrate the OpenPCDet framework [97], which provides training, data augmentation, and visualizations of cutting-edge LiDAR perception modules, into our modular stack. Among the available options, we select the SECOND [115] model as our object detector. SECOND improves speed due to several factors, including sparse convolutions and LiDAR-only input. Additionally, it achieves better performance through improved loss functions and other high-level optimizations.

To evaluate the effectiveness of our proposed approach, we utilize the LAV benchmark [15] as a widely accepted proxy for the CARLA Leaderboard [10]. By conducting extensive experiments, we demonstrate state-of-the-art performance, surpassing even end-to-end state-of-the-art driving stacks.

Furthermore, we explore the advantages offered by end-to-end learning while preserving the benefits of the modular pipeline approach. We introduce a variant of the Perception PlanT architecture, termed E2E PlanT, where we tread a lightweight object detector as a fixed pre-processing step for LiDAR data and then set up an end-to-end sensor fusion framework for images and detected objects. By incorporating this variant into our framework, we aim to combine the strengths of end-to-end learning with the interpretability and modularity of the PlanT pipeline.

Throughout our work, we provide valuable insights into different parameter settings, allowing for a comprehensive understanding of their influence on system performance. Additionally, we conduct extensive studies that specifically highlight the impact of our proposed improvements, shedding light on the key factors contributing to the advancements achieved. In summary, this thesis aims to enhance modular pipeline approaches for autonomous driving by focusing on improving the object detection component. By narrowing the performance gap between ground truth detections and object detection, as well as outperforming state-of-the-art end-to-end driving stacks on the LAV benchmark, we contribute to the development of more robust and accurate autonomous driving systems. Our exploration of end-to-end advantages while retaining the modular pipeline framework further demonstrates the potential for synergistic integration of these approaches. Through extensive studies and evaluations, we provide valuable insights into the effectiveness of our proposed enhancements, paving the way for future advancements in autonomous driving technologies.

The rest of this work is structured as follows: in Chapter 2 we discuss existing approaches to self-driving, including modular pipeline, end-to-end approaches and evaluation of self-driving stacks. We also provide an overview of the evaluation of self-driving models and describe the data and algorithms used in object detection. Furthermore, we highlight important advancements in the field of Object Detection. In Chapter 3, we provide a more detailed description of the problem setting, offer preliminary background information, and introduce the two models we propose in this work: SECOND PlanT and E2E PlanT. Chapter 4 focuses on describing our experimental setup for both offline and online experiments, implementation details, baseline models, and presents quantitative and qualitative results. Finally, in Chapter 5, we provide a summary of the thesis and outline directions for future research.

2 Related Work

This chapter focuses on the task of self-driving and object detection. Two different approaches are explored for self-driving: a modular pipeline and end-to-end learning. Section 2.1 provides a detailed discussion of previous work conducted on these approaches. Furthermore, in Section 2.2, an extensive analysis of object detection is carried out, focusing specifically on 3D object detection in the context of self-driving.

2.1 Approaches to Self-Driving

The development of an autonomous driving system revolves around creating a function that takes in high-dimensional sensory input, such as an RGB image, RGB video, LiDAR laser scans, radar data, or a combination of these, and generates a low-dimensional output that includes lateral control (steering angle) and longitudinal control (acceleration and braking). The primary objective is to find a function that can reliably and robustly perform this mapping. In pursuit of this goal, two main approaches have emerged as the dominant methods for addressing the challenges of self-driving: modular pipelines and end-to-end systems.

2.1.1 Modular Pipeline

The modular pipeline approach in self-driving technology involves breaking down the entire system into independent components. These interconnected modules often consist of perception, planning, and control [120], but can be extended to even more complex pipelines. In a basic modular pipeline, the perception module processes the input data, providing information about the environment to the planner. The planner, using the output from the perception module, the current location, and high-level driving commands, determines the appropriate route, generates waypoints, and finally, the controller utilizes these waypoints to produce driving commands.

Modular pipelines have been explored since 2006 [100] and extensively discussed [54, 74, 86]. They have been widely adopted by prominent companies in the real-world industry, including Waymo [111], Uber [103], Tesla [98], NVIDIA, and Zoox [127]. The modular approach allows for independent and parallel development of each component, enabling faster progress and enhancing the interpretability of the system. This interpretability is beneficial for safety guarantees as humans can understand and analyze the individual components. Due to the modularity, research can be specifically done on independent modules. Research on planners has shifted from rule-based planning [99, 28, 53] towards learning-based planners [16, 21, 106, 82], mostly explored with access to privileged information and without the usage of a full modular pipeline. Datasets such as Waymo [93], nuScenes [8], and KITTI [35, 34] were introduced to specifically advance standalone modules such as detection and perception.

Although modular pipelines have been extensively studied and adopted in the industry, recent research has primarily focused on end-to-end driving stacks in a closed-loop setting. While more advanced modular stacks have been developed recently [44, 43, 122], they have not yet demonstrated stronger performance on benchmarks such as the CARLA challenge. In the CARLA challenge, Pylot [37] was the first model ranked on the CARLA Leaderboard [10] using a modular pipeline an more recent modular pipeline models [85, 84] barley improved the performance. Perception PlanT, introduced in [82], shows much stronger performance while still being lackluster compared to PlanT [82], which uses privileged information instead of a perception module. It is worth noting that concurrent work, SMS [112], showcased promising results on CARLA while utilizing a modular pipeline.

However, modular pipeline systems also have certain disadvantages. One significant drawback is their inability to facilitate end-to-end training. Consequently, the parameters of individual components cannot be fine-tuned on the downstream objective of driving. Instead, these systems rely on offline evaluations, which can deviate from online evaluations [72]. In addition, modular pipeline systems are prone to error propagation [66].

In our work, driven by the prevalence of modular pipeline approaches in the industry, our objective is to narrow the performance gap between modular pipelines and end-to-end systems in the CARLA challenge. To achieve this, we introduce SECOND PlanT, a modular pipeline that integrates the SECOND object detector [115], a lightweight detection network, the PlanT planner [82], and two PID Controllers. Through our approach, we showcase substantial advancements over previous modular pipelines within the CARLA Simulator, thereby addressing the performance limitations.

2.1.2 End-to-End

An alternative approach to the traditional modular pipeline is the utilization of a single component that directly maps the sensory input to the control output, typically implemented as a neural network. In contrast to the modular pipeline, this approach involves training the network end-to-end, directly optimizing for the desired driving metrics. Most end-to-end methods make use of imitation learning (IL) to mimic expert driving [2], or use reinforcement learning (RL) [95].

The foundation for the end-to-end approach was laid by the pioneering work of ALVINN in 1989 [76], and since then, it has remained an active research area in the context of autonomous driving [47]. Numerous studies have contributed to the advancements of end-to-end models [76, 70, 6, 24, 4, 41, 69, 49, 121, 16, 25, 11], and currently, the CARLA leaderboard is dominated by IL-based [16, 21, 113, 15, 87] and RL-based [13, 14, 101] end-to-end approaches, with the former dominating the leaderboard in recent times.

Compared to the modular pipeline approach, end-to-end models often struggle with generalization to unseen scenarios that differ from the training data and lack interpretability. Additionally, debugging and identifying errors within the system can be more challenging due to the model mostly being treated as a black box.

However, there are several advantages associated with the end-to-end approach. Firstly, it allows for the direct optimization of the neural network for the driving task, making it end-to-end trainable. This means that the entire system can be optimized holistically, leading to potential performance improvements [46, 44]. Additionally, the process of obtaining annotations for training data is relatively inexpensive compared to the modular pipeline. One common approach for labeling data involves attaching sensors to the car and recording the driver's actions while they are driving. Other approaches are even cheaper and utilize privileged experts to generate data inside a simulator [21, 15].

In our work, we explore the combination of the modular approach with end-to-end training and propose E2E PlanT, a variation that aims to combine the advantages of end-to-end training with the benefits of the modular pipeline by providing explainability via transformer attention of different objects.

2.1.3 Evaluation

Evaluating self-driving models is complex and critical, with challenges including high development costs, and safety concerns. Special testing facilities or safety drivers are necessary to mitigate the risks associated with testing autonomous systems. To evaluate a self-driving model, it is crucial to expose it to diverse driving scenarios and weather conditions, often requiring extensive distances. One approach to mitigate this challenge is to focus on individual tasks within modular pipelines, such as object detection, instead of evaluating the entire system. Another approach is to compare the system's behavior to recorded human driver decisions in open-loop evaluations. However, such methods do not always correlate well with real-world driving performance [23].

An alternative is closed-loop evaluations in driving simulators, where the self-driving system controls a simulated car and its performance is assessed based on simulator feedback. Simulators offer cost-effective evaluation, precise control over driving conditions, and testing of dangerous scenarios without real-world consequences. The CARLA (CAR Learning to Act) Simulator [30] is a notable free and open-source platform for self-driving technology. It supports various sensors, offers realistic graphics, and includes a leaderboard [10] for objective comparisons among teams on secret test routes hosted externally. This facilitates fair evaluations and provides a measurable benchmark for assessing progress in the self-driving community.

Currently, no method has achieved optimal driving under the novel conditions of the CARLA leaderboard. Nonetheless, the self-driving community is steadily progressing towards this goal. Utilizing the CARLA simulator and its evaluation capabilities, researchers and developers can refine autonomous driving approaches in a controlled environment, fostering innovation and advancing the field. In this thesis, we utilize the CARLA simulator for our research and evaluation purposes.

2.2 Object Detection

Perception systems play a crucial role in the modular pipeline variants of autonomous driving systems. These systems commonly rely on multi-modal data, including images from cameras, point clouds from LiDAR scanners, HD maps, and more. Various tasks are encompassed within a perception system, such as object detection, object tracking, lane detection, and semantic segmentation. High-quality perception results serve as reliable observations for the following steps such as object tracking, trajectory prediction, and path planning. In this thesis, our primary focus will be on 3D object detection, which involves localizing, identifying the size, and classifying other vehicles within a given 3D scene.

Before the breakthrough in deep learning-based detection methods [45] object detection algorithms were not sufficiently advanced to be effectively employed in autonomous driving applications, despite the introductions of successful detectors [26, 105] in the early 2000s. In recent years, 3D object detection for autonomous driving has made significant progress and continues to be a highly studied field [128, 65, 78, 3, 61].

While 2D object detection [62, 48, 36] focuses on generating 2D bounding boxes on images and serves as a fundamental problem in computer vision, our research primarily centers around 3D detection. Accurate localization in 3D space is crucial for driving scenarios, making 3D detection more complex than its 2D counterpart. Furthermore, we narrow down our focus to object detection methods that utilize image, LiDAR, or a combination of both as inputs.

2.2.1 Image-Based 3D Object Detection

Image-based 3D object detection aims to predict the objects using one or more images produced by cameras attached to the vehicle. *Monocular* object detection methods produce detections using only a single image [125, 7, 80, 91, 55, 57, 124]. Since depth estimation is difficult for monocular methods, some methods make use of depth estimation as an auxiliary task [109, 114, 29]. Additionally, certain methods leverage prior knowledge of object shapes through pre-training or auxiliary tasks [17, 68, 12]. *Stereo-based* 3D object detection methods [18, 56, 119] utilize two camera images captured simultaneously, leveraging additional geometric constraints to infer more accurate depth information. However, achieving accurate calibration and synchronization in real-world applications can be challenging [65]. *Multi-view* 3D object detection methods have recently experienced rapid development, driven by increasing computational capabilities. Among these methods, the most successful approaches [110, 58, 63] aim to generate object queries from the bird's-eye view (BEV) representation, which is constructed by projecting multi-view images onto the BEV plane. These approaches leverage the advancements made in object detection using Transformers [104] to incorporate camera view features and improve detection accuracy.

2.2.2 LiDAR-Based 3D Object Detection

LiDAR-based 3D object detection operates on LiDAR data, i.e. point clouds or range images. Point clouds are large, sparse, and irregular compared to images, requiring special model designs to achieve real-time inference. In practice, LiDAR-based object detectors currently outperform all image-based methods in the task of 3D object detection [65]. Figure 2.1 provides a chronological overview of the most common LiDAR-based object detectors in the previous 8 years categorized by the following groups:

Point-based 3D object detection methods [89, 117, 90, 75] make use of a point-based backbone network, where points are gradually sampled and features are learned through point cloud operators. The 3D bounding boxes are then predicted based on the downsampled points and features. *Grid-based* 3D object detectors [108, 116, 126, 115, 52, 118] first rasterize point clouds into discrete grid representations, such as voxels, pillars, and BEV feature maps. They then apply conventional 2D CNNs or 3D sparse NNs to extract features from the grids. Finally, 3D objects are detected using the BEV grid cells. *Point-Voxel based* 3D object detectors [88, 64, 96, 39, 20] employ a hybrid architecture that leverages both points and voxels for object detection, often resulting in improved performance but higher complexity and inference time. *Range-based* 3D object detectors [67, 33, 94] leverage range images which is a dense and compact 2D representation, produced by the LiDAR, in which each pixel contains 3D distance information instead of RGB values. Since range images are 2D representations, they leverage 2D object detection models to handle range images.

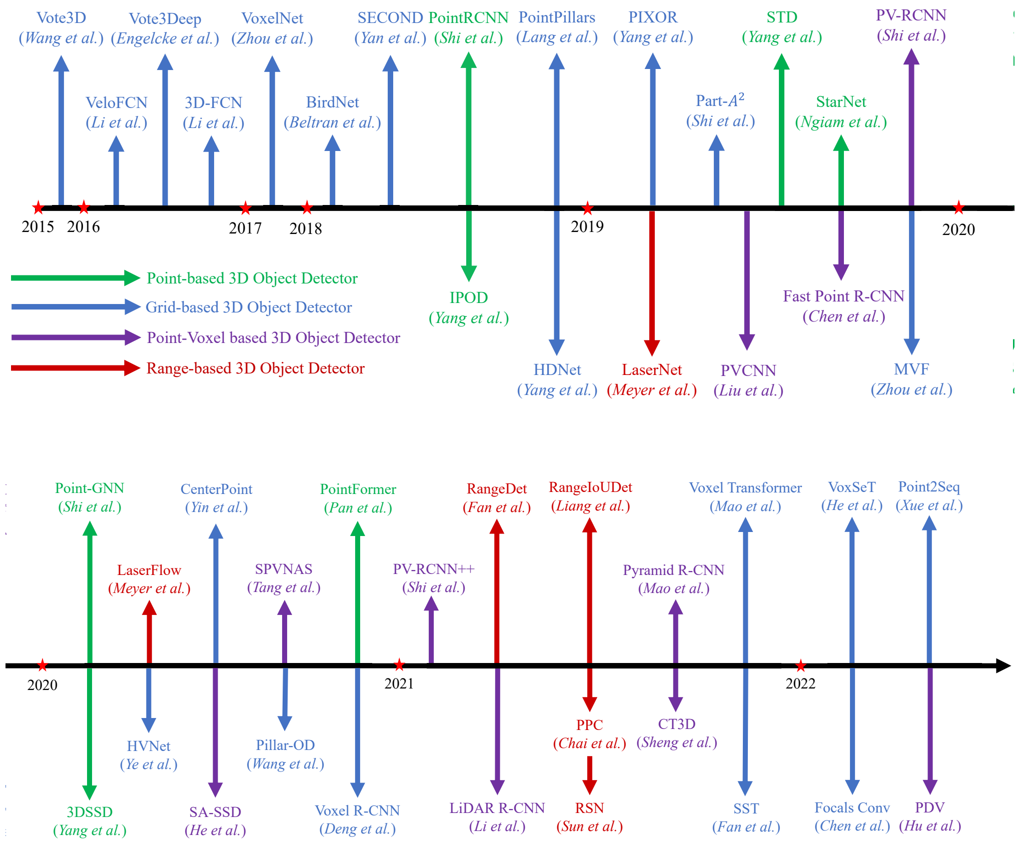


Figure 2.1: **3D Object Detection Research.** Chronological overview of the LiDAR-based 3D object detection methods. Adapted from [65].

2.2.3 Multi-Modal 3D Object Detection

Multi-modal (also known as sensor-fusion) 3D object detection methods fuse multiple sensory inputs. These approaches can be divided into LiDAR-camera, radar and map fusion-based methods. *LiDAR-Camera Fusion* models [19, 77, 50, 107, 59] leverage the advantages of both sensors. While Cameras provide color information, LiDAR specializes in 3D localization. Due to the superior performance of LiDAR-based detectors, most architectures incorporate image information at different stages of the LiDAR detection pipeline. More information on the radar/map fusion can be found in [65].

In our work, we specifically focus on LiDAR-based object detection due to its superior performance in 3D object detection tasks [65]. Among the various LiDAR-based methods, we narrow down our focus to the SECOND (Sparsely Embedded

CONvolutional Detection) approach. SECOND is a grid-based method that offers a lightweight solution by projecting the 3D space into a 2D representation, specifically a Bird's Eye View (BEV) feature map. Furthermore, we choose not to employ sensor fusion in our approach, aiming to ensure real-time inference speed. By utilizing a single sensor modality and adopting a lightweight grid-based method like SECOND, we strive to strike a balance between accuracy and efficiency in LiDAR-based object detection for autonomous driving applications.

3 Method

In this Chapter, we give a detailed explanation of the methods we use throughout the thesis. We begin by introducing the problem setting in Section 3.1. Next, we provide necessary preliminaries to ensure that the reader is familiar with the subsequently used techniques in Section 3.2. Finally, our main contributions are presented in the last two Sections: in Section 3.3, we introduce SECOND PlanT, our new and improved version of Perception PlanT [82], and in Section 3.4 we present a novel end-to-end adaptation of the original PlanT [82] architecture.

3.1 Problem Setting

We consider the task of driving in an urban environment, where the objective is for the agent to safely navigate from a starting point through one or more predetermined target points to a destination. During the journey, the agent must prioritize safety by avoiding collisions, staying within its designated lane, and adhering to traffic regulations like stop signs and traffic lights. Throughout the course of the drive, the agent encounters various challenging scenarios, which are elaborated upon in greater detail in Section 4.1. We differentiate between types of agents:

(1) A **sensorimotor** agent relies solely on sensor data such as LiDAR, cameras, Global Positioning System (GPS), a speedometer, high-definition map (HD Map) and inertial measurement units (IMU) as input. The agent's output consists of the steering, throttle and brake controls of the car.

(2) A **privileged agent** not only has all the sensor data available but also has direct access to a simulator, allowing it to gather precise information about the positions of all other vehicles, as well as the locations and current states of traffic lights and locations of stop signs. Similar to the sensorimotor agent, the output of the privileged agent includes the steering, throttle and brake controls of the car.

For both types of agents, we assume they have access to high-level GPS target points provided by the simulator. These target points are sparsely distributed and depend on the starting location and destination, potentially spanning hundreds of meters apart. In certain instances, we conduct experiments in a sensorimotor setting where we provide the agent with ground truth information about traffic lights. In this case, we will mention and specify the scenario accordingly.

3.2 Preliminaries

In this Section we provide a brief overview of key concepts and techniques essential to understand the subsequent Chapters.

3.2.1 SECOND

SECOND (Sparsely Embedded CONvolutional Detection) is a grid-based 3D object detection network (see Section 2.2.2) which was first introduced in [115]. It utilizes spatially sparse convolutional networks to extract z-axis information from the point cloud data and then samples down the point cloud into a representation similar to 2D image data. Additionally, a new angle loss function is introduced to improve the orientation estimation. This is done by addressing the issue where the bounding box flipped by 180 degrees leads to large losses even though the bounding box is accurately representing the object. Finally, SECOND features a novel data augmentation method for point clouds to enhance the convergence speed during training and overall performance.

Architecture. SECOND's architecture is illustrated in Figure 3.1 and can be divided into three main steps:

(1) Point cloud grouping: The raw point cloud is first cropped based on the object distribution in the dataset. Subsequently, an iterative process is employed to assign points to 5k-8k voxels.

(2) Feature extraction: The feature extraction process consists of two stages. First, two Voxel Feature Encoding (VFE) layers, as introduced in [126], and a linear layer are applied. Afterwards, sparse convolutional layers are employed for further processing.

Sparse convolutions tackle the issue that the third dimension introduced by the point clouds (compared to 2D images) increases the complexity of the calculation significantly. They exploit that most voxels in a 3D point cloud are empty and introduce an efficient GPU calculation schema.

(3) Detection generation: Similar to two-stage Region Proposal Networks (RPN) (see Section 3.2.2), SECOND utilizes Single Shot Multibox Detectors (SSD) [62] to generate the final detections in one stage. While RPN is used in two-stage object detectors that initially generate class-agnostic proposals, SSD employs a predefined set of bounding boxes (anchors) at various scale and aspect ratios to predict object classes and bounding box offsets in a single stage. By discretizing the feature space into the predefined anchors, it becomes possible to process all of them in parallel. After inputting the generated feature maps from (2), additional convolution, batch normalization, and a ReLU layer are applied to the extracted feature maps. Finally, the box is classified, its location is regressed, and the direction of the bounding box is decided by binary classification.

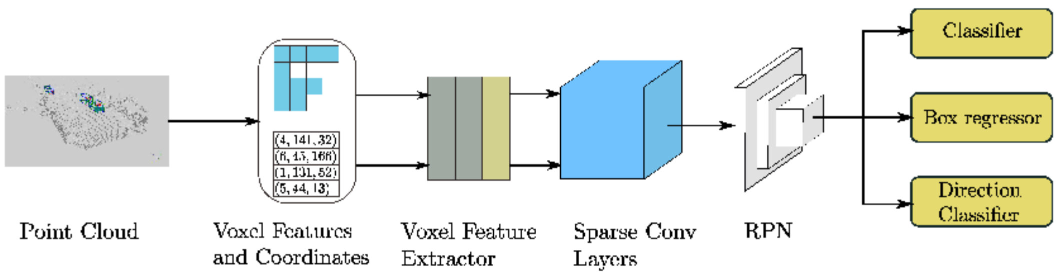


Figure 3.1: **SECOND architecture.** The model receives the raw point cloud as input and generates regressed bounding box proposals with an object class, and a binary classified direction. Image taken from [115].

Loss Function. To account for the predictions of the three detection heads, SECOND employs a loss function, which consists of four components:

(1) *Angle loss*. Previously, VoxelNet [126] directly predicted the radian offsets. However, this approach was causing issues when comparing two boxes with offset angles of 0 and π, respectively. Although such boxes look almost identical, the loss function produces a large value. To address this, SECOND proposes a new loss function angle loss $\mathcal{L}_\theta$, which makes use of the smooth L_1-Loss:

$$\text{Smooth-L}_1(x) = \begin{cases} \frac{0.5x^2}{\beta} & \text{if } |x| < \beta, \\ |x| - 0.5\beta & \text{otherwise.} \end{cases}$$

The angle loss is then defined as:

$$\mathcal{L}_\theta = \text{Smooth-L}_1(\sin(\theta_p - \theta_t)),$$

where θ_p is the predicted orientation angle θ_t the true orientation angle. If θ_p and θ_t differ by a value close to π, the roots of the sine help to keep the loss low. Let $|\theta_p - \theta_t| \approx \pi$, without loss of generality assume that $\theta_p + \pi \approx \theta_t$. Then, $\sin(\theta_p - \theta_t) \approx \sin(\pi) = 0$ and thus $\mathcal{L}_\theta \approx 0$.

(2) *Direction Loss*. To learn the direction the box faces, a binary classification is trained using the cross-entropy loss $\mathcal{L}_{\text{dir}}$.

(3) *Classification Loss*. To address the imbalance between positively and negatively classified bounding boxes, focal loss $\mathcal{L}_{\text{cls}}$ [60] is used to train the classification head.

(4) *Regression Loss*. The regression loss $\mathcal{L}_{\text{reg}}$ for the location and extent of the bounding boxes is also included.

The combined loss function is defined as:

$$\mathcal{L}_{\text{total}} = \beta_1 \mathcal{L}_{\text{cls}} + \beta_2(\mathcal{L}_\theta + \mathcal{L}_{\text{reg}}) + \beta_3 \mathcal{L}_{\text{dir}}$$

Data Augmentation. To improve the convergence speed and robustness of the detector, a novel data augmentation technique is introduced. At the start, a database containing all the ground truth boxes and their associated points is generated. During training, random entries from the database are selected and added to the scene, thereby increasing the number of ground truth points per training sample. A collision test is performed to avoid physically impossible outcomes. Additionally, a random rotation and linear transformation are applied to each object. Finally, global scaling of the world is applied.

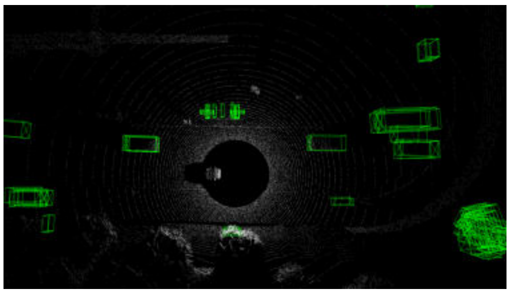
3.2.2 Region Proposal Network (RPN)

Region Proposal Network (RPN) [81] introduce a proposal-based approach to object detection and belong to the category of two-stage object detectors. A region proposal identifies potential object locations within an image, allowing subsequent classification tasks to focus on these regions rather than the entire image. This reduces computational requirements and improves overall efficiency. The RPN is an end-to-end trainable neural network. The input of the RPN is a feature map of a CNN Backbone, and the output is a set of rectangular object proposals, each with an objectness score.

The key idea behind RPN is to treat the task of object proposal generation as a binary classification problem. Instead of relying on handcrafted features or predefined anchor boxes, the RPN learns to generate proposals by simultaneously predicting objectness scores and refined bounding box coordinates. The infinite set of all possible bounding boxes is approximated by a finite set of suitable boxes and the induced error is handled by regressing the box offsets later on. This is achieved by using a sliding window with an objectness classifier. The classifier produces a probability $p \in [0, 1]$ that predicts if there is an object in the given sliding window. To each sliding window, k corresponding anchor boxes are proposed. The anchors are all larger than the sliding window due to the spatial downsampling of the CNN and thus the resulting larger receptive field. After sliding over the entire window, Non-Maximum Suppression (see Section 3.2.3) is used to remove redundant predictions before regressing the location and extent of the bounding box to further fit the object. The RPN is trained to optimize the objectness classification and the box regression simultaneously.

3.2.3 Non-Maximum Suppression (NMS)

Non-Maximum Suppression (NMS), first introduced as an edge detection technique in [83], is a post-processing technique widely used in object detection algorithms. It addresses the problem that during the object detection process, multiple bounding boxes for the same object may be generated due to variations in scale, orientation, or overlapping regions. The primary goal of NMS is to select the most confident and representative object detection results while discarding redundant or overlapping ones. By eliminating duplicate detections, NMS helps to improve precision and to reduce the number of false positives.


(a) Object predictions before NMS

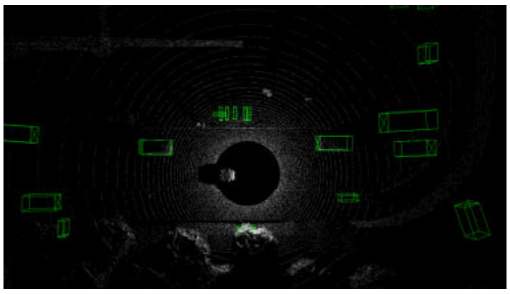

(b) Object predictions after NMS

Figure 3.2: Visualization of the effect of Non-Maximum Suppression.

Algorithm 1 Pseudocode for Non-Maximum Suppression (NMS)

```
    Input: $\mathcal{B} = \{b_1, \dots, b_N\}, \mathcal{S} = \{s_1, \dots, s_N\}, \lambda$
1:  $\mathcal{D} \leftarrow \emptyset$
2:  while $\mathcal{B} \neq \emptyset$ do
3:      $m \leftarrow \arg\max \mathcal{S}$
4:      $\mathcal{M} \leftarrow b_m$
5:      $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{M}$
6:      $\mathcal{B} \leftarrow \mathcal{B} - \mathcal{M}$
7:      for $b_i \in \mathcal{B}$ do
8:          if $\text{IoU}(\mathcal{M}, b_i) \geq \lambda$ then
9:              $\mathcal{B} \leftarrow \mathcal{B} - b_i$
10:             $\mathcal{S} \leftarrow \mathcal{S} - s_i$
    return $\mathcal{D}, \mathcal{S}$
```

Given a set of objects, NMS identifies the detection with the highest confidence score and removes the overlapping ones that have significant overlap with the selected one. The overlap between two detections is typically quantified using the Intersection over Union (IoU) metric (see Section 4.3.3), which measures the ratio of the shared area to the total area covered by both detections. Repeating this process until all overlapping proposals with a low score are gone yields the filtered result list. A pseudocode example for NMS is displayed in Algorithm 1, where we refer to $\mathcal{B} = \{b_1, \dots, b_N\}$ as the original box proposals, $\mathcal{S} = \{s_1, \dots, s_N\}$ as their corresponding scores and λ as the IoU overlap threshold. Even with a task-specific, well-tuned overlap threshold, sometimes NMS runs into the issue of removing a correct bounding box, due to the actual objects overlapping in the corresponding scene. Improvements like Soft-NMS [5] tackle this issue by gradually reducing the confidence scores of overlapping detections instead of directly deleting them. An illustration of NMS is displayed in Figure 3.2.

3.2.4 (Conditional) Imitation Learning & Behavior Cloning

Hard coding and creating rule-based policies can be a difficult task that does not scale well with large amounts of data. Hence, a data-driven approach is desirable. We define the policy as a parameterized function π_θ, which in practice often is represented by a Neural Network. Mathematically, this can be formulated as finding the optimal parameter θ^* that minimizes the expected loss:

$$\theta^* = \arg\min_\theta \mathbb{E}_{s\sim P(s|\pi_\theta)} \left[\mathcal{L}(\pi^*(s), \pi_\theta(s))\right],$$

where the state $s \in \mathcal{S}$ may be only partially observed and given by the environment, the action $a \in \mathcal{A}$ may be discrete or continuous (e.g. a steering angle or gas/brake), the parameterized policy $\pi_\theta : \mathcal{S} \to \mathcal{A}$ produces an action given a state, the optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$ provided by an expert, the state dynamics $P(s_{i+1}|s_i, a_i)$ which in a simulator often is deterministic and a feasible loss function $\mathcal{L}(a^*, a)$, where $a^* = \pi^*(s)$ is the optimal action provided by the expert.
Since we have to do a rollout of the current policy π_θ, a pure imitation learning approach can be inefficient. This is because all training has to be done in an online fashion by doing a rollout determined by the current policy π_θ.
In practice, this is either slow or infeasible since we do not have permanent access to an expert thus, often use the more simple **Behavior Cloning** instead. Formally, the objective of Behavior Cloning can be written as:

$$\theta^* = \arg\min_\theta \mathbb{E}_{(s^*,a^*)\sim\mathcal{D}} \left[\mathcal{L}(a^*, \pi_\theta(s^*))\right],$$

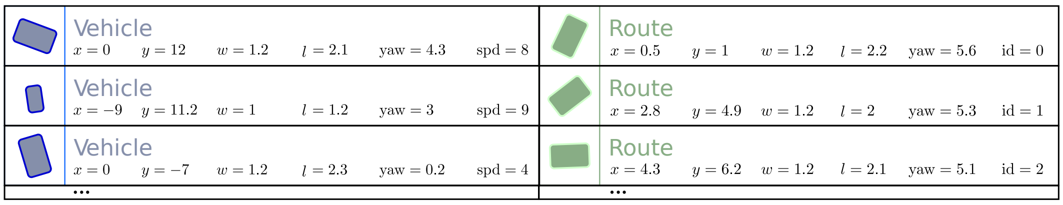
where we have direct access to an optimal action a^*, given a state s^* sampled from a dataset $\mathcal{D}$ that contains state action pairs. In the context of self-driving, this could be easily obtained by recording a human driver's actions with sensors. We have thus reduced the optimization problem to a supervised learning problem.

There exist some scenarios, especially in the context of self-driving, where standard Imitation Learning or Behavior Cloning is not sufficient. One such scenario could be a junction, in which it is feasible to go straight or turn right or left. So, we need an additional high-level navigational command to determine the correct action. To avoid this ambiguity **Conditional Imitation Learning (CIL)** [24] uses an additional navigational command c as input, e.g. $c \in \{\text{left, right, straight}\}$ that can be provided by a GPS. This removes the task ambiguity induced by the environment. Formally, we extend the dataset by adding a navigation command c to each state-action pair, yielding:

$$\theta^* = \arg\min_\theta \mathbb{E}_{(s^*,a^*,c^*)\sim\mathcal{D}} \left[\mathcal{L}(a^*, \pi_\theta(s^*, c^*))\right].$$

3.2.5 PlanT

PlanT is a learning-based planner originally introduced in [82]. It represents other vehicles and the route of the ego vehicle in a given driving scene using tokens and



Vehicle	$x = 0$	$y = 12$	$w = 1.2$	$l = 2.1$	yaw $= 4.3$	spd $= 8$	Route	$x = 0.5$	$y = 1$	$w = 1.2$	$l = 2.2$	yaw $= 5.6$	id $= 0$
Vehicle	$x = -9$	$y = 11.2$	$w = 1$	$l = 1.2$	yaw $= 3$	spd $= 9$	Route	$x = 2.8$	$y = 4.9$	$w = 1.2$	$l = 2$	yaw $= 5.3$	id $= 1$
Vehicle	$x = 0$	$y = -7$	$w = 1.2$	$l = 2.3$	yaw $= 0.2$	spd $= 4$	Route	$x = 4.3$	$y = 6.2$	$w = 1.2$	$l = 2.1$	yaw $= 5.1$	id $= 2$
...							...						

Figure 3.3: **Object Tokens.** Example of how the tokenization for vehicles and route segments could look like.

utilizes a transformer (see Section 3.2.6) along with a GRU (see Section 3.2.7) to predict ego vehicle waypoints and future positions of other vehicles. The model is trained through imitation learning (see Section 3.2.4) on data generated in the CARLA Simulator [30] by a rule-based, privileged expert adapted from [21]. While PlanT primarily focuses on the planning aspect of the modular pipeline with data-driven learning, the authors also conduct ablation studies by incorporating a perception module. This module, known as **Perception PlanT**, generates vehicle tokens and the traffic light state solely from sensor data. Additional details on Perception PlanT are provided in Section 4.5.4. The authors refer to the original model that relies on ground truth information as **Privileged PlanT**. The overall architecture of PlanT is illustrated in Figure 3.4.

Tokenization. PlanT represents the route and the objects in a given scene using tokens, where each token corresponds to a two-dimensional bounding box in bird's-eye view (BEV) space. Figure 3.3 provides some example tokens. Formally, the set of vehicles $\mathcal{V}^t \in \mathbb{R}^{V_t \times 6}$ and the set of route segments $\mathcal{S}_t \in \mathbb{R}^{S_t \times 6}$ are extracted from the ground truth vehicles and an HD-Map, respectively. At timestep t, each object $\boldsymbol{o}_{i,t} \in \mathcal{V}^t \cup \mathcal{S}_t$ is represented as a vector $\boldsymbol{o}_{i,t} = \{z_{i,t}, x_{i,t}, y_{i,t}, \phi_{i,t}, w_{i,t}, l_{i,t}\}$, where $(x_{i,t}, y_{i,t})$ represents the position of the bounding box, $\phi_{i,t} \in [0, 2\pi]$ its orientation, and $(w_{i,t}, l_{i,t})$ its extent and $z_{i,t}$ is an object-type specific attribute. For vehicles, $z_{i,t}$ represents the speed, while for route objects, it indicates the ordering of the route segments starting from 0. For Privileged Plant, the vehicle tokens are extracted directly from the simulator, while in Perception PlanT a perception module is employed to generate the vehicle bounding boxes. In both cases, only vehicles within a distance of D_{max} or closer to the ego vehicle are tokenized. The route points are either directly provided by CARLA or predicted. The route points are then preprocessed by subsampling using the Ramer-Douglas-Peucker algorithm [79, 31]. In the next step, the points are extended to a 2D bounding box by using the Euclidean distance to the next route point as the length and the lane width as the width. Finally, the number of segments is reduced to $N_s \leq S_t$.

Token Embeddings. As explained in Section 3.2.6, learnable embeddings for each input token are crucial when using a transformer backbone. A linear projection $\rho : \mathbb{R}^6 \to \mathbb{R}^H$ is applied, where H is the desired hidden dimensionality. Learnable

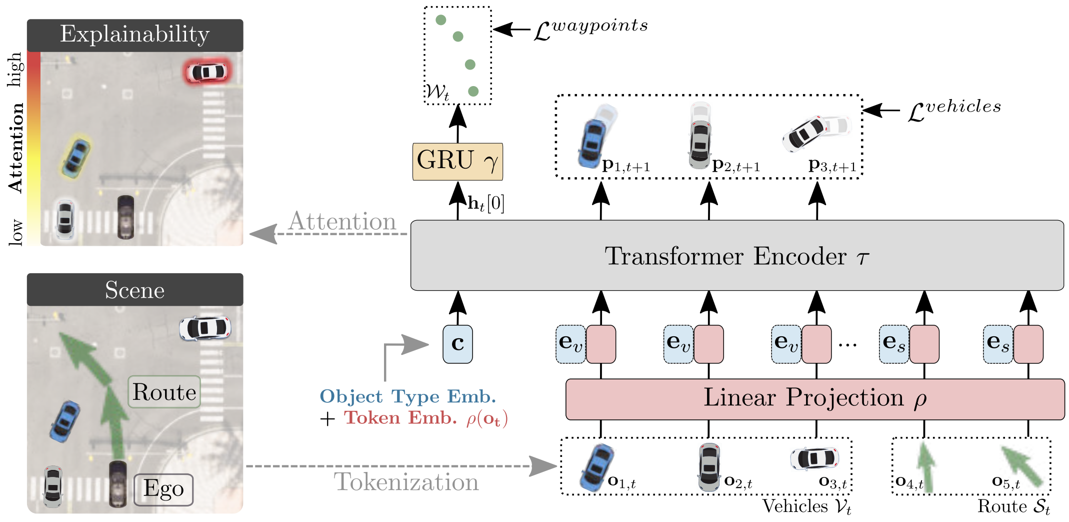


Figure 3.4: **PlanT Architecture [82].** The scene is tokenized and projected using a linear layer ρ. In the next step, learnable object-type embedding vectors are added, and the resulting vectors are processed using a transformer encoder τ. Finally, PlanT generates future waypoints using a GRU decoder γ and performs an auxiliary task to predict the future location, orientation, and speed of the other vehicles.

embedding vectors $\boldsymbol{e}_v, \boldsymbol{e}_s \in \mathbb{R}^H$ are added to the projected tokens, resulting in:

$$\boldsymbol{e}_{i,t} = \begin{cases} \rho(\boldsymbol{o}_{i,t}) + \boldsymbol{e}_v, & \forall \boldsymbol{o}_{i,t} \in \mathcal{V}_t \\ \rho(\boldsymbol{o}_{i,t}) + \boldsymbol{e}_s, & \forall \boldsymbol{o}_{i,t} \in \mathcal{S}_t \end{cases}$$

Waypoint Prediction. The primary task of PlanT is to predict the future waypoints of the ego vehicle. The transformer encoder $\tau : \mathbb{R}^{V_t+N_s+1\times H} \rightarrow \mathbb{R}^{V_t+N_s+1\times H}$ is based on the BERT architecture [27]. In addition and also based on [27], a learnable [CLS] token $\boldsymbol{C} \in \mathbb{R}^H$ is stacked on top of the tokens. Along with the token embeddings, it is fed into the transformer τ. The [CLS] token is processed using attention-based aggregation of all the other features before being fed into an auto-regressive waypoint decoder, that uses GRUs $\gamma : \mathbb{R}^{H+1} \rightarrow \mathbb{R}^{4\times 2}$ (see Section 3.2.7). The waypoint decoder is described in more detail in [21, 38]. Before utilizing γ, a binary traffic light flag l_t is appended to the first part of the transformer output $\boldsymbol{h}_t = \tau([\boldsymbol{c}, \boldsymbol{e}_{1,t}, \ldots, \boldsymbol{e}_{V_t+N_s,t}])$. The modified output is then used as the initial hidden state of the decoder. The decoder predicts the future trajectory for the next 4 time steps as a sequence of 2D waypoints $\mathcal{W}_t = (x_{t+1}, y_{t+1}), \ldots, (x_{t+4}, y_{t+4})$ according to the following equation:

$$\mathcal{W}_t = \gamma(l_t, \boldsymbol{h}_t[0]).$$

Vehicle Future Prediction. As an auxiliary task, PlanT also predicts the future attributes of all other present vehicles. These predictions are important for the ego vehicle trajectory and exploiting synergies between the two tasks can be beneficial. Since the extent of all vehicles stays the same, it is excluded from those predictions. All other attributes i (including speed, orientation, and position) are discretized into Z_i intervals since discrete estimates are sufficient for safe driving. For each of those attributes i, the future vehicle tokens are predicted using a linear layer $\psi_i : \mathbb{R}^H \to \mathbb{R}^{Z_i}$, and a softmax function is applied to obtain class probabilities $\boldsymbol{p}_{1,t+1,i}, \dots, \boldsymbol{p}_{V_t,t+1,i}$ of each predicted attribute i of each vehicle:

$$\boldsymbol{p}_{j,t+1,i} = \text{Softmax}(\psi_i(\boldsymbol{h}_{i,t})), \quad \text{where } i = 1, \dots, 6.$$

Loss Function. To combine the waypoint prediction task and the vehicle future prediction task during the imitation learning training, a multi-task setting is employed with a weighted combination of two losses. For the waypoint prediction task, PlanT follows the approach used in [15, 16, 21] by utilizing an L_1-loss with the ground truth future waypoints $\boldsymbol{w}^{gt}$. Since the space of possible positions for all other vehicles is discretized, a cross-entropy loss $\mathcal{L}_{CE}$ is used, by employing a one-hot encoding of the ground truth future vehicle attributes $\boldsymbol{p}^{gt}_{j,t+1}$. The overall loss function is a combination of the two losses, weighted by a factor $\lambda \in [0, 1]$:

$$\mathcal{L} = \underbrace{\frac{1}{4}\sum_{w=1}^{4} \|\boldsymbol{w}_w - \boldsymbol{w}_w^{gt}\|_1}_{\mathcal{L}^{waypoints}} + \underbrace{\frac{\lambda}{V_t}\sum_{j=1}^{V_t}\sum_{i=1}^{6} \mathcal{L}_{CE}(\boldsymbol{p}_{j,t+1,i}, \boldsymbol{p}^{gt}_{j,t+1,i})}_{\mathcal{L}^{vehicles}}.$$

3.2.6 Transformer

The **Transformer** is a deep learning model introduced in [104]. It provided a major breakthrough in the field of Natural Language Processing (NLP) by offering a more efficient and effective alternative to Recurrent Neural Networks (RNNs) for sequence-to-sequence tasks. Since its introduction, transformers have not only been used in NLP but also play a significant role in various other fields, including general object detection [9] and planning in the context of self-driving [82]. Figure 3.5 provides an overview of the Transformer architecture.

The core idea behind the Transformer is to replace the recurrent convolution operations with a self-attention mechanism. The model consists of an encoder and a decoder, both composed of stacked layers of self-attention and feed-forward neural networks. The encoder processes the input sequence, while the decoder uses contextual information to generate a sequence of outputs.

The self-attention mechanism allows the model to weigh the importance of different positions in the input sequence to capture dependencies and relationships between input tokens better. It operates on three vectors for each position: the *query* vector, the *key* vector, and the *value* vector. These vectors are derived from the input sequence

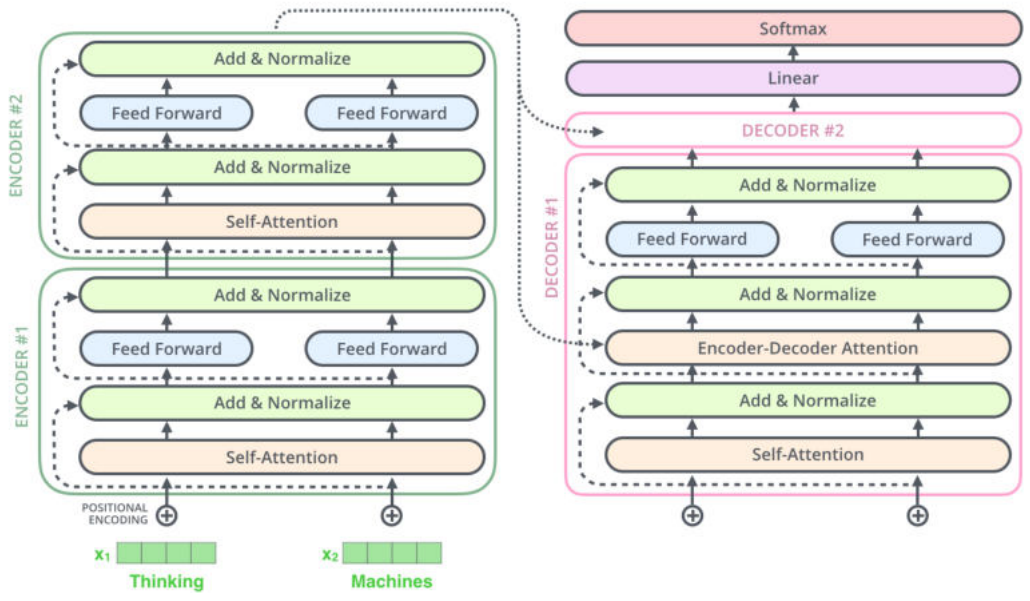


Figure 3.5: **Transformer architecture.** Retrieved from [1].

and are linearly transformed into different vector spaces to capture different aspects of the data. The attention weights are computed by taking the dot product between the query and key vectors, which measure the relevance between positions. The weighted sum of the value vectors is then computed using the attention weights, resulting in the final output of the self-attention mechanism.

A position-wise feed-forward neural network is applied to each position separately and consists of two linear transformations with a ReLU activation function in between.

Additionally, (learnable) token embeddings can be employed. By assigning a dense vector representation to each token, token embeddings allow the model to capture the relationships between different input tokens. In the context of NLP positional encoding is applied, providing crucial information about the position of each token in the sequence.

3.2.7 GRU

The GRU (Gated Recurrent Unit) is a type of RNN introduced in [22]. It is similar to a Long Short-Term Memory (LSTM) [42] but has fewer parameters and includes a forget gate. The GRU architecture comprises recurrent units, also known as *cells*, that are designed to selectively update and reset their internal state based on the input data. It utilizes gating mechanisms to control the information flow within the network, enabling it to retain and update relevant information while discarding

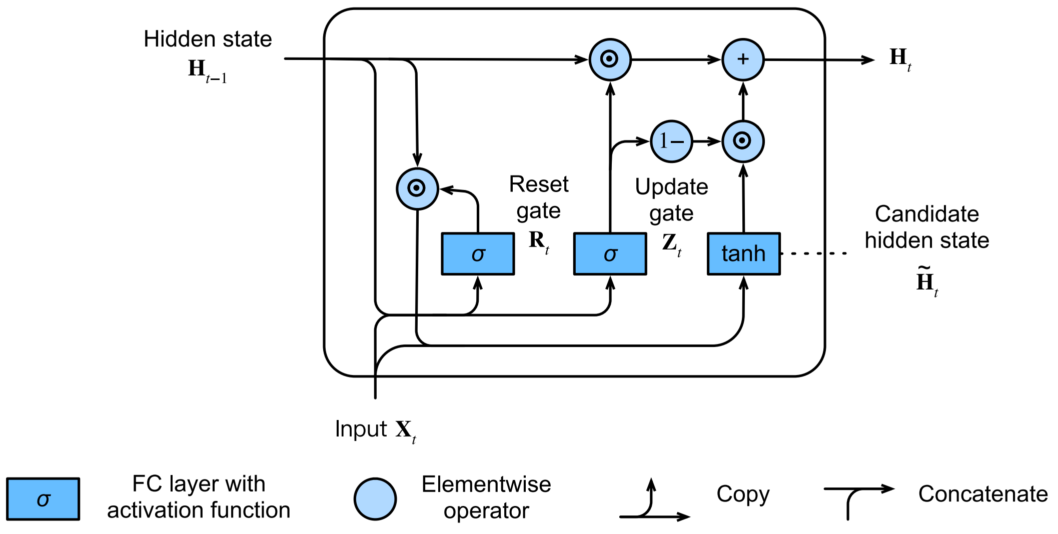


Figure 3.6: **GRU architecture.** Retrieved from [123].

irrelevant or redundant information. The key components of a GRU cell are shown in Figure 3.6. They contain the following:

(1) *Update Gate*: This gate determines to which extent the previous state should be updated and how much of the new state should be added. It takes the input data and the previous hidden state as input and produces an output between 0 and 1. A value of 0 corresponds to no update, while a value of 1 indicates a full update.

(2) *Reset Gate*: This gate determines how much of the previous state should be forgotten or ignored. Similar to the update gate, it produces an output between 0 (complete reset) and 1 (no reset).

(3) *Candidate Activation*: In this step, a candidate activation vector is generated by combining the input data and the previous hidden state, taking into account the reset gate. It calculates the new potential hidden state that could be added to the existing one.

(4) *Hidden State*: This is the output of the GRU cell, which combines the previous hidden state and the new candidate activation. It is calculated by taking the element-wise product of the output of the update gate and the candidate activation, and then adding the element-wise product of `(1-update gate)` and the previous hidden state.

The GRU cells can be stacked to form multiple layers, with the output of one layer serving as the input to the next layer.

3.2.8 PID controller

The **PID (Proportional-Integral-Derivative)** controller is a widely used feedback control mechanism in the field of engineering and automation. It is currently the most commonly used controller in industrial applications due to its simplicity and robustness.
The basic concept behind the PID controller is to measure the error between the desired target value (e.g., the velocity of a car) and the actual output of the system. This error is then used to compute a control signal that minimizes the error over time.

With $e(t)$ the error at timestep t, the PID Controller consists of three elements:

(1) *Proportional (P) Term* $P(t) = K_p e(t)$. It calculates the control signal based on the current error between the target value and the actual output. The control effort is directly proportional to the magnitude of the error. However, using only the P term can result in oscillation and overshooting of the target.

(2) *Integral (I) Term* $I(t) = K_i \int_0^t e(t')dt'$. It applies a correction to the control signal by accumulating past errors over time.

(3) *Derivative (D) Term* $D(t) = K_d \frac{\partial e(t)}{\partial t}$. It takes into account the rate of change of the error over time. It acts as a damping component that can anticipate and counteract abrupt changes in the system's behavior. The derivative term stabilizes the system's response, reduces overshoot, and improves the controller's responsiveness.

Combining all three elements with their constants $K_p, K_i, K_d > 0$ yields the correcting variable $u(t)$ at timestep t:

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{\partial e(t)}{\partial t}.$$

In the context of self-driving, PID controllers are often split into longitudinal and lateral components. For example, the longitudinal controller introduced in [16] predicts the target speed v based on the average velocity that the vehicle needs to pass through all waypoints:

$$v = \frac{1}{K} \sum_{k=1}^{K} \frac{||w_k - w_{k-1}||_2}{\triangle t},$$

where $\triangle t$ is the temporal spacing between waypoints, $w_0 = [0, 0]$ the position of the ego vehicle, and all other 2D waypoints $w_1, \ldots, w_K$ represent the predicted or given trajectory of the ego vehicle.

3.3 SECOND PlanT

Building upon the work done for Perception PlanT, we propose **SECOND PlanT** an improved Perception PlanT variant that addresses the performance differences between Privileged and Perception PlanT, as shown in [82]. As discussed in Section 3.2.5, the authors of [82] performed an ablation study that specifically excluded the use of privileged information. Instead, they utilized a perception module combined with the planner called *Perception PlanT*. The perception module is based on CenterNet [125] and takes an RGB image and 180° LiDAR input that is projected into BEV space. The detailed hyperparameters and numbers are described in Section 4.5.5. We improved the previous model in three key areas:

Firstly, the LiDAR input has been expanded from 180 degrees to a full 360 degrees, providing crucial information about the vehicles located behind the ego vehicle. This expansion of the LiDAR input is particularly beneficial in scenarios such as lane merging, as it helps prevent collisions.

Secondly, we introduce a new object detector SECOND [115] (see Section 3.2.1), which exclusively uses LiDAR data. This eliminates the computationally intensive processing of RGB data, saving inference time and enabling real-time predictions. Additionally, SECOND introduces a novel angle loss that facilitates learning and enhances the accuracy of bounding box orientation predictions. We conducted extensive experiments on SECOND and its parameters, described in Section 4.6.1.

Lastly, we have significantly improved the data augmentation techniques by incorporating insights gained in [115]. Previously, only small rotational (-5 to 5 degrees) and translational (-1 to 1 meter shifts) data augmentation was applied to the ego vehicle. Now, we employ four augmentation techniques, resulting in a more robust model:

(1) *Random Flip*: With a probability of 50% the x and y coordinates get multiplied by -1.

(2) *Entire World Rotation*: Each point cloud and its corresponding ground truth bounding boxes are randomly rotated around the z-axis by sampling an angle α from a uniform distribution: $\alpha \sim \mathcal{U}([-\pi, \pi])$.

(3) *Entire World Scaling*: Each point cloud and its corresponding ground truth bounding boxes are randomly scaled by multiplying all coordinates by a factor β randomly sampled from another uniform distribution: $\beta \sim \mathcal{U}([0.95, 1.05])$.

(4) *Foreground-Background Balancing*: At the beginning of the training process, a lookup table of ground truth bounding boxes and their corresponding LiDAR points is created. To address the imbalance of background and foreground points in any given scene, objects and their corresponding points are added to other scenes, ensuring that they do not overlap with any other point cloud data.

The presence of occlusion in the LiDAR data generation process can result in certain objects within the point cloud having limited or no points. Consequently, accurately predicting these objects becomes impossible. To tackle this issue, we have implemented a strategy during training where we remove any bounding box that contains fewer than five LiDAR points. This approach ensures faster convergence and enhances overall performance by eliminating unreliable or insufficient data during the training process. Additionally, some smaller changes were made to the dataset, which are further described in Section 4.3.1.

Given that the LiDAR-generated point cloud lacks relevant color information regarding the traffic light status, we augment our approach by incorporating an off-the-shelf binary image classifier. This classifier is trained to accurately detect images that depict red or yellow lights, thereby supplying the necessary traffic light flag input for the waypoint decoder in PlanT. More detailed information about the implementation of the SECOND PlanT model and the light detector can be found in Section 4.4.

3.4 E2E PlanT

(Privileged) PlanT [82] relies on handcrafted tokens to function, which might be suboptimal since they are not learnable and often fail to capture the complexity and variability of the data. Motivated further by the fact that SECOND PlanT requires route token inputs while end-to-end methods can function without them, we propose E2E PlanT as an extension to (Privileged) PlanT [82]. By incorporating image feature tokens extracted and learned from the RGB image through a CNN backbone, E2E PlanT aims to enhance the end-to-end trainability of the self-driving model while retaining the explainability advantage of PlanT. Further, since the route information is contained in the image tokens, these could be removed from the E2E PlanT input enabling map-free driving.

The architecture changes consist of the following and are illustrated in Figure 3.7:

Tokenization. In addition to object and route tokens we introduce a set of image tokens $\mathcal{I}_t \in R^{I_t \times 512}$, which are extracted prior to the global pooling layer of the backbone via a CNN backbone. Depending on the size of the applied average pooling, the backbone yields I_t image tokens.

Token Embeddings. Similar to the other tokens, we pass the image tokens through a linear layer $\rho_2 : \mathbb{R}^{512} \to \mathbb{R}^H$ and append learnable embedding vectors $\boldsymbol{e}_i \in \mathbb{R}^H$.

Waypoint Prediction. For the waypoint prediction task, we expand the transformer input size by the processed image tokens $\tau : \mathbb{R}^{V_t+N_s+I_s+1\times H} \to \mathbb{R}^{V_t+N_s+I_s+1\times H}$. The waypoints are still predicted by the auto-regressive waypoint decoder γ.

Vehicle Future Prediction. For the future vehicle prediction we do not utilize the processed image token, so this task works the same as before.

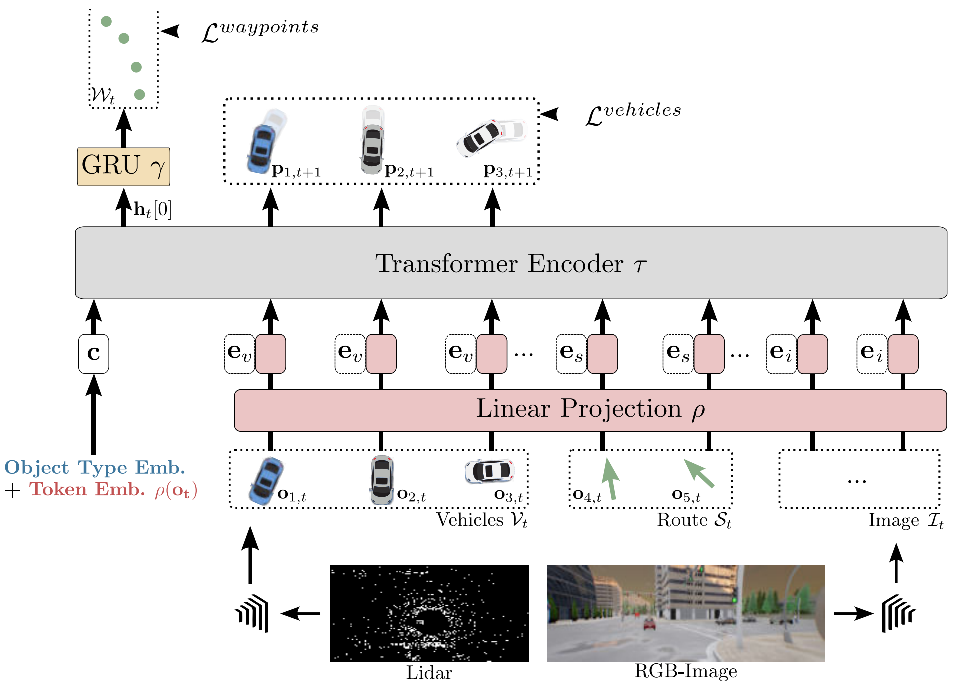


Figure 3.7: **E2E PlanT.** The new architecture includes a CNN backbone that generates image tokens from the RGB-Image. The object tokens of the vehicle are either generated through privileged access to the simulator or with an object detector. Adapted from [82].

Loss Function. Since the overall task has not changed, we also retain the same loss function as before.

4 Experiments

In this Chapter, we present the experiments conducted to evaluate, compare, and discuss our proposed methods SECOND PlanT and E2E PlanT. We commence by describing the task at hand in Section 4.1. Subsequently, we provide an overview of the data, evaluation scheme, and metrics employed for the online experiments in Section 4.2, as well as for the offline experiments in Section 4.3. Following that, we delve into the implementation details in Section 4.4. Finally, we introduce the baselines in Section 4.5 and feature the most important results obtained from the experiments in Section 4.6.

4.1 Task

We consider the task of driving in an urban environment using the CARLA simulator [30], specifically focusing on version 0.9.10.1[1] and the evaluation protocol of the CARLA Leaderboard 1.0. The agents are required to adhere to high-level GPS target points $\mathcal{T}$ provided by an A* navigational planner, aiming to reach the destination within a specified time frame while minimizing infraction penalties. These target points are sparse and can be hundreds of meters apart, unlike the local waypoints predicted by the agent's policy π. We allow for routes that cover both highway and residential areas. The penalized infractions include:

- Collisions with pedestrians, vehicles, or static objects,
- Running a red light or a stop sign,
- Driving on the wrong lane or on the sidewalk,
- Leaving the route specified by the navigational planner.

During each route, the driving agent may encounter several predefined dangerous scenarios aimed at increasing the level of driving difficulty and simulating hazardous situations. This, in turn, requires the agent to navigate skillfully and effectively overcome these challenges in order to ensure safe arrival at his destinations. These scenarios are based on the NHTSA precrash typology [71] and include the following situations:

[1]`https://github.com/carla-simulator/carla/releases/tag/0.9.10.1`

- Scenario 01: Control loss without previous action. The ego-vehicle loses control due to poor road conditions and it must recover by returning to its original lane.
- Scenario 02: Longitudinal control after leading vehicle's brake. The leading vehicle decelerates suddenly due to an obstacle, and the ego-vehicle must perform an emergency brake or an avoidance maneuver.
- Scenario 03: Obstacle avoidance without prior action. The ego-vehicle encounters an obstacle or an unexpected entity on the road and must perform an emergency brake or an avoidance maneuver.
- Scenario 04: Obstacle avoidance with prior action. While performing a maneuver, the ego-vehicle encounters an obstacle on the road and must perform an emergency brake or an avoidance maneuver.
- Scenario 05: Lane changing to evade slow leading vehicle. The ego-vehicle performs a lane change to evade a leading vehicle, which is moving too slowly.
- Scenario 06: Vehicle passing dealing with oncoming traffic. The ego-vehicle must navigate around a blocking object using the opposite lane while yielding to oncoming traffic.
- Scenario 07: Crossing traffic running a red light at an intersection. The ego-vehicle is driving straight at an intersection, but a crossing vehicle runs a red light, forcing the ego-vehicle to avoid a collision.
- Scenario 08: Unprotected left turn at the intersection with oncoming traffic. The ego-vehicle is performing an unprotected left turn at an intersection while yielding to oncoming traffic.
- Scenario 09: Right turn at an intersection with crossing traffic. The ego-vehicle is performing a right turn at an intersection while yielding to crossing traffic.
- Scenario 10: Crossing negotiation at an unsignalized intersection. 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.

To ensure the tasks are as realistic as possible and in order to enhance the variety of sensor data, the routes are distributed across eight different towns. These towns vary in size, graphical resolution, and feature intersections that represent both European and US styles. Additionally, there are 15 distinct weather settings that vary according to the times of the day and lighting conditions. During our research, the CARLA Leaderboard [10] was updated from version 1.0 to 2.0. The new leaderboard incorporates various updates, including new scenarios and new infractions. It is important to note that in this thesis, we solely refer to the settings of the leaderboard version 1.0.

4.2 Online Experiments

In this Section, we focus on our online experiments. Firstly, we introduce the base dataset utilized for training PlanT and E2E PlanT in Section 4.2.1. Subsequently, in Section 4.2.2, we delve into the evaluation and benchmarking of our driving agents using the LAV benchmark. Finally, in Section 4.2.3, we introduce and define all metrics employed.

4.2.1 Dataset & Training

For training **PlanT** and **E2E PlanT**, the *base dataset* is generated following the methodology described in [82]. The dataset is created by using a privileged expert that has access to the ground truth locations, extent, and orientation of all other vehicles, as well as information about their actions and dynamics. Leveraging this information, the rule-based expert policy determines the future positions of other vehicles to prevent most collisions. The dataset consists of approximately 228,000 data points. As in [82], the dataset generation process is repeated three times with different traffic initialization, resulting in a total of approximately 684,000 data points.

During each data generation process, data is collected for around 3,500 routes. Sensor data and ground truth information are saved at a rate of 2 frames per second. For LiDAR data, a sensor is positioned 1.3 meters in front and 2.5 meters above the ego vehicle's origin. The LiDAR scanner rotates at a frequency of 20Hz, producing 1,200,000 LiDAR points per second. For RGB images, a camera is mounted at the rear of the vehicle, directed towards the front, and placed 1.5 meters behind and 2 meters above the ego vehicle's origin. The camera has a resolution of 900x256 and a field of view of 100°. Ground truth bounding boxes for other vehicles are included if the corresponding vehicle is within 50 meters of the ego vehicle. Additionally, a light flag is saved and set to 1 if there is a relevant red light within a distance of 15 meters.

To increase data variety, the routes are distributed across different towns (Towns 01, 02, 03, 04, 05, 06, 07, and 10HD), and a subset of the traffic scenarios described in Section 4.1 (Scenarios 01, 03, 04, 07, 08, 09, and 10) is utilized. Furthermore, weather conditions are shuffled between Clear, Cloudy, Wet, MidRain, WetCloudy, HardRain, and SoftRain, at each saving frame. We also vary the sun's altitude angle to simulate different times of day, and the sun's azimuth angle is randomly changed to further enhance variety.

We adopt the LAV training and evaluation scheme introduced in [15]. This scheme involves deliberately excluding all data points generated in Town02 and Town05 during the training phase. Subsequently, all online evaluations are exclusively conducted within these two hold-out towns, which will be described in Section 4.2.2.

4.2.2 Evaluation

Relying solely on offline evaluation is not always sufficient to gauge the online driving capabilities of driving agents [23]. This is due to the fact that a single significant mistake during driving can have varying consequences depending on the specific situations encountered. Therefore, online evaluation becomes crucial for assessing self-driving models.

The CARLA Leaderboard [10] serves as a widely recognized benchmark in the field of self-driving. The leaderboard evaluation consists of 10 distinct routes with undisclosed weather conditions. Each route is repeated five times with 2 weather conditions each, resulting in a total of 100 routes. These routes are spread across two hidden and inaccessible towns. However, due to a limited monthly evaluation budget of 200 hours for participating teams, conducting extensive experiments and ablations directly on the leaderboard is not feasible. As a result, we choose to emulate the leaderboard evaluation process locally.

To achieve this, we adopt the approach presented in LAV [15] by conducting evaluations exclusively in Town02 and Town05, which were excluded during the training phase. Thus, the agent has never encountered these towns before, making the LAV dataset a suitable proxy for the leaderboard evaluation. Additionally, to enhance and expand the dataset introduced in [15], we incorporate routes from Town07 and Town10HD during training. For evaluation purposes, we follow the LAV protocol [15], evaluating the agent on four representative routes, each with four different weather conditions: ClearNoon, CloudySunset, SoftRainDawn, and HardRainNight. This results in a total of 16 evaluation routes.

For the comprehensive evaluation of the complete **SECOND PlanT** model, comprising the trained object detector and the trained PlanT model, as well as **E2E PlanT**, we employ online evaluation within the CARLA Simulator.

It is important to note that the evaluation process within CARLA is non-deterministic. Factors such as sensor noise, the traffic manager, and physics introduce variance during online evaluation [82]. Therefore, we report the online performance as a mean value with standard deviation, calculated over three complete evaluation runs, encompassing a total of 48 routes.

4.2.3 Metrics

The online metrics are designed to assess the agents' performance during real-time driving. We employ the main three official metrics used in the CARLA Leaderboard: *route completion (RC)*, *infraction score (IS)*, and *driving score (DS)*. These metrics provide comprehensive insights into the agents' ability to complete routes, adhere to traffic rules, and avoid collisions. They also allow us to assess the performance of our agents with other baselines.

Route Completion (RC). route completion measures the percentage of route distance R_i covered by the agent in route i, averaged across all N routes. It is calculated as follows:

$$\mathrm{RC} = \frac{1}{N}\sum_{i}^{N} \mathrm{R}_{i=1} \in [0, 100]\,.$$

A value of RC < 100 may occur if the agent deviates from the designated lane or drives on a sidewalk, if the agent veers off the specified route, or if the agent becomes blocked and fails to complete the route before a timeout occurs.

Infraction Score (IS). The infraction score is initially set to a perfect score of 1.0 for the agent. For each infraction, this score is multiplied by the corresponding penalty coefficient $p_j \in [0, 1]$, resulting in a decrease of score. The following infractions, along with their pre-defined penalty coefficients, are aggregated:

- 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.

Note that subsequent infractions thus have a diminishing impact due to the multiplicative nature of the score.

$$\mathrm{IS} = \prod_{j \in \{\mathrm{Ped},\mathrm{Veh},\mathrm{Stat},\mathrm{Red},\mathrm{Stop}\}} (p_j)^{\#\text{ infractions of type } j} \in [0, 1]\,.$$

Driving Score (DS). The driving score serves as the primary metric for performance and combines the route completion and infraction score. It is calculated by averaging the route completion values weighted by the corresponding infraction scores across all N routes. The driving score is also used for ranking on the CARLA Leaderboard. With RC_i and IS_i denoting the route completion and infraction score of route i, respectively, the DS is computed as follows:

$$\mathrm{DS} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{R}_i \mathrm{IS}_i \in [0, 100]\,.$$

Observe that this calculation differs from simply multiplying the average route completion by the average infraction score.

4.3 Offline Experiments

In this Section, we concentrate on our offline experiments. We begin by describing the modifications we made to the dataset train to SECOND in Section 4.3.1. Then,

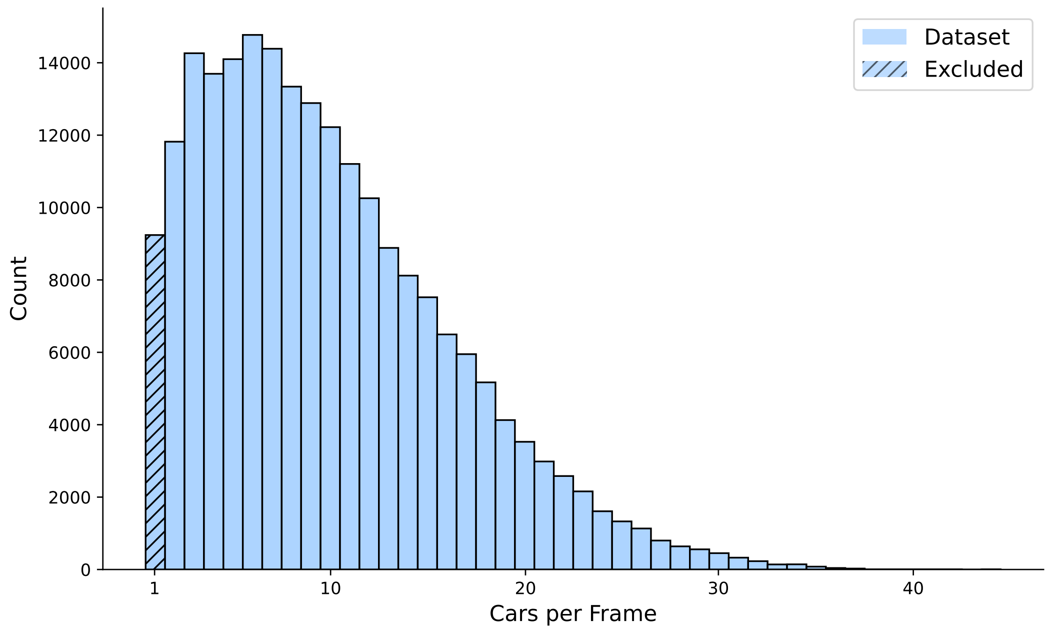


Figure 4.1: **Distribution of Cars per Frame**. To train the detector, we excluded all data points that only contained the ego vehicle.

in Section 4.3.2, we introduce the scheme of the offline evaluations before defining all used offline metrics in Section 4.3.3.

4.3.1 Dataset & Training

To train *SECOND* and other LiDAR-based object detection models, we utilize the OpenPCDet Framework [97]. OpenPCDet was chosen due to its convenient access to a wide range of state-of-the-art models. All models can be pre-trained and used with multiple common object detection benchmarks such as KITTI [34], WAYMO [93] and nuScenes [8]. Additionally, OpenPCDet already incorporates diverse data augmentation techniques.

In order for the models to be compatible with different datasets, a unified coordinate system for the point cloud and ground truth object labels is required. Hence, a transformation of the base dataset is necessary. This involved formatting the point cloud and bounding boxes of the base dataset, and converting them into unified coordinates. This transformation was accomplished through the translation and rotation of the point cloud and bounding boxes. Furthermore, since the detectors only require LiDAR input, extraneous information such as RGB images and traffic light information was discarded.

Upon analyzing the resulting dataset and examining the vehicle distribution per

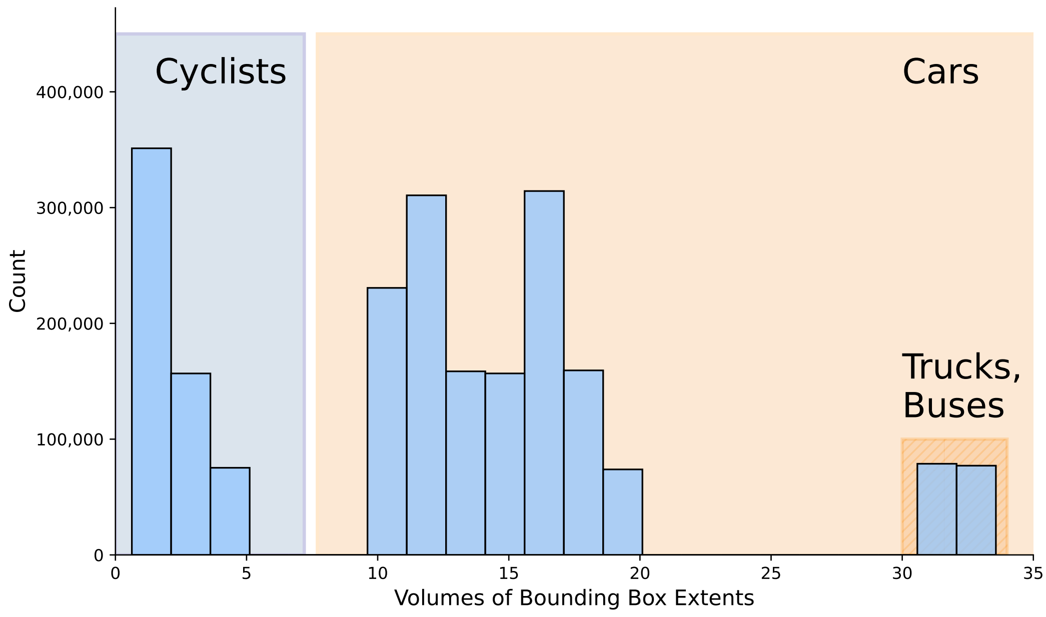


Figure 4.2: **Distribution of car volumes in the base dataset.** Utilizing thresholds for the vehicle's volumes could enable finer classification.

frame (refer to Figure 4.1), we observe that certain scenes contained only the ego vehicle within the range of the LiDAR scan. Consequently, a total of 9240 samples are removed from the training dataset. It is important to note that the base dataset does not differentiate between different vehicle types. All vehicles, including cars, trucks, bicycles, motorcycles, and others, are classified as "Car." Further analysis (see Figure 4.2) led us to discover that applying a threshold of approximately 7.5 for the bounding box volume could be a suitable approach to differentiate between the classes "Car" and "Cyclist." Further, a threshold of 25 could be applied to split the class "Car" into "Car" and "Commercial Vehicle" such as trucks and buses. This differentiation could be used in future work to enable more precise labeling for the planner and facilitate a more specific classification.

To replicate real-world driving scenarios and account for unseen towns, we have deliberately excluded all samples associated with Town02 and Town05 from the training set. Consequently, our evaluation is solely based on these two hold-out towns.

Additionally, we conduct training of a *light detector* described in Section 4.4 in more detail. We train the light detector with the RGB images of the base dataset combined with the binary label of the light detector.

4.3.2 Evaluation

Offline evaluation of object detectors is a practical approach when faced with the challenge of evaluating a large number of trained models efficiently. The evaluation of the **SECOND** perception module is thus performed using offline evaluation.

Since the KITTI dataset [35, 34] is widely used as a benchmark for comparing different 3D object detectors. We sample an equal amount (3,769) of evaluation samples from our hold-out dataset of Town02 and Town05 and use the samples to evaluate the trained detectors. This aligns with the methodology employed in [35], allowing us to leverage evaluation functionality.

4.3.3 Metrics

In addition to the online metrics described in Section 4.2.3, we use a set of offline metrics to evaluate the performance of object detectors used in SECOND PlanT. These metrics offer additional insights into the object detector's performance, allowing for a comprehensive assessment and informed decisions for further improvement. By employing offline metrics, we can evaluate the detector's performance without the need for time-intensive online evaluations. In the following, we will introduce some preliminary metrics before discussing the two main offline metrics: $\mathrm{AP}_{\mathrm{3D}}$ and $\mathrm{AP}_{\mathrm{BEV}}$.

IoU. Intersection over Union (IoU) is a metric used to measure the overlap between the predicted bounding box B_p and the ground truth bounding box B_{gt} of an object. It is computed by dividing the area of intersection between the two bounding boxes by the area of their union. IoU is typically used as a threshold to determine whether a predicted bounding box is considered a true positive or false positive. Formally, IoU is defined as:

$$\mathrm{IoU}(B_p, B_{gt}) = \frac{\mathrm{Area}(B_p \cap B_{gt})}{\mathrm{Area}(B_p \cup B_{gt})} \in [0, 1].$$

Recall. Recall, also known as *sensitivity* or *true positive rate (TPR)*, measures the ability of an object detection model to detect all relevant objects in a scene. It is calculated by dividing the number of true positive (TP) detections by the sum of true positive and false negatives (FN). A high Recall value indicates that the model is good at detecting objects, but it may also generate more false positives. For a predefined IoU threshold ξ that decides whether a prediction is a TP or an FN, recall is calculated as:

$$\mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \in [0, 1],$$

where TP corresponds to the number of objects that get correctly detected ($\mathrm{IoU} \geq \xi$) and FN to the number of objects that get not detected ($\mathrm{IoU} < \xi$).

Precision. Precision measures the accuracy of the object detections made by the model. It is calculated by dividing the number of true positive detections by the sum of true positives and false positives (FP). A high Precision value suggests that the model has a low false positive rate, but it may also lead to missing some true positive detections. For a predefined IoU threshold ξ that decides whether a prediction is a TP or a FN, Precision is calculated as:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \in [0,1],$$

where TP corresponds to the number of objects that get correctly detected (IoU $\geq \xi$) and FP to the number of wrong detections and redundant predictions.

AP. Average Precision (AP), introduced in [32], is a widely used evaluation metric for object detection. It summarizes the precision-recall curve of the model by calculating the average precision at different recall levels. The precision-recall curve is obtained by varying the detection thresholds of the model and computing precision and recall values at each threshold. AP is then calculated by integrating the area under the precision-recall curve. AP provides an overall measure of the model's performance, taking into account both precision p and recall r:

$$\text{AP} = \int_0^1 \max_{r':r'\geq r} p(r')dr \in [0,1].$$

To calculate the AP in our experiments, we follow KITTI [35] by iteratively assigning the ground truth labels to the predictions starting with the largest overlap, measured by IoU. We require TP to overlap by more than **70%** and count multiple detections of the same object as false positives. With $p(r)$ being the precision for a given recall level r, we estimate the AP with eleven equally spaced recall levels $\{0, 0.1, \ldots, 1\}$:

$$\text{AP} = \frac{1}{11} \sum_{r \in \{0,0.1,\ldots,1\}} \max_{r':r'\geq r} p(r') \in [0,1].$$

AP$_{\text{3D}}$. KITTI's 3D Average Precision AP$_{\text{3D}}$ [35] is calculated by considering the IoU of the 3D bounding box, following the same procedure described earlier for AP. This allows us to evaluate the model's performance specifically in the context of 3D object detection.

AP$_{\text{BEV}}$. Similarly, KITTI's BEV Average Precision AP$_{\text{BEV}}$ [35] is calculated by taking into account the IoU of the 2D bounding box, using the same approach as before. By projecting the ground truth and predicted bounding boxes onto 2D space, we assess the model's performance in accurately detecting objects from a bird's-eye view perspective. This evaluation is particularly relevant for assessing the performance of object detectors in the context of PlanT, as it relies on the input of 2D bounding boxes.

4.4 Implementation

In this Section, we provide detailed information about the model configurations used for PlanT, SECOND, SECOND PlanT, and E2E PlanT during both training and inference.

PlanT. To train and evaluate our LAV PlanT model, we follow the settings outlined in the original paper [82] and utilize their open-source codebase[2]. The transformer architecture is based on BERT [27, 102] and consists of 8 layers with a hidden size of $H = 512$ and 8 heads, resulting in a total of 41.4 million parameters. We train the model from scratch. To filter out unimportant vehicles we set $D_{\max} = 30m$, and the number of route tokens is set to $N_s = 2$. For the discretization of predicted future tokens of other vehicles, we use $Z_1 = 4$ for speed, $Z_2 = Z_3 = 128$ for position, and $Z_3 = 32$ for orientation. In the combined loss function, we assign a weight of $\lambda = 0.2$ to the auxiliary task. We train PlanT using four RTX 2080Ti GPUs for 47 epochs, excluding all data points from Town02 and Town05, as described in Section 4.2.1.

We employ AdamW optimization with a learning rate of 0.0001, a batch size of 128, and a weight decay of 0.1 for the last epoch. Gradient normalization of 1.0 is also applied during training. During inference, we use two PID Controllers to obtain the steering and gas/brake commands from the predicted waypoints. The hyperparameters for these controllers are taken from the codebase of Transfuser [21]. The parameters for the turn controller are $K_P = 0.9, K_I = 0.75, K_D = 0.3$, whereas, for the speed controller, they are $K_P = 5.0, K_I = 0.5, K_D = 1.0$.

Perception PlanT. As our lower bound for SECOND PlanT, we retrain the original Perception PlanT [82]. Perception PlanT makes use of 180° of the LiDAR cloud ranging up to 32m to the front and 16m to each side. During training, Perception PlanT utilizes data augmentation by shifting the scene 1 meter to the left or to the right and rotating the camera position by ±5°. For the detections, a threshold $\phi = 0.1$ is applied to exclude uncertain predictions.

SECOND. We train all variants of SECOND by leveraging a modified version of the OpenPCDet codebase [97]. The object detectors are trained for 80 epochs on eight RTX 2080Ti GPUs, with a batch size of 64. For optimization, we employ Adam with weight decays of 0.001 and 0.01. As for the learning rate scheduler, we use a OneCycle scheduler with a momentum of 0.9. The cycle steps are set to 35 and 45, respectively, and the learning rate decay is set to 0.1. We initiate the cycle with 10% of the learning rate. To determine the hyperparameters for the learning rate, weight decay, and maximum gradient norm, we conduct a hyperparameter grid search.

SECOND PlanT. SECOND PlanT combines PlanT with an object detector. To that end, training is performed separately, as described previously. During inference, three additional steps are taken. First, any detections with a score lower than the threshold $\phi = 0.3$ are discarded. Ablations on this threshold can be found

[2]`https://github.com/autonomousvision/plant`

in Section 4.6. Secondly, the remaining detections are passed through NMS (see Section 3.2.3) with an IoU overlap threshold 0.2. Lastly, we infer the speed of the predicted vehicles through matching. To enable matching, we keep track of the predictions from the last four timesteps and employ the Hungarian method [51]. Once matching is completed, we predict the speed of each object by dividing the Euclidean distance of the matched vehicle's position by the time between the two timesteps.

Light Detector. We implement a light detector for classifying traffic light states. We utilize a pre-trained ResNet34 model to extract 512 features from the camera input. A one-layer feed-forward neural network with a hidden layer size of 128 is employed for binary image classification. The entire model is trained for 6 epochs on a single RTX 2080Ti GPU in the LAV setting with a learning rate of 0.0001 with binary cross-entropy loss. During inference, we utilize a light classification threshold of 0.5.

E2E PlanT. Since E2E PlanT builds upon PlanT, we utilize the forked codebase of PlanT for both training and evaluation. Regarding the training, most of the hyper-parameters remain the same. We additionally employ a ResNet34 [40], pre-trained on ImageNet, as the backbone CNN and extract features from the image captured by the car's camera. Feature extraction is done before the global pooling layer of the ResNet. Before passing the RGB image to the CNN, we apply standard ImageNet normalization to maximize the effect of pre-training. Normalization is performed for each pixel using z-normalization: $z = \frac{x-\mu}{\sigma}$. We utilize $\mu = (0.485, 0.456, 0.406)$ and $\sigma = (0.229, 0.224, 0.225)$, as used in ImageNet, for each channel respectively. If no pooling is applied, the backbone yields $I_t = 232$ image tokens. In our experiments, we demonstrate the impact of varying the number of image tokens between 56, 14, and 3, achieved by using $2 \times 2, 4 \times 4$, and 8×8 average pooling layers within the CNN backbone, respectively. Depending on the feature mode, a batch size of 128 or 64 is used for training on four RTX 2080Ti GPUs.

4.5 Baselines

In this Section, we establish performance baselines against which we will evaluate our proposed models. Baselines serve as a critical reference point, enabling us to assess the effectiveness and potential contributions of our novel approaches. We include three of the top four models from the CARLA Leaderboard: LAV [15], TCP [113] and TransFuser [21]. Additionally, we describe two more models, Perception PlanT [82] and Privileged Plant [82], serving as the lower and upper bounds for the performance of our models respectively. Each of these models is briefly discussed in the following.

4.5.1 TCP

TCP (Trajectory-guided Control Prediction) [113] is an end-to-end model that exclusively uses monocular camera input. It currently holds the second position on the CARLA Leaderboard. The model has two outputs: one for waypoints and one for direct control commands. During inference, these outputs are combined using an adaptive ensemble approach that applies a weighted average based on the lateral control state of the vehicle.

For our experiments, we use the publicly released code[3] to retrain the TCP model in the LAV setting. We note that this is not the exact same model as the one used for TCP's leaderboard entry.

4.5.2 LAV

LAV (Learning from All Vehicles) [15] currently holds the third position on the CARLA Leaderboard. It is an end-to-end learning-based approach that trains a motion planner using not only ego trajectories but also trajectories of all nearby vehicles. The model incorporates a PointPillars [52] and a PointPainting [107] backbone for each LiDAR point and merges semantic class information, which is extracted from the RGB, with the LiDAR point cloud resulting in an intermediate bird's-eye-view representation. For BEV object detection, a one-stage Center-Point [118] is employed. Additionally, a binary brake classifier makes use of information from all four camera images to handle red light situations.

For our experiments, we use the publicly released code[4] to retrain the LAV v2 model in the LAV setting. We remark that this is not the exact same model as the one used for LAV's leaderboard entry.

4.5.3 TransFuser

TransFuser [21] is a widely used baseline for CARLA models. It is trained end-to-end using both camera input and LiDAR input. The LiDAR point cloud is projected into BEV space, and both sensor information is processed and fused using transformers. The model outputs waypoints and uses PID controllers during inference. Currently, TransFuser is placed rank four on the CARLA Leaderboard.

For our experiments, we use the released code[5] to retrain the model in the LAV setting. We note that this is not the exact same model as the one used for TransFuser's leaderboard entry.

[3]`https://github.com/OpenDriveLab/TCP`
[4]`https://github.com/dotchen/LAV`
[5]`https://github.com/autonomousvision/transfuser`

4.5.4 Perception PlanT

As already introduced in Section 3.2.5, Perception PlanT [82] serves as our lower bound baseline. It uses an off-the-shelf perception module leveraged from TransFuser, which takes an RGB and BEV LiDAR point cloud as input. The module predicts 2D bounding boxes, which are then used as input for PlanT. Further implementation details for Perception PlanT can be found in Section 4.4.

For our experiments, we use the released code[6] to retrain the model in the LAV setting.

4.5.5 Privileged PlanT

Privileged PlanT serves as our upper bound since it directly utilizes ground truth detections as input for PlanT, obtained from the simulator. It is discussed in detail in Section 3.2.5.

For our experiments, we also use the released code[6] to retrain the model in the LAV setting.

4.6 Results

This section presents the most significant findings obtained from our experiments. Firstly, in Section 4.6.1, we focus on evaluating the performance of SECOND PlanT and provide a comprehensive comparison to the baselines using the LAV Benchmark. Secondly, in Section 4.6.2, we analyze the improvements in inference time. We further provide ablation studies of the improvement components in Section 4.6.3. Additionally, in Section 4.6.4, we examine the impact of the detection threshold, while Section 4.6.5 delves into the performance of the light detector and explores current failure cases. Furthermore, Section 4.6.6 showcases the performance of both SECOND PlanT and the baselines when removing stop sign infractions. Lastly, Section 4.6.7 presents our results for E2E PlanT, including the final evaluation scores, and ablations on the number of image tokens, the removal of route input, and the CNN backbone.

4.6.1 SECOND PlanT

We compare our SECOND PlanT method against the baselines described in Section 4.5. As mentioned in Section 4.2.2, all methods including ours are trained without access to Town02 and Town05 to test generalization capabilities. The results of the different methods on the LAV benchmark are presented in Table 4.1. To mitigate the impact of randomness during model optimization and evaluation, all

[6] `https://github.com/autonomousvision/plant`

Method	**Input**	**R**	**T**	**DS ↑**	**RC ↑**	**IS ↑**
LAV v2 [15]	C+L	✗	✗	27.15±2.24	**98.56±1.23**	0.27±0.02
Perception PlanT [82]	C+L	✗	✗	37.59±3.88	86.05±6.10	0.46±0.08
TransFuser [21]	C+L	✗	✗	38.91±8.14	84.08±6.37	0.46±0.06
TCP [113]	C	✗	✗	57.90 ±7.36	84.63±4.62	0.67±0.07
Perception PlanT [82]	C+L	✓	✗	61.93±5,29	93.54±4.67	0.68±0.04
SECOND PlanT (ours)	C+L	✓	✗	**69.26±4.32**	91.50±4.21	**0.77±0.05**
Privileged PlanT [82]	*Obj.*	✓	✓	*68.91*±5.37	*93.64*±5.50	*0.74*±0.02

Table 4.1: **LAV Results.** We report the mean± std for 3 trained models on 3 different evaluation seeds each (144 routes in total). SECOND PlanT reaches the performance of the privileged planner and surpasses all previous methods in terms of driving score and infraction score. Abbreviations for model input: C - camera, L - LiDAR, R - route, T - ground truth traffic light state.

models underwent evaluation using three different training seeds and three evaluation runs, resulting in a total of nine complete evaluations and 144 routes in total. Unless otherwise stated, we report the mean and standard deviation across all 144 routes.

The first four entries in the table represent the baselines that rely solely on sensor data and do not use route information (R) or privileged information such as the ground truth traffic light state (T). Despite relying solely on the camera sensor, TCP [113] significantly outperforms the other sensor models with a driving score of 57.90. While LAV v2 achieves the highest route completion score, it does so by causing much more infractions than the other models.
SECOND PlanT achieves a driving score of 69.26 points, surpassing previous approaches on the LAV benchmark. A clear difference of 7.3 points in the driving score is observed when comparing it to the previous state-of-the-art Perception PlanT.

Privileged PlanT serves as our upper bound baseline since it utilizes the same planner but has access to privileged ground truth information instead of detecting objects and the traffic light state. However, due to the high variance of the CARLA simulator evaluations, we even outperform our upper bound by roughly one point in our sample of nine evaluations, demonstrating that the performance of our model is now limited by the planner and not the detection quality.

Figure 4.3 provides a qualitative example of SECOND PlanT in a typical driving scenario. Even partially occluded vehicles with limited point cloud data are accurately detected, thereby providing the planner with near ground truth inputs.

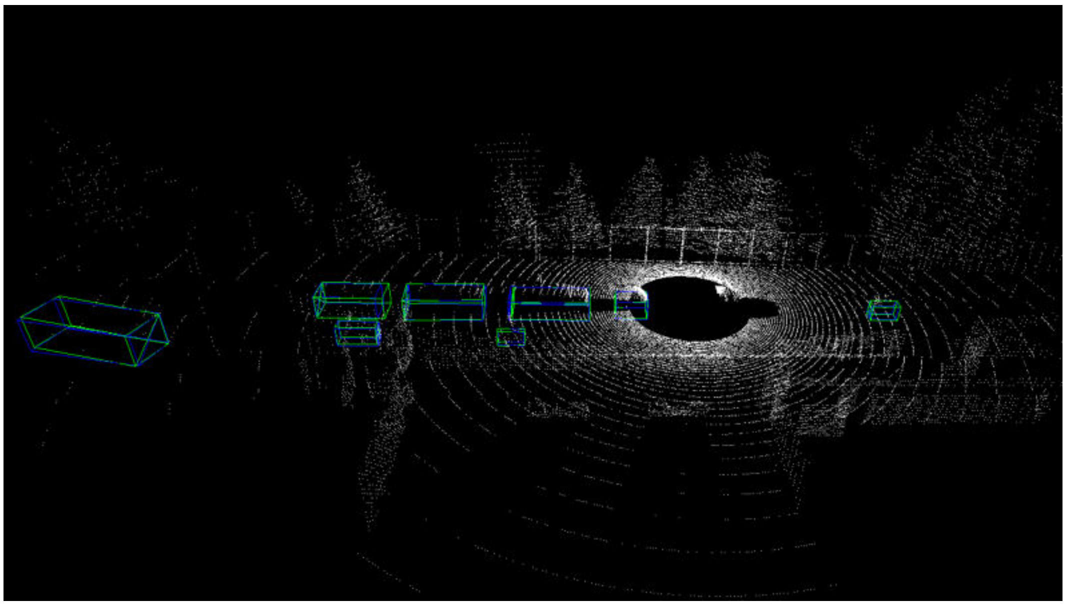


Figure 4.3: **3D LiDAR Object Detections.** Detections of our trained SECOND PlanT. Ground truth bounding boxes are represented in blue and detected bounding boxes are colored in green. SECOND PlanT achieves near-perfect object detection even for vehicles that contain very few points due to occlusion.

Model	IT Objects (ms)	IT Light (ms)
SECOND PlanT (ours)	**18.88**	**5.09**
Perception PlanT [82]	30.3	

Table 4.2: **Inference speed.** Inference speed for object detections and traffic light classification of SECOND Plant compared to the previous Perception PlanT method measured in ms. Times are calculated as an average across all steps of one evaluation route.

4.6.2 Inference Speed

To determine the time improvement achieved by the new lightweight object detector, we measured the inference time of both SECOND and the light detector in milliseconds. The measurements were conducted on a single RTX 3080 GPU, recording the inference time over an entire route and computing the average. Table 4.2 demonstrates that our new model significantly reduces the inference time from the previous 30.3 ms to just 23.97 ms, enabling efficient real-time object detections and light predictions. In principle, the two modules could also be parallelized, further boosting efficiency.

LiDAR range	**Aug.**	**DS ↑**	**RC ↑**	**IS ↑**	**AP$_{3D}$ ↑**	**AP$_{BEV}$ ↑**
180°	old	59.34±6.53	90.64±4.46	0.68 ±0.04	–	–
180°	✗	57.11±6.31	89.93±3.89	0.63±0.06	42.92	43.06
180°	✓	56.74±1.06	**92.58±1.98**	0.63±0.03	41.21	41.59
360°	✗	53.64±4.89	83.10±1.12	0.69±0.04	**74.45**	**75.45**
360°	✓	**67.36±2.73**	90.59±3.99	**0.76±0.03**	70.52	71.68

Table 4.3: **LiDAR Range and Data Augmentation.** We compare the impact of data augmentation and LiDAR range increase on offline and online metrics, compared to the existing Perception PlanT model. To specifically address object detections in the online performance we allow all 5 models access to the traffic light state.

4.6.3 Ablation Study

To evaluate the effectiveness of the various improvements introduced in comparison to Perception PlanT, we conducted model retraining and evaluation under different conditions. To further focus on the influence of object detections we allow the models access to the traffic light state. The results of our ablation study, measuring online performance on the LAV benchmark using three different evaluation seeds, are presented in Table 4.3.

Compared to Perception PlanT, we observed a significant improvement in the driving score of 8 points by simultaneously incorporating the new LiDAR range, the new data augmentation techniques, and employing the new object detector SECOND. Interestingly, the offline metrics AP$_{3D}$ and AP$_{BEV}$ that we used to guide our selection of the object detector model do not strongly correlate with online performance. Despite data augmentation showing a negative impact on pure offline performance, it reduces performance variance and performs particularly well when combined with an increased LiDAR range. While models utilizing only the front LiDAR scan exhibit higher route completion, they sometimes result in collisions with vehicles approaching from the rear. The best overall performance is achieved when combining the new object detector SECOND with a full 360° LiDAR range and data augmentation during training.

4.6.4 Detection Threshold Selection

The detection threshold ϕ plays a crucial role in the modular pipeline driving stack. Setting the threshold too low may result in the planner being overwhelmed by potentially incorrect detections and getting stuck without progressing further. Conversely, setting the threshold too high may cause the planner to miss important vehicles in its surroundings. Figure 4.4 illustrates a scene where numerous false positives are visible with a detection threshold of 0.1 and 0.3.

ϕ	**DS** ↑	**RC** ↑	**IS** ↑
0.1	58.47±3.89	86.71±5.95	0.69±0.05
0.2	64.17±2..34	**97.37±2.57**	0.66±0.12
0.3	**67.36±2.73**	90.59±3.99	**0.76±0.03**
0.4	62.76±4.95	90.24±3.84	0.71±0.05

Table 4.4: **Detection Thresholds.** DS with ground truth light on LAV (3 evaluation runs) with different detection threshold ϕ to filter out uncertain predictions of the Perception Module.

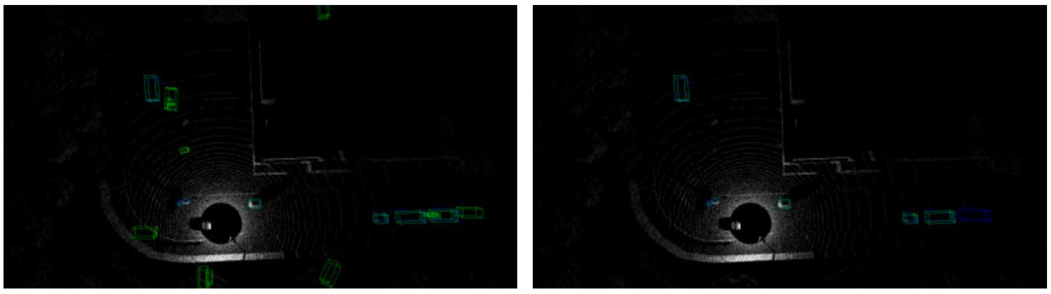

(a) Predicted objects for ϕ = 0.1 (b) Predicted objects for ϕ = 0.3

Figure 4.4: **Detection Threshold.** Setting the detection threshold can have a large impact on the planner's decisions.

To determine an appropriate detection threshold, we conducted an ablation study using three online evaluation runs on the LAV benchmark. To get rid of random infraction noise, we provide the agent with ground truth traffic light state during these experiments. The results suggest that a detection threshold of 0.3 yields optimal performance, and thus we adopt this threshold for SECOND PlanT.

4.6.5 Light Detector

As LiDAR data does not provide any information about the state of traffic lights, an additional light detector is necessary for driving. Table 4.5 presents the amount of traffic light infractions aggregated over the total of 144 routes. It is evident that traffic light infractions are still higher when using the detected traffic light state, although our detector performs well overall. Evaluating the planner with ground truth traffic light information still results in 7 infractions. Upon further analysis, we identify two main failure cases of the light detector.

Figure 4.5 illustrates the most prominent failure case, where the light detector mistakes the green light of a pedestrian or the crossing lane as its own traffic light. Upon closer examination of this failure case, we noticed that it only occurs when the camera is directly facing the pedestrian traffic light, which is caused by a lane change

GT Light	#Traffic Light Infractions ↓
✗	18
✓	**7**

Table 4.5: **Light Detector.** Total number of traffic light infractions across all 144 evaluation routes. Despite the increased number of infractions, the light detector demonstrates good overall performance.

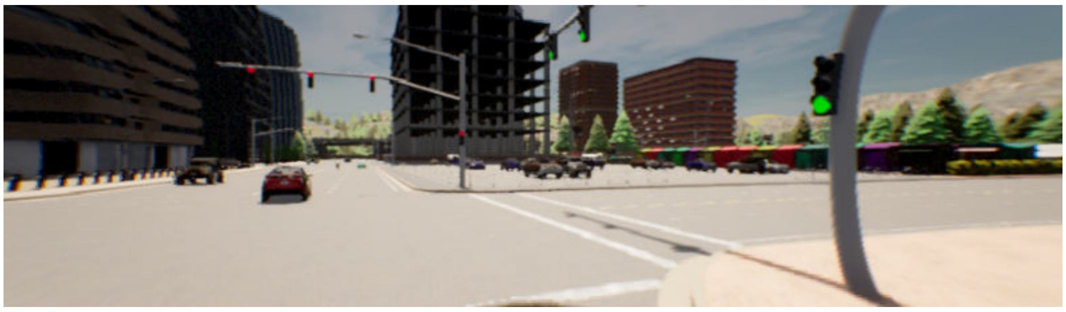


Figure 4.5: **Failure Case 1 of the Light Detector.** The detector mistakenly identifies another visible green light as its own.

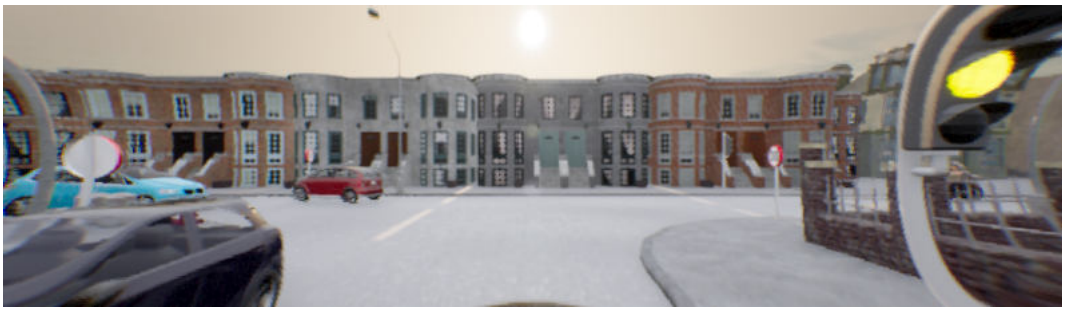


Figure 4.6: **Failure Case 2 of the Light Detector.** The detector occasionally fails to detect the traffic light turning yellow just before crossing.

prior to the crossing. Another rare failure case occurs when the traffic light turns yellow right before the agent drives through it, as shown in Figure 4.6. Additionally, Figure 4.7 and Figure 4.8 display traffic lights in two different styles. Despite the low resolutions, the model confidently predicts a red light with a certainty of over 99.99%, while a human observer may struggle to detect the light. We hypothesize that data augmentation during training could potentially help mitigate these failure cases.

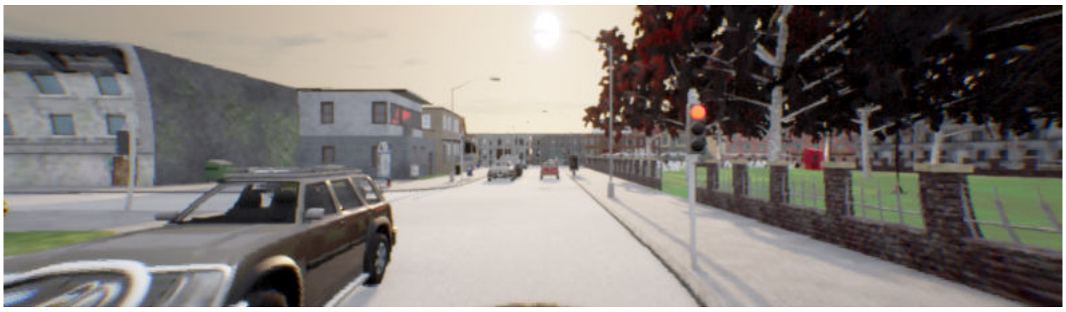


Figure 4.7: **Detected Traffic Light in EU Style.** Scene displaying a detected traffic light in European style with a certainty of above 99.99%.

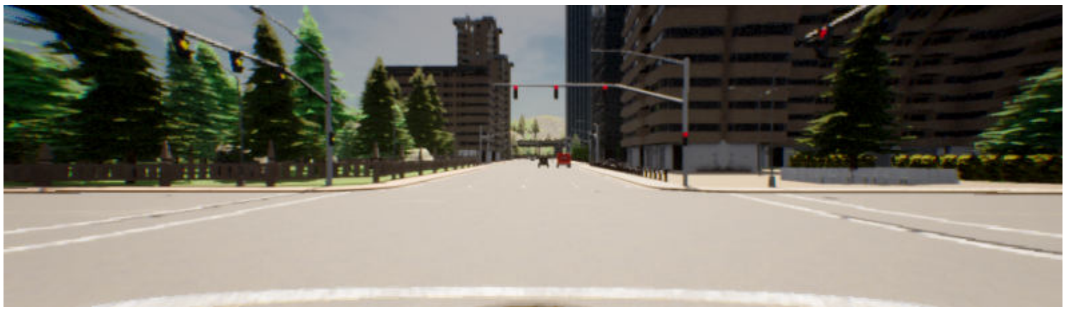


Figure 4.8: **Detected Traffic Light in US Style.** Scene showcasing a detected traffic light in American style with a certainty of above 99.99%.

4.6.6 Removal of Stop Sign Penalty

On the CARLA Leaderboard [10], all reported models achieve a perfect stop sign infraction score of 0.0. Additionally, it has been observed that certain routes in Town02 and Town05 may incorrectly report stop sign infractions even when there are no stop signs visible [21]. This observation motivates Table 4.6, where we compare the baselines from Table 4.1 again, but this time excluding stop sign penalties from the score aggregation for all methods. Despite previously achieving a driving score of 69.26 and an infraction score of 0.77, SECOND PlanT's performance significantly improved to 84.52 and 0.93, respectively. It is evident that SECOND PlanT still outperforms all end-to-end baselines by a substantial margin. Despite generating a total of 18 traffic light infractions (see Table 4.5), SECOND PlanT achieves an infraction score of 0.93. Perception PlanT achieves the same infraction score with an even higher driving score, primarily attributed to the mentioned traffic light infractions and better route completion.

Method	**Input**	**R**	**T**	**DS ↑**	**RC ↑**	**IS ↑**
LAV v2 [15]	C+L	✗	✗	45.73±3.95	**98.56±1.23**	0.46±0.04
TransFuser [21]	C+L	✗	✗	50.61±6.11	84.08±6.37	0.58±0.04
Perception PlanT [82]	C+L	✗	✗	57.56±4.75	86.05±6.10	0.66±0.07
TCP [113]	C	✗	✗	73.62 ±7.21	84.63±4.62	0.83±0.07
Perception PlanT [82]	C+L	✓	✗	**86.46±5.61**	93.54±4.67	**0.93±0.04**
SECOND PlanT (ours)	C+L	✓	✗	84.52±4.47	91.50±4.21	**0.93±0.03**
Privileged PlanT [82]	*Obj.*	✓	✓	*89.62±7.91*	*93.64±5.50*	*0.96±0.03*

Table 4.6: **LAV Results without Stop Sign Penalty.** The evaluation results on all 144 routes, obtained by excluding stop sign infractions for all methods.

4.6.7 E2E PlanT

Table 4.7 presents the results of online evaluations for different variants of E2E PlanT, conducted on one training run and three evaluation seeds. For comparison, we also include online evaluations of SECOND PlanT and Privileged PlanT, selecting the training run with median performance. Our experiments with different image token sizes reveal that E2E PlanT performs optimally with three image tokens. The significant performance variance observed across different numbers of image tokens suggests that the model is not effectively utilizing the image information yet. E2E PlanT with three image tokens achieves a comparable score to Privileged PlanT, which employs the same architecture without the image tokens. Removing the route input has a significant detrimental effect on the model's performance, suggesting that learning the route implicitly from the images is not a straightforward task. Therefore, future work should focus on architectural improvements to reduce reliance on the map input and enhance the overall performance of the model.

Route Input	#image tokens	DS ↑	RC ↑	IS ↑
✓	3	**69.04±3.10**	96.49±0.90	0.72±0.02
✓	14	35.90±4.12	**99.07±1.07**	0.36±0.04
✓	56	40.68±2.62	96.88±2.04	0.43±0.01
✓	232	65.77±5.25	92.84±3.95	0.72±0.02
✗	3	14.17±2.00	63.48±7.77	0.31±0.04
✗	14	7.83±0.10	71.14±1.86	0.17±0.01
✗	56	12.33±0.39	67.42±3.25	0.23±0.00
✗	232	12.22±1.60	71.58±5.13	0.20±0.04
SECOND PlanT		67.55±4.10	89.83±6.32	**0.77±0.04**
Privileged PlanT [82]		*72.31±1.25*	*94.79±3.33*	*0.77±0.02*

Table 4.7: **LAV Results of E2E PlanT Variants.** Performance of E2E PlanT variants on 3 evaluation seeds each compared to the median performance of 3 trained models for SECOND PlanT and Privileged PlanT, also evaluated on 3 seeds each.

5 Conclusion

Summary. In recent years, the research field of autonomous driving has focused predominantly on the advancement of end-to-end models, leaving modular pipeline methods with lackluster development. However, our simple modular pipeline SECOND PlanT, which consists of an object detector, a planner, and a controller, has showcased performance that is comparable to state-of-the-art end-to-end driving stacks. Remarkably, it outperforms all existing published modular pipeline stacks within the CARLA framework on the LAV benchmark by a significant margin, while also improving the inference time compared to previous methods. By exploring and optimizing important steps in the self-driving stack such as the detection threshold for the object detector, enabling full 360° LiDAR coverage, employing various data augmentation techniques, and enhancing the dataset, we have effectively harnessed the progress achieved in planning and object detection research. These collective efforts culminated in the development of a state-of-the-art modular pipeline.

We have further introduced E2E PlanT, a modular pipeline that incorporates additional end-to-end-learnable image tokens. Although the performance of E2E PlanT did not surpass Privileged PlanT, it highlights the potential of hybrid architectures for future investigation.

Future Work. Although LAV approximates the CARLA Leaderboard, the immediate next steps involve conducting experiments on other challenging settings, such as Longest6 and the official CARLA Leaderboard [10], to ensure the overall performance of SECOND PlanT across various benchmarks. Currently, all of our models rely on map input. However, in real-world scenarios, HD Maps are not consistently available and their creation and maintenance are costly. Therefore, further investigation into the removal of map input is desirable. SECOND PlanT's architecture allows for more extensive qualitative analysis. Future work could delve into the interpretability aspect and explore any remaining failure cases of the model. While our object detector achieves performance close to the upper bound when using ground truth objects, the performance of our planner, PlanT, is constrained by the expert driving algorithm used to generate the dataset. The imperfect score of the expert driver sets a performance ceiling for the overall pipeline. Since obtaining human labels within a simulator is infeasible, further research is needed to enhance the quality of the dataset. Furthermore, we have identified failure cases in our light detector. These issues could potentially be addressed by incorporating data augmentation techniques during light detector training or by integrating a light detector head that operates on the extracted image features within the E2E PlanT architecture. Although the performance of E2E PlanT did not surpass that of the base model, this could be attributed to the handcrafted tokenization of vehicles and routes. Conducting additional experiments where learnable features are extracted from the LiDAR point cloud or the detections of the perception module may result in improved performance.

Bibliography

[1] J. Alammar. The illustrated transformer. `http://jalammar.github.io/illustrated-transformer/`. Accessed: 2023-06-09.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[3] E. Arnold, O. Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby, and A. Mouzakitis. A survey on 3d object detection methods for autonomous driving applications. *IEEE Transactions on Intelligent Transportation Systems*, 20(10):3782–3795, 2019.

[4] M. Bansal, A. Krizhevsky, and A. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018.

[5] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis. Soft-nms — improving object detection with one line of code. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5562–5570, Los Alamitos, CA, USA, oct 2017. IEEE Computer Society.

[6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[7] G. Brazil and X. Liu. M3d-rpn: Monocular 3d region proposal network for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9287–9296, 2019.

[8] 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 *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

[9] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 213–229. Springer, 2020.

[10] Carla. Carla autonomous driving leaderboard, 2020. `https://leaderboard.carla.org/leaderboard/`, 2020. Accessed: 2023-06-09.

[11] S. Casas, A. Sadat, and R. Urtasun. Mp3: A unified model to map, perceive, predict and plan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14403–14412, 2021.

[12] F. Chabot, M. Chaouch, J. Rabarisoa, C. Teuliere, and T. Chateau. Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2040–2049, 2017.

[13] R. Chekroun, M. Toromanoff, S. Hornauer, and F. Moutarde. Gri: General reinforced imitation and its application to vision-based autonomous driving. *arXiv preprint arXiv:2111.08575*, 2021.

[14] D. Chen, V. Koltun, and P. Krähenbühl. Learning to drive from a world on rails. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15590–15599, 2021.

[15] D. Chen and P. Krähenbühl. Learning from all vehicles. In *CVPR*, 2022.

[16] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl. Learning by cheating. In *Conference on Robot Learning (CoRL)*, 2019.

[17] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2147–2156, 2016.

[18] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun. 3d object proposals for accurate object class detection. *Advances in neural information processing systems*, 28, 2015.

[19] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1907–1915, 2017.

[20] Y. Chen, S. Liu, X. Shen, and J. Jia. Fast point r-cnn. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9775–9784, 2019.

[21] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger. Transfuser: Imitation with transformer-based sensor fusion for autonomous driving. *Pattern Analysis and Machine Intelligence (PAMI)*, 2022.

[22] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, Oct. 2014. Association for Computational Linguistics.

[23] F. Codevilla, A. M. López, V. Koltun, and A. Dosovitskiy. On offline evaluation of vision-based driving models. In *European Conference on Computer Vision*, pages 246–262. Springer, 2018.

[24] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4693–4700. IEEE, 2018.

[25] F. Codevilla, E. Santana, A. M. López, and A. Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9329–9338, 2019.

[26] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[27] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[28] E. D. Dickmanns, R. Behringer, D. Dickmanns, T. Hildebrandt, M. Maurer, F. Thomanek, and J. Schiehlen. The seeing passenger car'vamors-p'. In *Proceedings of the Intelligent Vehicles' 94 Symposium*, pages 68–73. IEEE, 1994.

[29] M. Ding, Y. Huo, H. Yi, Z. Wang, J. Shi, Z. Lu, and P. Luo. Learning depth-guided convolutions for monocular 3d object detection. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition workshops*, pages 1000–1001, 2020.

[30] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[31] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10:112–122, 1973.

[32] M. Everingham, L. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88:303–338, 2010.

[33] L. Fan, X. Xiong, F. Wang, N. Wang, and Z. Zhang. Rangedet: In defense of range view for lidar-based 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2918–2927, 2021.

[34] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[35] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[36] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[37] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica. Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8806–8813. IEEE, 2021.

[38] N. Hanselmann, K. Renz, K. Chitta, A. Bhattacharyya, and A. Geiger. King: Generating safety-critical driving scenarios for robust imitation via kinematics gradients. In *European Conference on Computer Vision(ECCV)*, 2022.

[39] C. He, H. Zeng, J. Huang, X.-S. Hua, and L. Zhang. Structure aware single-stage 3d object detection from point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11873–11882, 2020.

[40] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[41] S. Hecker, D. Dai, and L. Van Gool. End-to-end learning of driving models with surround-view cameras and route planners. In *Proceedings of the european conference on computer vision (eccv)*, pages 435–453, 2018.

[42] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[43] 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 *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXVIII*, pages 533–549. Springer, 2022.

[44] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang, et al. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17853–17862, 2023.

[45] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[46] B. Jaeger, K. Chitta, and A. Geiger. Hidden biases of end-to-end driving models. *arXiv preprint arXiv:2306.07957*, 2023.

[47] J. Janai, F. Güney, A. Behl, A. Geiger, et al. Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Foundations and Trends® in Computer Graphics and Vision*, 12(1–3):1–308, 2020.

[48] H. Jiang and E. Learned-Miller. Face detection with the faster r-cnn. In *2017 12th IEEE international conference on automatic face & gesture recognition (FG 2017)*, pages 650–657. IEEE, 2017.

[49] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE, 2019.

[50] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander. Joint 3d proposal generation and object detection from view aggregation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE, 2018.

[51] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[52] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12697–12705, 2019.

[53] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, et al. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10):727–774, 2008.

[54] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE intelligent vehicles symposium (IV)*, pages 163–168. IEEE, 2011.

[55] B. Li, W. Ouyang, L. Sheng, X. Zeng, and X. Wang. Gs3d: An efficient 3d object detection framework for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1019–1028, 2019.

[56] P. Li, X. Chen, and S. Shen. Stereo r-cnn based 3d object detection for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7644–7652, 2019.

[57] P. Li, H. Zhao, P. Liu, and F. Cao. Rtm3d: Real-time monocular 3d detection from object keypoints for autonomous driving. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 644–660. Springer, 2020.

[58] Z. Li, W. Wang, H. Li, E. Xie, C. Sima, T. Lu, Y. Qiao, and J. Dai. Bevformer: Learning bird's-eye-view representation from multi-camera images via spa-

tiotemporal transformers. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part IX*, pages 1–18. Springer, 2022.

[59] M. Liang, B. Yang, S. Wang, and R. Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *Proceedings of the European conference on computer vision (ECCV)*, pages 641–656, 2018.

[60] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327, 2020.

[61] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, and M. Pietikäinen. Deep learning for generic object detection: A survey. *International journal of computer vision*, 128:261–318, 2020.

[62] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016.

[63] Y. Liu, T. Wang, X. Zhang, and J. Sun. Petr: Position embedding transformation for multi-view 3d object detection. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVII*, pages 531–548. Springer, 2022.

[64] Z. Liu, H. Tang, Y. Lin, and S. Han. Point-voxel cnn for efficient 3d deep learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[65] J. Mao, S. Shi, X. Wang, and H. Li. 3d object detection for autonomous driving: A comprehensive survey. *International Journal of Computer Vision*, pages 1–55, 2023.

[66] R. McAllister, Y. Gal, A. Kendall, M. Van Der Wilk, A. Shah, R. Cipolla, and A. Weller. Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning. In *International Joint Conferences on Artificial Intelligence, Inc.*, 2017.

[67] G. P. Meyer, A. Laddha, E. Kee, C. Vallespi-Gonzalez, and C. K. Wellington. Lasernet: An efficient probabilistic 3d object detector for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12677–12686, 2019.

[68] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka. 3d bounding box estimation using deep learning and geometry. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7074–7082, 2017.

[69] M. Müller, A. Dosovitskiy, B. Ghanem, and V. Koltun. Driving policy transfer via modularity and abstraction. *arXiv preprint arXiv:1804.09364*, 2018.

[70] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. Cun. Off-road obstacle avoidance through end-to-end learning. *Advances in neural information processing systems*, 18, 2005.

[71] W. G. Najm, J. D. Smith, M. Yanagisawa, et al. Pre-crash scenario typology for crash avoidance research. Technical report, United States. National Highway Traffic Safety Administration, 2007.

[72] E. Ohn-Bar and M. Trivedi. Are all objects equal? deep spatio-temporal importance prediction in driving videos. *Pattern Recognition*, 64, 09 2016.

[73] W. H. Organization. *Global status report on road safety 2018*. World Health Organization, 2018.

[74] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1(1):33–55, 2016.

[75] X. Pan, Z. Xia, S. Song, L. E. Li, and G. Huang. 3d object detection with pointformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7463–7472, 2021.

[76] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[77] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 918–927, 2018.

[78] R. Qian, X. Lai, and X. Li. 3d object detection for autonomous driving: a survey. *Pattern Recognition*, 130:108796, 2022.

[79] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Vision Graphics and Image Processing*, 1(3):244–256, 1972.

[80] C. Reading, A. Harakeh, J. Chae, and S. L. Waslander. Categorical depth distribution network for monocular 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8555–8564, 2021.

[81] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[82] K. Renz, K. Chitta, O.-B. Mercea, A. S. Koepke, Z. Akata, and A. Geiger. Plant: Explainable planning transformers via object-level representations. In *Conference on Robotic Learning (CoRL)*, 2022.

[83] A. Rosenfeld and M. Thurston. Edge and curve detection for visual scene analysis. *IEEE Transactions on Computers*, C-20:562–569, 1971.

[84] 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 *2022 Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE)*, pages 181–186. IEEE, 2022.

[85] L. A. Rosero, I. P. Gomes, J. A. R. da Silva, T. C. d. Santos, A. T. M. Nakamura, J. Amaro, D. F. Wolf, and F. S. Osório. A software architecture for autonomous vehicles: Team lrm-b entry in the first carla autonomous driving challenge. *arXiv preprint arXiv:2010.12598*, 2020.

[86] W. Schwarting, J. Alonso-Mora, and D. Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:187–210, 2018.

[87] H. Shao, L. Wang, R. Chen, H. Li, and Y. Liu. Safety-enhanced autonomous driving using interpretable sensor fusion transformer. *arXiv preprint arXiv:2207.14024*, 2022.

[88] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10529–10538, 2020.

[89] S. Shi, X. Wang, and H. Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 770–779, 2019.

[90] W. Shi and R. Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1711–1719, 2020.

[91] A. Simonelli, S. R. Bulo, L. Porzi, M. López-Antequera, and P. Kontschieder. Disentangling monocular 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1991–1999, 2019.

[92] S. Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. Technical Report DOT HS 812 115, Department of Transportation, Washington, DC, USA, 2015.

[93] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020.

[94] P. Sun, W. Wang, Y. Chai, G. Elsayed, A. Bewley, X. Zhang, C. Sminchisescu,

and D. Anguelov. Rsn: Range sparse net for efficient, accurate lidar 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5725–5734, 2021.

[95] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[96] H. Tang, Z. Liu, S. Zhao, Y. Lin, J. Lin, H. Wang, and S. Han. Searching efficient 3d architectures with sparse point-voxel convolution. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVIII*, pages 685–702. Springer, 2020.

[97] O. D. Team. Openpcdet: An open-source toolbox for 3d object detection from point clouds. `https://github.com/open-mmlab/OpenPCDet`, 2020. Accessed: 2023-06-09.

[98] Tesla. Tesla ai day 2022. `https://www.youtube.com/live/ODSJsviD_SU?feature=share&t=3480`. Accessed: 2023-06-09.

[99] C. Thorpe, M. H. Hebert, T. Kanade, and S. A. Shafer. Vision and navigation for the carnegie-mellon navlab. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):362–373, 1988.

[100] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[101] M. Toromanoff, E. Wirbel, and F. Moutarde. End-to-end model-free reinforcement learning for urban driving using implicit affordances. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7153–7162, 2020.

[102] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova. Well-read students learn better: On the importance of pre-training compact models. *arXiv preprint arXiv:1908.08962v2*, 2019.

[103] Uber. A principled approach to safety. Available at `https://uber.app.box.com/v/UberATGSafetyReport?uclick_id=597e4853-c389-47d5-8e6f-d27aa2ea7b74`, 2021. Accessed: 2023-06-09.

[104] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[105] P. Viola and M. J. Jones. Robust real-time face detection. *International journal of computer vision*, 57:137–154, 2004.

[106] M. Vitelli, Y. Chang, Y. Ye, A. Ferreira, M. Wołczyk, B. Osiński, M. Niendorf, H. Grimmett, Q. Huang, A. Jain, et al. Safetynet: Safe planning for real-world self-driving vehicles using machine-learned policies. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 897–904. IEEE, 2022.

[107] S. Vora, A. H. Lang, B. Helou, and O. Beijbom. Pointpainting: Sequential fusion for 3d object detection. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4603–4611, Los Alamitos, CA, USA, jun 2020. IEEE Computer Society.

[108] D. Z. Wang and I. Posner. Voting for voting in online point cloud object detection. In *Robotics: Science and Systems*, pages 10–15, 07 2015.

[109] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8445–8453, 2019.

[110] Y. Wang, V. C. Guizilini, T. Zhang, Y. Wang, H. Zhao, and J. Solomon. Detr3d: 3d object detection from multi-view images via 3d-to-2d queries. In *Conference on Robot Learning*, pages 180–191. PMLR, 2022.

[111] Waymo. Safety report. Available at `https://waymo.com/safety/safety-report`, 2021. Accessed: 2023-06-09.

[112] K. Weng, I. Cao, C. Wang, and P. Sharma. Sms: A safety-enhanced modular stack for autonomous driving. In *CVPR 2023 Workshop*, 2023.

[113] 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 *NeurIPS*, 2022.

[114] B. Xu and Z. Chen. Multi-level fusion based 3d object detection from monocular images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2345–2353, 2018.

[115] Y. Yan, Y. Mao, and B. Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10), 2018.

[116] B. Yang, W. Luo, and R. Urtasun. Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018.

[117] Z. Yang, Y. Sun, S. Liu, and J. Jia. 3dssd: Point-based 3d single stage object detector. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11040–11048, 2020.

[118] T. Yin, X. Zhou, and P. Krähenbühl. Center-based 3d object detection and tracking. *CVPR*, 2021.

[119] Y. You, Y. Wang, W.-L. Chao, D. Garg, G. Pleiss, B. Hariharan, M. Campbell, and K. Q. Weinberger. Pseudo-lidar++: Accurate depth for 3d object detection in autonomous driving. *arXiv preprint arXiv:1906.06310*, 2019.

[120] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*, 8:58443–58469, 2020.

[121] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun. End-to-end interpretable neural motion planner. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8660–8669, 2019.

[122] W. Zeng, S. Wang, R. Liao, Y. Chen, B. Yang, and R. Urtasun. Dsdnet: Deep structured self-driving network. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXI 16*, pages 156–172. Springer, 2020.

[123] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021. `https://d2l.ai/chapter_recurrent-modern/gru.html`. Accessed: 2023-06-09.

[124] Y. Zhang, J. Lu, and J. Zhou. Objects are different: Flexible monocular 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3289–3298, 2021.

[125] X. Zhou, D. Wang, and P. Krähenbühl. Objects as points. In *arXiv preprint arXiv:1904.07850*, 2019.

[126] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4490–4499, 2018.

[127] Zoox. Safety report volume 2.0. Available at `https://zoox.com/safety/`, 2021. Accessed: 2023-06-09.

[128] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye. Object detection in 20 years: A survey. *Proceedings of the IEEE*, 2023.