THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Pensieve Perception: Uncertainty, Language, and Novel Views for Autonomous Driving

GEORG HESS

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, 2025

Pensieve Perception: Uncertainty, Language, and Novel Views for Autonomous Driving

GEORG HESS
ISBN 978-91-8103-209-3

Acknowledgements, dedications, and similar personal statements in this thesis, reflect the author's own views.

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5667
ISSN 0346-718X

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000

Cover:
A drawing showing an autonomous driving scene within a Pensieve. Generated using ChatGPT.

Printed by Chalmers Digital Printing
Gothenburg, Sweden, May 2025

Pensieve Perception: Uncertainty, Language, and Novel Views for Autonomous Driving

GEORG HESS
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Autonomous driving stands as one of the most complex challenges in artificial intelligence and robotics. These systems are expected to provide safe and efficient navigation across diverse and dynamic environments. This thesis addresses critical obstacles in the development of autonomous driving technologies by focusing on three primary areas: uncertainty estimation in object detection, reduction of annotation requirements through self-supervised learning, and the creation of scalable, realistic simulations via neural rendering. First, we introduce a novel framework for uncertainty estimation in object detection, leveraging the Random Finite Set formalism to provide a principled approach for training and evaluating probabilistic object detectors. This framework enables practitioners to better understand the capabilities and limitations of object detectors, and effectively design and deploy safer and more reliable systems. Second, we present LidarCLIP, a self-supervised learning method designed to bridge the gap between lidar point clouds and language understanding. By aligning lidar data with the CLIP embedding space through image-point cloud pairs, LidarCLIP learns semantic scene understanding without the need for costly human annotations. This approach has the potential to significantly reduce the dependency on labeled data, accelerating the development and deployment of autonomous driving systems. Last, we develop NeuRAD and SplatAD, advanced neural rendering techniques for joint camera and lidar simulation. NeuRAD offers a state-of-the-art neural simulator that facilitates scalable and sensor-realistic closed-loop simulations, while SplatAD enhances this capability by improving both visual quality and computational efficiency. These methods pave the way for scalable validation and testing of autonomous driving systems in diverse and rare scenarios, facilitating comprehensive safety assessments. Collectively, this thesis addresses existing challenges in autonomous driving and opens up new avenues for future research in building safe and reliable autonomous driving systems at scale.

Keywords: Autonomous driving, object detection, uncertainty estimation, self-supervised learning, novel views synthesis

To Boel.

List of Publications

This thesis is based on the following publications:

[A] **Georg Hess**, Christoffer Petersson, Lennart Svensson, "Object Detection as Probabilistic Set Prediction". ECCV 2022.

[B] **Georg Hess**[†], Adam Tonderski[†], Christoffer Petersson, Kalle Åström, Lennart Svensson, "LidarCLIP or: How I Learned to Talk to Point Clouds". WACV 2024.

[C] Adam Tonderski[†], Carl Lindström[†], **Georg Hess**[†], William Ljungbergh, Lennart Svensson, Christoffer Petersson, "NeuRAD: Neural Rendering for Autonomous Driving". CVPR 2024.

[D] **Georg Hess**[†], Carl Lindström[†], Maryam Fatemi, Christoffer Petersson, Lennart Svensson, "SplatAD: Real-Time Lidar and Camera Rendering with 3D Gaussian Splatting for Autonomous Driving". CVPR 2025.

Other publications by the author, not included in this thesis, are:

[E] Z. Zhang, **G. Hess**, J. Hu, E. Dean, L. Svensson, K. Åkesson, "Future-Oriented Navigation: Dynamic Obstacle Avoidance with One-Shot Energy-Based Multimodal Motion Prediction". *Submitted to IEEE Robotics and Automation Letters (RA-L)*, 2024.

[F] C. Lindström[†], **G. Hess**[†], M. Fatemi, L. Hammarstrand, C. Petersson, L. Svensson, "Are NeRFs ready for autonomous driving? Towards closing the real-to-simula-tion gap". *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2024.

[G] M. Alibeigi, W. Ljungbergh, A. Tonderski, **G. Hess**, A. Lilja, C. Lindström, D. Motorniuk, J. Fu, J. Widahl, C. Petersson, "Zenseact Open Dataset: A large-scale and diverse multimodal dataset for autonomous driving". *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[H] J. Pinto, **G. Hess**, W. Ljungbergh, Y. Xia, H. Wymeersch, L. Svensson, "Deep Learning for Model-Based Multi-Object Tracking". *IEEE Transactions on Aerospace and Electronic Systems*, 2023.

[†]These authors contributed equally to this work.

[I] **G. Hess**, J. Jaxing, E. Svensson, D. Hagerman, C. Petersson, L. Svensson, "Masked Autoencoder for Self-Supervised Pre-training on Lidar Point Clouds". *IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW)*, 2023.

[J] J. Pinto, **G. Hess**, W. Ljungbergh, Y. Xia, L. Svensson, H. Wymeersch, "Next Generation Multitarget Trackers: Random Finite Set Methods vs Transformer-based Deep Learning". *IEEE 24th International Conference on Information Fusion (FUSION)*, 2021.

[K] J. Berlin†, **G. Hess**†, A. Karlsson†, W. Ljungbergh†, Z. Zhang, K. Åkesson, P.L. Götvall, "Trajectory generation for mobile robots in a dynamic environment using nonlinear model predictive control". *IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021.

Contents

Abstract i

List of Publications v

Acknowledgements xiii

Acronyms xiv

I Overview 1

1 Introduction 3
1.1 Challenges in Autonomous Driving 4
1.2 Outline 8

2 Autonomous Driving 9
2.1 Building Blocks of an Autonomous Vehicle 10
Sensors 11
Software Stack 13
2.2 Learning to Drive 16
Collecting Data 16
Testing the Performance 19

2.3 Pain Points of Autonomous Driving . . . 22
Uncertainty . . . 22
Scalable Training . . . 23
Scalable Testing . . . 24

3 Perceiving the Surroundings 25
3.1 Learning from Human Supervision . . . 26
3.2 Modeling Uncertainty . . . 27
3.3 Object Detection . . . 29
Detection Models . . . 30
Evaluation Metrics . . . 30
Probabilistic Object Detection . . . 32

4 Finding the Right Data 37
4.1 Self-Supervised Learning . . . 37
4.2 Weakly Supervised Learning . . . 42
Learning from Language . . . 42
Auto-Labeling . . . 43
4.3 Applications to Autonomous Driving . . . 44

5 Going Beyond Real Data 47
5.1 Novel View Synthesis . . . 48
Neural Radiance Fields . . . 49
3D Gaussian Splatting . . . 50
Generative Modeling for NVS . . . 51
5.2 Neural Rendering in AD Simulation . . . 52

6 Summary of included papers 57
6.1 Paper A . . . 57
6.2 Paper B . . . 58
6.3 Paper C . . . 58
6.4 Paper D . . . 59

7 Concluding Remarks 61

References 65

II Papers **83**

A Object Detection as Probabilistic Set Prediction **A1**
1 Introduction . . . A3
2 Related Work . . . A5
3 Probabilistic Modeling for Object Detection . . . A7
3.1 Modeling Detections with Random Finite Sets . . . A8
3.2 Proper Scoring Rule for Object Detection . . . A10
3.3 Modeling All Objects . . . A12
4 Experiments . . . A15
5 Conclusions . . . A18
Appendix A - Cost Matrix . . . A20
Appendix B - Experimental Details . . . A22
B.1 - Model Implementation . . . A22
B.2 - Training Details . . . A22
B.3 - Inference Details . . . A23
Appendix C - Additional Results . . . A23
C.1 - Qualitative Results . . . A23
C.2 - PMB-NLL Decomposition . . . A23
Appendix D - Comparison to DETR loss . . . A27
D.1 - DETR Loss Revisited . . . A27
D.2 - MB-NLL Relation to DETR . . . A31
References . . . A34

B LidarCLIP **B1**
1 Introduction . . . B3
2 Related work . . . B6
3 LidarCLIP . . . B7
3.1 Joint retrieval . . . B9
4 Experiments . . . B10
4.1 Zero-shot classification . . . B11
4.2 Retrieval . . . B12
4.3 Generative applications . . . B15
5 Limitations . . . B17
6 Conclusions . . . B17
Appendix A - Model details . . . B22
Appendix B - Training details . . . B22

Appendix C - Baseline details B23
C.1 - PointCLIP B23
C.2 - CLIP2Point B23
C.3 - Generative applications B24
Appendix D - Additional results B24
D.1 - Zero-shot object classification B24
D.2 - Retrieval B25
D.3 - Zero-shot scene classification B26
D.4 - LidarCLIP for lidar sensing capabilities B27
D.5 - Lidar to image and text B27
References B39

C NeuRAD **C1**
1 Introduction C4
2 Related work C5
3 Method C7
3.1 Scene representation and sensor modeling C8
3.2 Extending Neural Feature Fields C9
3.3 Automotive data modeling C10
3.4 Losses C13
3.5 Implementation details C13
4 Experiments C14
4.1 Datasets and baselines C14
4.2 Novel view synthesis C15
4.3 Novel scenario generation C16
4.4 Ablations C16
5 Conclusions C17
Appendix A - Implementation details C21
Appendix B - Evaluation details C22
Appendix C - UniSim implementation details C23
C.1 - Data processing C24
C.2 - Model components C24
C.3 - Supervision C26
Appendix D - Inferring ray drop C27
Appendix E - Modeling rolling shutter C28
Appendix F - Simulation gap C29

Appendix G - Additional results C29
G.1 - Limitations C31
References C40

D SplatAD **D1**
1 Introduction D4
2 Related work D5
3 Method D6
3.1 Scene representation D7
3.2 Camera rendering D8
3.3 Lidar rendering D10
3.4 Optimization and implementation D13
4 Experiments D14
4.1 Image rendering D16
4.2 Lidar rendering D17
4.3 Ablations D17
5 Conclusion D18
Appendix A - Lidar rendering details D22
A.1 - Lidar tiling D22
A.2 - Lidar points to rasterization points D23
Appendix B - Training details D24
Appendix C - Baseline details D26
Appendix D - Rolling shutter details D26
Appendix E - Additional qualitative results D27
Appendix F - Evaluation details D27
References D34

Acknowledgments

And now my PhD journey is ended. Looking back over the last 4 years, I would like to express my gratitude to everyone who has made this adventure possible and has supported me along the way. They say it takes a village to raise a child, but it's probably not far off for a PhD student either.

Thank you, Lennart and Christoffer for your invaluable guidance and support throughout the ups and downs of this journey. You have provided me with great advice, encouraged me to explore my interests, and played music with me more than I would have anticipated when I became a PhD student. Although there are not many traces of the original project plan in this thesis, I think it turned out quite well in the end.

A big thank you to all the great people that I have met during this time, be it at conferences, study trips, courses, or just around the office at Zenseact or Chalmers. I especially want to thank everyone in the AGP crew - old and new - for the open and welcoming environment, it has been an AGPleasure. Thank you Carl, Jonas, and Mats, for creating this group of support, for your coaching and leadership, and for letting me pursue all wonderful opportunities around the world. A special shoutout, of course, to Adam, Carl, William, and Lilja. Be it traveling the world, deadline mode all-nighters, whiteboard sessions, or acceptance AWs, it would not have been the same without you.

I want to express my gratitude to all my coauthors - thank you for your collaborations and for your contributions to this thesis. A big thank you also goes to all my colleagues at the signal processing group for great discussions and new perspectives on my work.

Last, but not least, I want to thank my family and friends for their love and support throughout these years. To the Knutsons, for your encouragement and interest in my work. To the taco-t gang, for celebrating my wins and listening to my rants about reviewer 2. To my parents for always believing in me and raising me to be independent and follow my dreams. To Boel, my fantastic wife and pillar of support. Thank you for putting up with me during deadline weeks, for keeping me grounded outside of work, and most of all, for making these years full of joy and love.

Georg Hess
Gothenburg, April 2025

Funding

This work was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and by Zenseact AB through their industrial PhD program.

Acronyms

AD:	Autonomous Driving
ADAS:	Advanced Driver Assistance System
ADS:	Autonomous Driving System
BEV:	Bird's-Eye View
CLIP:	Contrastive Language-Image Pre-training
CNN:	Convolutional Neural Network
CV:	Computer Vision
DL:	Deep Learning
eHMI:	External Human-Machine Interface
GNSS:	Global Navigation Satellite System
IoU:	Intersection over Union
LiDAR:	Light Detection And Ranging
LLM:	Large Language Model
mAP:	Mean Average Precision
MAE:	Masked Autoencoder
MLP:	Multi-Layer Perceptron
NeRF:	Neural Radiance Field

NLP:	Natural Language Processing
NVS:	Novel View Synthesis
RFS:	Random Finite Set
SGD:	Stochastic Gradient Descent
SSL:	Self-Supervised Learning
ViT:	Vision Transformer
WSL:	Weakly Supervised Learning

Part I

Overview

CHAPTER 1

Introduction

> I use the Pensieve. One simply siphons the excess thoughts from one's mind, pours them into the basin, and examines them at one's leisure. It becomes easier to spot patterns and links, you understand, when they are in this form.
>
> *Albus Dumbledore*

Autonomous driving represents one of the most complex challenges in artificial intelligence and robotics. These systems must ensure safe and comfortable journeys in any situation, whether the vehicle cruises down a sunny Route 66 at great speed, navigates a bustling downtown Tokyo at night, or drives on winding and snowy roads in northern Sweden. Such a system requires algorithms that enable the vehicle to perceive its environment, plan ahead, and take action in a split second. Autonomous driving has the potential to democratize transportation and drastically reduce road accidents, saving millions of lives. The rise of robotaxis has been shown to provide safer roads by using systems that are better at avoiding collisions than what humans are [1]. However, to bring these technologies to the wider public, their development

must be cost-efficient and scalable.

Much of the recent progress in the field of autonomous driving has been fueled by the rise of deep learning. This paradigm shift has allowed algorithms, most commonly in the form of neural networks, to learn directly from data, reducing reliance on manually crafted rules for complicated tasks [2]. However, while deep learning has driven rapid advancements, its use is also associated with a range of challenges, some of which are especially crucial for a safety-critical application like autonomous driving. These challenges span from acquiring large enough amounts of diverse data paired with expensive human annotations to train the neural networks, to finding accurate and efficient ways to evaluate their performance. This thesis studies multiple such challenges and proposes solutions to overcome them, with special emphasis on problems arising in the autonomous driving context. While the contributions made in this work are spread out over multiple topics, they are bound together by the aim to accelerate the development of safe and scalable autonomous driving.

1.1 Challenges in Autonomous Driving

This thesis addresses the following key challenges in autonomous driving:

Uncertainty Estimation in Object Detection Perception is one of the core tasks in autonomous driving, as it enables the system to understand the surroundings of the vehicle. Perception systems can, for instance, generate descriptions of the road geometry, classify weather conditions, or detect the location and state of traffic lights. What type of information is generated usually depends on what practitioners have deemed important enough to influence the vehicle decisions.

One of the most common perception tasks in autonomous driving is object detection, where the goal is to predict the class and location of objects such as other traffic participants, lane markings, traffic signs, etc., from sensory input from a suite of sensors like cameras, lidars, radars, and ultrasonics. Object detection is, like many perception tasks, one with multiple sources of ambiguities, like the partial occlusion of a car behind a truck or by a water droplet on the camera lens. Furthermore, object detectors inherently exhibit uncertainty due to both their design and training process. Given that autonomous driving is a safety-critical application, it is crucial to understand the limitations of its subsystems, such as object detectors. Uncertainty estimation is a tool that enables this type of reasoning, which also is useful for making informed decisions.

Although object detection is a long-studied problem [3], uncertainty estimation

has often been overlooked. Also known as probabilistic object detection, it requires reasoning about multiple sources of uncertainty jointly. These sources include the number of objects and the individual objects' attributes like class, location, and extent. The task is further complicated by the fact that we do not know the correspondence between the predicted and ground truth objects. While the topic has received some attention, previous approaches to probabilistic object detection [4], [5] lack rigorous theoretical justification. For instance, [4] proposes the task of probabilistic object detection and the probability-based detection quality (PDQ), but the latter is biased towards small objects [5]. In [5], the authors use principled methods to evaluate the performance of individual detections but rely on ad hoc methods for assigning predictions to ground truth objects. As a result, fair evaluation of probabilistic object detectors is not possible with existing methods.

In Paper A [6], we propose a novel framework for modeling the uncertainty in object detection. Our model is based on the Random Finite Set formalism [7], and comes with a principled method for training and evaluating probabilistic object detectors. Specifically, we view the prediction as a single random variable and show how to reason about assigning predictions to labels in the presence of uncertainty. Our framework is further compatible with existing probabilistic object detection models, and only requires reinterpreting their predictions.

Self-Supervised Learning for Reduced Annotation Regardless of the type of modeling, training deep learning algorithms for autonomous driving typically requires large amounts of annotated data, which is both time-consuming and expensive to obtain, hindering its scalable development. To reduce the annotation needs, the community has primarily explored two complementary techniques, namely active learning [8] and automatic labeling [9], [10]. The first refers to methods aiming to maximize the performance of a neural network with fewer annotations by letting the neural network decide which data to annotate. In this context, this often means labeling hard and/or rare cases. The latter, automatic labeling, refers to using large and powerful models, which are too slow to run in real-time, to label the bulk of easy and common cases. Moreover, these types of models can often use tricks such as hindsight or rely on higher-quality sensors to improve their performance.

Despite these advancements, there remains a significant dependency on human annotations. Self-supervised learning (SSL) has emerged as a powerful paradigm for extracting knowledge from unlabeled data, enabling the development of large generalist models that require minimal fine-tuning for specific tasks. Specifically, SSL is one of the key ingredients in the explosive development of large language models

[11]–[13] and empowers many pure vision models [14]–[17]. More so, some approaches utilize self-supervised learning in a multi-modal setting. One such approach is CLIP (Contrastive Language-Image Pre-training) [18], which leverages web-scale text-image pairs to connect the language and visual domains and learn from them jointly. CLIP learns human-like semantics of images that reach beyond the rigidly defined classes in human-annotated perception tasks and does so without explicit annotations. The capabilities offer intuitive ways to interact with large amounts of unlabeled data, *e.g.*, finding the image that best matches a textual description, and, further, offer a certain degree of interpretability.

A key challenge in applying similar techniques in an AD setting is the lack of language supervision for sensors other than the camera. Many autonomous driving systems rely on additional sensors, such as lidar and radar, for redundancy and the sensors' complementary strengths and weaknesses. However, there exist no web-scale radar-language or lidar-language pairs, which hinders the training of CLIP-like models for these sensors. Some works have studied how to use existing CLIP image models for point cloud understanding by applying them to 2D projections of the point clouds [19], [20]. While this approach has shown promise for object-centric data where the same object easily can be viewed from multiple directions, it is not applicable for understanding large outdoor scenes in the autonomous driving context.

In Paper B [21], we propose LidarCLIP, a self-supervised learning approach for connecting outdoor lidar point clouds and text, without the need for any text-point cloud pairs. To overcome the lack of language supervision for lidar, we propose using images as an intermediary representation. Specifically, using prevalent image-point cloud pairs, we supervise a point cloud encoder to match the image CLIP embedding space. As an effect, LidarCLIP displays strong capabilities for semantic scene understanding, anomaly detection, data curation, and potential as a lidar generalist model, without requiring any human labeling. For example, LidarCLIP can identify rare road scenarios or weather conditions across massive unlabeled datasets, focusing human attention where it matters most.

Neural Rendering for Scalable Development Even if we can reduce the reliance on human annotation, the training and validation of autonomous driving systems still require the collection of massive amounts of data. For instance, the training process must contain a diverse set of scenarios for the system to generalize well. Similarly, the validation process must cover a wide range of scenarios for developers to trust that the system is safe and performs well in all settings before deployment. However, trying to collect data from safety-critical scenarios by driving

in the real world scales poorly, especially with an ever-increasing operational design domain, as such encounters become prohibitively rare. This can be overcome to a certain degree by using test tracks or proving grounds, *i.e.*, closed-off areas designated for controlled testing activities. Here, near-crash scenarios can be explored safely using inflatable cars or other crash dummies. While giving some indication of real-world performance for these critical scenarios, test tracks lack the complexity and diversity of real-world driving and are a costly tool to apply at scale.

The last challenge addressed in this work is that of efficiently creating data that cannot be collected in the real world. Traditionally, this is solved by using synthetic data from simulators, where practitioners have full control over the environment. Here, scenes are composed by combining human-made assets whose behavior is dictated by predefined rules or data-driven models [22]. While the validation of the system's motion planning and control requires only low-level signals such as moving bounding boxes, the perception systems further require realistic sensor data. To achieve this, sensor data are often generated using physics-based game engine approaches. However, reaching a level of sensor-realism sufficient to trust the results achieved in simulation is a massive undertaking, requiring deep domain knowledge for accurate simulation, and a workforce of human artists to create diverse assets.

Advancements in neural reconstruction and rendering offer an attractive alternative to the traditional game engine-based simulators [23]–[25]. Here, one learns a representation directly from raw sensor data that allows for the rendering of novel views from the same scene where the location of traffic participants, ego-vehicle sensors, or both, have been altered. Although this approach has achieved impressive visual quality, existing methods suffer from some drawbacks. First, these methods are computationally expensive. Many require hours of training to reconstruct less than a minute of video, and the inference time is not suitable for real-time applications. Second, existing methods do not apply to the common AD sensor setup of multiple cameras and lidar. Thus, they cannot be used as a simulation engine for a wide variety of sensor platforms.

To address this, we propose new neural rendering approaches for joint camera and lidar rendering. In Paper C [26], we present NeuRAD, a state-of-the-art neural simulator for joint camera and lidar rendering. Through modeling of important sensor characteristics such as rolling shutter effects, beam divergence, camera-specific exposure, and non-returning lidar rays, NeuRAD outperforms previous methods by a large margin across multiple datasets. As shown in [27], NeuRAD can be used to rapidly reconfigure observed nominal driving scenarios into safety-critical scenarios that require emergency maneuvers, thereby significantly accelerating safety

validation. Although NeuRAD advances neural rendering by enabling sensor-realistic simulations, it faces challenges in training and inference time. To address these, we develop SplatAD (Paper D) [28], which achieves breakthrough improvements in computational efficiency and further improves visual quality. SplatAD significantly reduces training time and inference latency compared to NeuRAD, while producing more photorealistic renderings as measured by standard metrics. These improvements make neural rendering practical for large-scale validation and training, enabling autonomous driving developers to dramatically expand their testing coverage of rare and safety-critical scenarios.

1.2 Outline

The remainder of the thesis is divided into two parts: the first introduces the background and motivation, while the second presents the included research papers. The first part is divided into six chapters, where each of the first four introduces a new topic. In the first chapter, we describe the task of autonomous driving and the components that make up an autonomous driving system. In the second chapter, we introduce how autonomous driving systems perceive their surroundings and how to train such systems. The topics related to this include supervised learning and uncertainty estimation. In the third chapter, we cover self-supervised and weakly supervised learning in general, and language supervision in particular. In the fourth chapter, we introduce neural rendering and how it can be used to create realistic simulations of autonomous driving scenarios. In the fifth chapter, we summarize the included papers and their contributions. The last chapter contains concluding remarks on both potential improvements and future applications of the included works.

CHAPTER 2

Autonomous Driving

While the works in this thesis do not directly solve the problem of autonomous driving (AD), their common denominator is the enablement of AD development. Thus, this chapter introduces the task of autonomous driving, common ways of designing an AD system, and ways to continuously improve the system's performance. However, it is important to note that the specifics of these components can vary significantly across different AD systems. This chapter aims to provide a broad introduction to autonomous driving and highlights general principles applicable to a wide range of implementations. Further, in this thesis, the focus is on self-driving vehicles that share the road with human drivers. However, many of the discussed systems and methods to a large degree remain applicable also for applications such as mining, agriculture, warehouse operations, or even general physical AI systems. In addition, these techniques are also common in advanced driver assistance systems (ADAS), where the discerning factor compared to AD is that ADAS has a lower level of automation and a higher degree of human supervision.

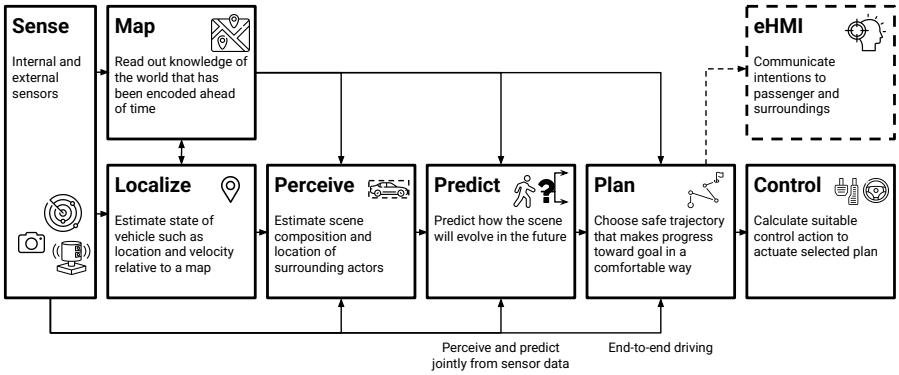

Figure 2.1: An autonomous driving system can be divided into multiple building blocks. The vehicle senses both the external and internal environment using sensors. Besides sensor information, the AD system can rely on information about the world that has been encoded ahead of time, *e.g.*, map information. This information can be used in all subsequent blocks with localization, perception, prediction, and planning. Depending on the specific implementation, the subsystems might be separate with clearly defined interfaces (modular stack), consist of a single large block (end-to-end), or something in between. Regardless of variation, the final output from the stack is actuation commands for steering the vehicle, and optionally an eHMI interface to communicate with passengers and surrounding traffic participants. Figure inspired by [29].

2.1 Building Blocks of an Autonomous Vehicle

On a high level, autonomous vehicles consist of four core components: (i) the physical vehicle platform, (ii) an array of sensors that monitor both the external environment and the vehicle's internal state, (iii) computational hardware that processes sensor data and makes decisions, and (iv) actuators that execute control commands. Each component must meet strict requirements in terms of reliability and performance to ensure safe autonomous operation. Additionally, the vehicle can be equipped with an eHMI (external human-machine interface) to communicate its intent to passengers and the surroundings. In this thesis, we define the AD system as the set of algorithms running on the computational device used to process inputs and produce actuation commands. The inputs to the AD system are (i) a mission specifying to which location it should drive, (ii) sensor data, and (iii) potentially a map or other prior knowledge. See Fig. 2.1 for an overview.

Sensors

The types of sensors deployed in an AD system can vary between different realizations, governed by factors such as cost, intended operational domain, and level of automation. For instance, consumer vehicles naturally have tighter sensor budgets than robotaxis since these are produced at a much greater scale. On the other hand, aiming to operate in a wide range of environmental conditions, including adverse weather, is associated with strict requirements on sensor robustness. Further, autonomous driving systems without any human supervision have a greater need for redundancy for safe operations. As an example, Tesla is developing a vision-only system and has discarded any distance-measuring sensors such as lidar, radar, or ultrasonic sensors [30][1]. While this reduces the hardware costs compared to including lidars or radars, it also makes developing safe AD much more challenging. In comparison, robotaxi companies such as Waymo build orders of magnitude fewer cars and do not sell them to consumers, and therefore can afford to rely on more expensive sensor setups with multiple cameras, lidars, and radars.

In this thesis, we assume that autonomous vehicles are equipped with at least one camera, one lidar, an Inertial Measurement Unit (IMU), and a Global Navigation Satellite System (GNSS) device. However, it is common to also employ multiple cameras, wheel speed sensors, radars, and ultrasonic sensors. Besides these sensors, some works examine the use of event cameras [31] and thermal cameras [32], but their use is still not widespread. In Fig. 2.2, the sensor setup from the Zenseact Open Dataset [33] is shown as an example.

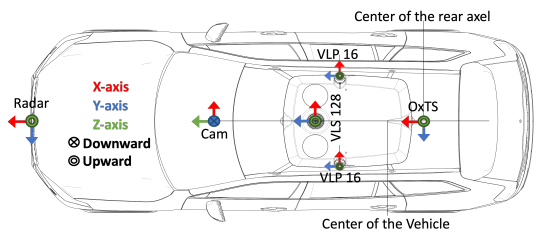

Figure 2.2: Sensor setup in Zenseact Open Dataset [33] with one front-looking radar, one camera, three lidars (one VLS128 and two VLP16), and one positioning system (OxTS).

[1]This choice has sparked some controversy, as cameras are notoriously sensitive to poor weather like rain, fog, or darkness.

Camera Cameras are used to capture images and videos. They are popular in autonomous driving as they can perceive objects at a great distance and are relatively cheap. On a high level, their core components are a sensor to measure the amount of incoming light and a lens used to focus the light onto the sensor. The lens determines how much of the scene is visible in an image and how large objects appear depending on their relative position to the camera. As for the sensor, it can be exposed for a varying amount of time, where long exposure times lead to brighter and less noisy images but also make it harder to take sharp images. Any movement of the environment or the camera itself will lead to smearing out the image, known as motion blur. Further, some sensors use a rolling shutter, meaning that instead of exposing the entire image at the same time, the image is exposed gradually, *e.g.* row-by-row. Rolling shutter cameras are typically cheaper than global shutter cameras and can often be run at higher frame rates, but introduce distortions known as the rolling shutter effect. A common example of this is the skewing of nearby objects when traveling at a high velocity.

Lidar Light detection and ranging (lidar) sensors are used to perceive the 3D structure of the environment. As the name suggests, lidar sensors use light to perform measurements. Unlike cameras, which are passive and only capture reflected light, lidars actively emit laser pulses. The emitted pulses of the infrared laser are used to measure the time-of-flight to determine the distance to surfaces. Simultaneously, the returning power is also measured, to discern between different highly reflective and less reflective surfaces, providing additional information about material properties and object classifications. The most common versions are spinning lidars, where an array of diodes is rotated around the sensor to capture 360° of the scene. The fact that lidars actively illuminate the scene makes them great for use in darkness, a setting cameras struggle with. In addition, they provide direct 3D information, making them useful for understanding the distance to different objects or measuring the road geometry. However, the range of automotive lidars is typically limited to ~ 250 meters, which can be compared to cameras which can typically see much further in bright conditions. Moreover, although their price has reduced drastically over the last years, lidars are still orders of magnitude more expensive than cameras.

Radar Radar, short for radio detection and ranging, sensors are another common sensor in AD systems. Rather than using light, they emit radio waves and measure the returning waves to infer the distance, direction, and radial velocity of objects relative to the sensor. Radars are popular because of their robustness to adverse weather

conditions, such as rain, fog, or snow. Further, in comparison to lidars, they are much cheaper. However, radar measurements are also more sparse and less accurate, and until recently, were limited to 2D spatially, ignoring the height component of the environment. Newer generations of so-called 4D radars (3D + velocity) overcome this limitation but are not yet widespread.

Software Stack

There are two dominant ways to design an AD stack, (i) the traditional modular or layered pipeline and (ii) the end-to-end approach [34]. For clarity, when referring to an AD system, we mean it to be modular, unless otherwise stated.

Modular pipeline

In the modular approach, the AD system is divided into subsystems, each performing a specific task, such as traffic light detection, road geometry estimation, or pedestrian motion prediction. As the information progresses through the stack, there are clear APIs between subsystems defining the expected inputs and outputs. The traditional pipeline is favored for its interpretability and ease of debugging, allowing developers to inspect the outputs of each module to identify the root cause of issues. In addition, each subsystem is usually connected to a well-established field of research and development, allowing practitioners to draw upon years of experience. However, since the subsystems are optimized separately for their specific task, it is not clear how well their performance aligns with the overall end goal of producing a safe and comfortable motion plan. In relation to this, it is also complicated to balance the capacity and associated performance of the different subsystems in a resource-constrained system.

Figure 2.1 shows the high-level parts that typically make up a modular pipeline. Note, however, that multiple modules can be combined into one, *e.g.*, perception and prediction. Below, we give a more detailed description of the different parts.

Sensor Preprocessing Before the sensor data can be used, it must be put into a suitable format for an AD system. For images, this can mean running the raw light intensity values through an image signal processor which applies, for example, Bayer transformation, noise reduction, and tone mapping. For lidars, point clouds might be motion-compensated, i.e., transformed to a common reference frame, and pruned from returns from the ego-vehicle. Depending on the sensor supplier, a varying degree of preprocessing might also be done directly by the sensor's own processor before reaching the rest of the AD system. Preprocessing can also entail

different types of calibration, such as estimating changes in camera parameters due to temperature changes. Furthermore, some AD systems might require different sensors to be synchronized, rather than processing them in a streaming fashion.

Localization To make informed decisions, the AD system must understand where it is. Localization refers to computing the pose of the ego-vehicle relative to a reference. Typically, localization is done both in a local and a global frame, using different sources of information. The local pose is updated at a high frequency to produce a smooth trajectory and might be the basis used for ego-motion compensating the lidar point cloud. The global pose is instead computed relative to a map, providing information on position relative to lanes and other infrastructure. Again, there exist different approaches, where some systems rely heavily on accurate high-definition maps (HD maps) that have detailed information, such as lane connectivity, lane markers, and other static structures. Others instead might use maps with a lower level of detail (SD maps) mainly for high-level navigation.

Perception The perception module creates a description of the surrounding environment. A common part of this is classifying and localizing objects. Examples of objects are (potentially) dynamic vehicles and pedestrians, and static structures such as lane dividers and traffic lights. However, this module can also produce other types of descriptions, *e.g.*, classifications of which part of the ground is considered drivable or which part of the surrounding space is occupied. The input to the perception module is sensor data, either individual sensors, bundles, or some merged representation. Perception modules can also rely on some priors, such as maps, to fuse multiple sources of information.

Tracking Perception outputs are fused over time using tracking. By connecting individual detections, tracking gives a historical context for the movements of other actors, aiding accurate motion forecasting. Tracking also enables the reasoning about the existence of previously seen objects, although they might currently be obscured. Moreover, tracking can also be used simply to improve the quality of the perception predictions. In some implementations, tracking is directly integrated into the perception module [35].

Prediction Also known as motion forecasting, prediction is used to reason about likely future states of the environment, *i.e.*, answering questions such as “will that car yield for me or drive in my way?”. [36]. Prediction is an ambiguous task by nature,

as there are multiple plausible answers. To further complicate things, real data only contains one possible unfolded future, complicating both the training and evaluation of prediction systems. Similar to the tracking, the prediction module can be fused with the perception module. Instead of using tracked detections, the prediction model must rely on sensor data directly.

Planning Given the estimates of the current and future environment, the planning model is tasked with creating a plan for the ego-vehicle [37]. This plan is subject to several constraints, as it should (i) make progress toward the mission goal, (ii) adhere to traffic rules, (iii) be safe for the passenger and other traffic participants, (iv) be comfortable for the passengers, and (v) be physically plausible. This is further complicated by the real world being reactive, *i.e.*, our own actions will impact the actions taken by surrounding actors. Planning can further be divided into different time horizons. Long-term planning, often called route planning or navigation, determines the overall path from the current location to the destination. Mid-term planning focuses on maneuver decisions such as lane changes or turns at intersections. Short-term planning, or trajectory planning, computes the precise path the vehicle should follow in the immediate future, typically spanning a few seconds.

Control The last module is the controller, which computes high-frequency, low-level control actions for the actuators to follow the proposed plan as best as possible [38]. The controller typically operates at a much higher frequency than the planning module to ensure smooth and precise execution of the planned trajectory. It must account for the physical dynamics of the vehicle, including inertia, friction, and other forces that affect motion. Controllers can range from classical approaches like PID (Proportional-Integral-Derivative) controllers to more sophisticated model predictive control (MPC) methods that optimize control actions over a receding horizon [38].

End-to-end systems

There exist different interpretations of end-to-end systems, but here we use the definition of [34], that end-to-end AD systems are fully differentiable programs that transform raw sensor data to motion plans or low-level control signals. While the name might indicate a single, large, black-box system, we note that an end-to-end system may still consist of different modules (the same as for modular systems even). Their defining trait is that all systems can be trained jointly with gradients flowing from the last to the first module. Further, not all subsystems must be learnable, as long as their operations or the system as a whole [39], can be differentiated.

End-to-end systems have garnered increasing attention as they address several challenges associated with the modular approach [40], [41]. Most prominently, as they are completely differentiable, they can be directly optimized toward the ultimate goal of comfortable and safe driving. While questions regarding optimal resource allocation among different components may persist, end-to-end systems can be optimized more directly for key performance indicators relevant to autonomous driving. Moreover, the interfaces between different parts are more fluid, and the system can learn (to a certain degree) what information should be conveyed. This way, the planning is less dependent on the specific perception outputs, and, in theory, less susceptible to perception errors. Moreover, end-to-end systems do not necessarily require annotations for training each subsystem, as for the modular approach.

However, the end-to-end approach instead raises issues regarding interpretability, debugging, safety assurance, and robustness. For instance, when an end-to-end system fails, it is hard to pinpoint where something went wrong. Some methods try to address this by incorporating interpretable tasks, decoding the intermediate representation to semantically meaningful outputs [34]. However, even if you can decode information about the surrounding environment, it remains hard to tell how that information is used to decide upon a plan. Similarly, end-to-end systems lack the mathematical tools that can be used to prove the safety of a rule-based system [42].

2.2 Learning to Drive

With the AD stack definition in place, the remaining challenge is to figure out how to maximize the driving performance. Regardless of modular or end-to-end, practitioners have moved more and more toward data-driven methods like machine learning, or, in practice, deep learning. Naturally, this shapes the way autonomous vehicles are developed. This section describes the basic life cycle of a deep learning-powered AD software, illustrated in Fig. 2.3. In addition to providing an understanding of the methods themselves, the section also lays the basis for understanding the unique challenges and pain points of AD development.

Collecting Data

One of the core activities in the development of AD is data collection. The data collected is used to train learning-based systems and to evaluate and validate system performance. Real-world data collection is done using either development vehicles that are purposefully driven by professional drivers, consumer vehicles driven by

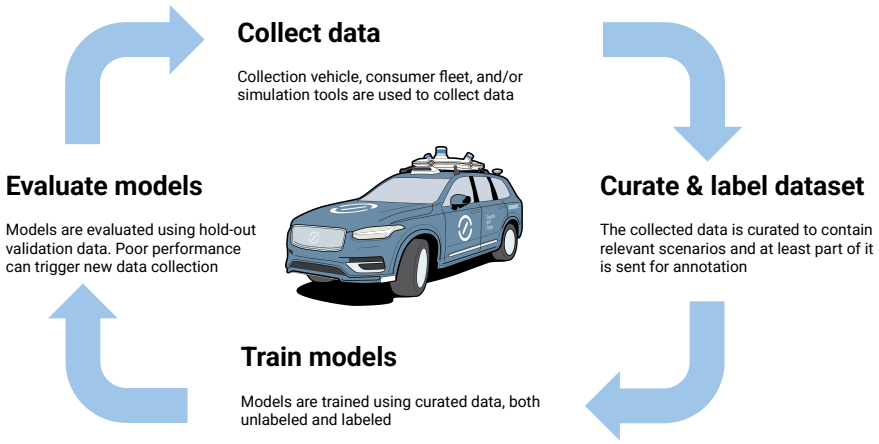

Figure 2.3: The iterative life cycle of an AD system. Data collection is usually governed by current system performance, *i.e.*, practitioners want more data from scenarios where the system struggles. Either this is handled directly at the data collection stage using techniques like targeted generation in simulation or triggers during collection, or during the curation stage. The curated data is optionally annotated, before being used for training the models of the stack. Models can also include algorithms not part of the stack that is running in the car but are used for development, *e.g.*, offline auto-annotation tools. Last, the trained models are evaluated, either in isolation, as a whole system, or both. This could be on already collected hold-out validation data, or by studying the driving behavior when driving in new scenarios. Collection vehicle designed by Jesper Moberg.

customers, or a combination thereof. By using professional drivers, developers have more direct control over what type of data is collected, making it a powerful debugging tool. However, due to the costs of maintaining a development fleet, the data quantity is usually limited in comparison to what is produced by an extensive consumer fleet. Notably, the development vehicles might have a sensor setup different from the consumer one, where additional high-resolution sensors are included as a reference.

After collection, or in some cases during collection, the collected data must be curated. Storing all collected data would soon become prohibitively expensive (a single car can produce several TBs of data in a day), annotating it even more so. Data should be selected such that the AD system can reach as high performance as possible while requiring as little human annotation as possible. What makes this challenging is that different subsystems might require different types of scenarios and knowing

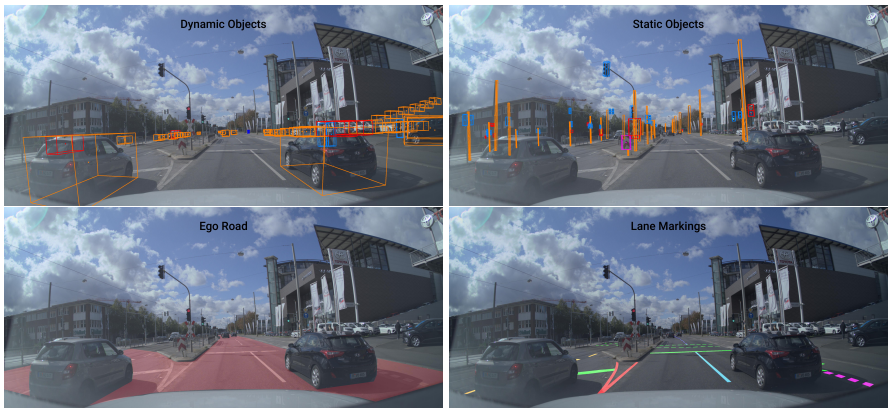

Figure 2.4: Examples of different types of annotations, from Zenseact Open Dataset [33].

beforehand if the raw data corresponds to something useful is not straightforward. Notably, this problem is less common in an academic setting compared to an industrial [43], [44]. Since academic labs rarely have access to a similar scale of data, related works on active learning [45] are done on a fairly small scale, with uncertain scaling properties. In addition, most benchmarking for different tasks is done on public datasets such as [33], [46]–[48] which have already been curated. As such, data curation at scale is mainly done in industry, and exact details are trade secrets.

After curation, the data are typically annotated, see Fig. 2.4 for examples of annotations. What type of annotations depends on which modules (as described in Sec. 2.1) need to be trained. Notably, for the end-to-end approach, annotation can in the simplest case mean the trajectory driven during data collection. However, for most tasks, human annotation is required, *i.e.*, a person looks at the data and provides ground truth for the specific task. This approach is expensive both in terms of cost and speed. According to [43], the cost of manually annotating objects in one hour of data is estimated to be approximately $150,000. Another option is to use automatic labeling. Often this entails using large, powerful offline methods. However, since these models are not robust enough to work in all possible settings, the hardest cases might still need human annotations even when using an auto-labeling tool [49]. Alternatively, for certain tasks, annotations can be created directly from the sensor data, *e.g.*, when predicting the future pose of the ego vehicle given the past and current images and lidar data.

Once the data and annotations are in place, the relevant systems can be trained. The

specifics of training neural networks using labels are explained closer in Sec. 3.1. But, on a high level, neural networks are adjusted such that their predictions are close to the annotations. This process requires a significant amount of computational power. As a reference, Tesla's occupancy network in 2022 required 100,000 GPU hours to finish the training, *i.e.*, if trained on a single GPU, it would take 11.4 years [50]. To accelerate iteration cycles, their training cluster comprises 10,000 GPUs, with an additional 4,000 dedicated to auto-labeling [50].

Testing the Performance

A key challenge in deploying safe AD is to ensure that it is safe *before* being unleashed on the roads. Much of this work starts already before any data is collected, where practitioners design the system, both hardware and software, to possess sufficient capabilities for the intended design domain. For instance, the sensors must have high enough resolution and range, the computing power must be powerful enough to process the data and run the algorithms, the system must have redundancy to handle unexpected failures, and the software must be designed such that it is possible to argue for its safety. However, even with these things in place, the task of proving the safety of a trained data-driven system remains challenging. Regulatory bodies can require data from millions of driven kilometers for certification of the final system. Since it is not feasible to conduct this type of testing for each update, practitioners use a range of methods, such as data replay, test tracks, monitored real-world driving, and shadow mode testing, to verify safety during the development.

Task-specific Evaluation The first step after training a system is generally to evaluate the performance using task-specific metrics on a hold-out validation dataset, *i.e.*, data that was not used for training. While this type of evaluation cannot be used by itself to determine if the performance is sufficient, its perk is that it quickly shows if the performance is poor. This enables developers to keep track of performance regression between different models. If the model performs poorly, potential reasons include inadequate model design or insufficient data. For the latter case, which is likely more common in matured systems, the easiest remedy is to collect more data. This ties back to the data collection process, where developers, after identifying what type of data is missing, request more data from the fleet of vehicles. Following a new training, one can verify that the issue has been resolved by again evaluating the performance for the validation data.

Open-Loop Simulation In open-loop simulation, collected data is replayed to the system which outputs a plan or low-level control actions. This is related to the task-specific evaluation, but instead, we can observe system-level response as information progresses between subsystems. For instance, we can measure the distance between planned actions and the ones that are considered optimal, *e.g.*, actions taken while the data were collected. However, since we rely on collected data that cannot be altered, the actions taken by the AD system are not truly actuated. As a result, we cannot know whether these errors would cascade during real-world driving. An example would be steering slightly too much away from the lane center and drifting toward the edge of the road until we reach a point that makes the system react in an unexpected way. Consequently, open-loop gives some indication of system behaviors, but does not capture the full complexity of real-world driving.

Closed-Loop Simulation In contrast, for closed-loop simulation, the actions taken by the AD system are actuated and its state is changed accordingly. Closed-loop simulation can be used both for testing parts of the AD system, and for testing it in its entirety. Which part of the systems that are evaluated has a significant impact on the efforts associated with creating a realistic simulation. For instance, for testing the motion planning of a modular AD system, the simulation should output the same type of data that is created by the upstream system, *e.g.*, local maps and bounding boxes over time. Simulation fidelity can then be related to concepts such as accurate behavior modeling of other actors and error modeling for perception and prediction modules. Such models can be combinations of advanced data-driven machine learning models, statistical models, or simple hand-crafted rules and heuristics. A common example is scenario testing, *e.g.*, other actors cutting in or a person running out on the road. Closed-loop simulation at this level is a popular approach, as it offers controllability and generally aligns better with real-world driving than open-loop simulation.

To include the perception modules in closed-loop simulation for full-system testing is a more challenging endeavor, as it requires turning abstract scenario descriptions into sensor-realistic environments. Often this is done using physics-based rendering, powered by game-engine-like simulators. There, procedural generation can create a variety of different static environments, which can then be coupled with a bank of manually created assets for vehicles and other traffic participants to act within this virtual world. While the quality of such systems has been rising steadily, questions remain regarding realism and scalable diversity, which in turn impacts the amount of trust which can be placed in results achieved in simulation. Further, creating realistic assets requires many hours of work from human artists, and producing high-fidelity

sensor data depicting the said assets further assumes deep domain knowledge and accurate sensor models.

Monitored Real-World Driving To overcome the poor sensor-realism in closed-loop testing, the most straightforward approach is to let the car drive itself in the real world and inspect the behavior. Many safety arguments are often supported by millions of autonomously driven kilometers. But, relying on this during development has multiple flaws. First, evaluating untested software requires trained safety drivers to monitor the vehicle and take over control if needed, making real-world testing costly and slow. Second, real-world driving is not reproducible, meaning that we cannot easily rerun a test when making any software updates. Finally, we have limited control over the environment, and as the performance of the system increases, encountering relevant scenarios becomes increasingly rare, further adding to the cost of real-world testing.

Test Track A prevalent approach to combine the sensor-realism of real data while having better control over the surroundings is to use test tracks, also known as proving grounds. These are areas closed off from the public where manufacturers can conduct targeted testing of their AD functionality. Often these have set up different types of roads and provide crash dummies of other vehicles and pedestrians. In this way, safety-critical scenarios can be tested without risking the safety of the driver or the surrounding environment. A common use case is safety certifications such as Euro NCAP [51], where functions such as automatic emergency braking are rated. A drawback of using test tracks is that they are also limited in diversity and setting up each scenario requires large amounts of time.

Shadow Mode Another alternative to monitored real-world driving is running the software in shadow mode, a concept closely related to open-loop simulation. Here, the systems are run on the live data, but the control signals are not actuated. For example, a human, or even a previously approved version of the software, might be driving the car, while a new software version is running in the background. Useful cases are when the system is disengaged either manually by the user or because the AD system is outside the current operational design domain. As a perk, shadow mode testing does not require trained safety drivers and can thus be run on more cars at the same time, *e.g.*, cars purchased and driven by regular consumers. However, as for the open-loop simulation, cascading errors cannot be evaluated using this approach.

2.3 Pain Points of Autonomous Driving

As highlighted above, developing AD systems is a complex endeavor. All these challenges can be derived from the fact that autonomous vehicles are safety-critical systems, where any error can have devastating consequences. Simultaneously, AD is expected to work in a wide range of scenarios. This combination imposes stringent requirements on all subsystems that influence the vehicle's decision-making processes. This section highlights some of the most significant challenges currently encountered in the development of autonomous driving systems, with a particular emphasis on the perception component. Note that this is not a complete discourse, ignoring aspects such as regulatory and ethical concerns, deployment and maintenance, or real-time requirements.

Uncertainty

Autonomous driving systems must adeptly manage uncertainty in various forms when operating in public environments. Firstly, while driving, we often lack complete information about our surroundings. For example, an AD system cannot determine whether a child is hidden behind a parked car and might suddenly run onto the road, or not. Even with visible objects, uncertainty persists. One reason is that the sensors used impose constraints on the information that can be gathered, such as estimating the 3D position and size of a car from a single image. Another reason is that the perception module might not have seen a specific object previously, but is still expected to make a reasonable guess of its properties. Therefore, perception algorithms must be capable of expressing their uncertainty, and this information must be formatted appropriately for downstream modules.

Secondly, certain tasks within an AD system are inherently ambiguous. Motion prediction, for instance, involves forecasting the future positions of other traffic participants. While we can determine the actual outcome by observing the scene as it unfolds, this is not the only possible scenario. The challenge lies in the fact that traffic participants, such as vehicles and pedestrians, can exhibit a wide range of behaviors influenced by numerous factors, including road conditions, traffic signals, and the actions of other drivers. Moreover, human behavior is inherently unpredictable, adding another layer of complexity to motion prediction. To address this, AD systems often rely on probabilistic models that can generate multiple potential future trajectories, each with an associated likelihood. These models must be robust enough to handle the inherent uncertainty and variability in human behavior, while also being computationally efficient to allow real-time decision-making. Furthermore, the system

must be capable of updating its predictions dynamically as new information becomes available, ensuring that the vehicle can respond appropriately to sudden changes in the environment.

Related to this, driving itself is an inherently ambiguous activity. There is seldom a single correct action to take; rather, there is a spectrum of actions that can achieve safe and comfortable driving. However, in real-world driving, only a single action can be executed and its outcome observed. This complexity complicates training and evaluation, as the decisions made by the AD system may differ from those of an expert human driver, yet still lead to an equally successful outcome.

Scalable Training

Training an AD system to be robust to a wide range of scenarios requires a massive amount of data. Moreover, the dataset should not only be large, but it should also be diverse. While managing and storing data are expensive, the requirement for data diversity is the driver for growing collection costs. As system performance increases, relevant scenarios become increasingly rare, further adding to the cost. Besides coming across relevant scenarios, accurately identifying them as such is also a challenge.

Another cost is the annotation of data. While automatic annotation is possible for some tasks, it is not always reliable. Examples might be when new objects are encountered (think, for instance, of the introduction of electric scooters), or when the system is not sure about a specific prediction. As such, a certain amount of data still needs to be annotated by humans.

Beyond being expensive to collect and annotate, real-world data itself comes with fundamental drawbacks. First, data from certain scenarios are close to impossible to collect in a safe manner, *e.g.*, a car crash. Such a scenario can be set up in a controlled environment, such as a test track, but this is not the same as driving in the real world. Unfortunately, it is these types of safety-critical scenarios that are the most important to collect. Second, collected data are non-interactive, *i.e.*, the system cannot interact with the environment by taking an action and observing the outcome. Many high-performance AI models are trained interactively using reinforcement learning directly in the environment they are supposed to operate in, for instance, AlphaGo [52], Agent57 [53], or, to some degree, ChatGPT [54]. Using this approach for AD is challenging, as the system must be able to drive itself in the real world.

Scalable Testing

One of the grand challenges in AD is to test and prove the safety of the system, and to do so in a scalable manner. Even if provided with a fully functional AD system, it remains challenging to assess its safety in a real-world setting. Safety drivers are required to oversee the vehicle and take over control if needed, making real-world testing both costly and slow. Moreover, any updates to the system likely require a certain degree of re-testing, further complicating the process. Thus, practitioners are constantly searching for new tools that enable scalable testing.

As highlighted earlier, simulation is one key tool for testing AD systems. By increasing the amount of compute power, simulation enables running a larger set of evaluation scenarios in parallel. However, current simulators are not yet able to fully replicate the complexity of the real world. Further, while running the simulations is relatively cheap, creating realistic simulations is expensive. Both asset creation and implementing physics-based simulation are labor-intensive tasks. Therefore, large-scale closed-loop simulation is often associated with only object-level simulation, where the environment is represented by different types of abstractions [22], [55], [56]. Running the simulation on a sensor level with sufficient realism and diversity remains an open challenge thus far.

CHAPTER 3

Perceiving the Surroundings

Perception is the process of acquiring information about the environment. For autonomous driving, it is one of the core components, enabling the vehicle to understand its surroundings and make informed decisions. For successful AD, the vehicle must know what type of other traffic participants are present, where they are, and how they are behaving. The vehicle must also know what the road looks like, and what the traffic signs and signals say.

This chapter introduces topics needed to understand how perception systems work. We use object detection as a concrete example, but the concepts apply to other perception tasks as well. The vast majority of perception systems are based on deep learning models trained on large amounts of labeled data, known as supervised learning. Thus, this chapter begins by covering the topics of supervised learning. However, no perception system can be perfect and it is important to understand the limitations of the system. Thus, we also introduce uncertainty estimation, which is a key component of safe AD perception.

3.1 Learning from Human Supervision

Most perception systems consist of deep neural networks that have been trained on large amounts of labeled data. One can think of these systems as a function f_θ parameterized by θ that maps an input x, such as images, to an output y, such as the position and velocity of other traffic participants. The goal of the training process is to learn the parameters θ such that the function f_θ can be used to make accurate predictions on new unseen data. A simple example would be to learn the slope m and intercept b of a line $y = mx + b$, making m and b the parameters. In practice, the function f_θ contains millions, or even billions, of parameters.

To find good parameters using supervised learning, we need a training dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ of N paired examples, where $X = \{x_i\}_{i=1}^N$ is an input and $Y = \{y_i\}_{i=1}^N$ is the corresponding ground truth output. On a high level, the training process consists of showing the network an input example x_i and adjusting the parameters θ to minimize the difference between the network's prediction $\hat{y}_i = f_\theta(x_i)$ and the ground truth, which is also known as a label, y_i. The difference is measured using a loss function $L_\theta(y_i, \hat{y}_i)$, which could be a simple mean squared error or a more complex function that also contains learnable parameters or has some other dependence on θ. Formally, the training process can be formulated as minimizing the following objective function:

$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} L_\theta(y_i, f_\theta(x_i)). \tag{3.1}$$

This is also known as empirical risk minimization, as the aim is to minimize the expected loss over the training data. Note that in most real-world applications, we will not find a θ^* that is globally optimal for (3.1), but rather find some local optimum or saddle points.

To search for θ^*, most deep learning frameworks rely on gradient-based optimization, and hence require the neural network f_θ to be differentiable with respect to the parameters θ. Given an initial guess θ_0, the parameters are iteratively updated until convergence. One of the most popular optimization methods is gradient descent, which updates the parameters in a direction that reduces the loss function,

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta \sum_i L_\theta(y_i, f_\theta(x_i)), \tag{3.2}$$

where η is the learning rate or step size. Since the number of training samples N typically is large, the true gradient is often approximated using a minibatch of m

samples, known as mini-batch gradient descent. In the extreme case, where $m = 1$, the method is known as stochastic gradient descent (SGD). Beyond (stochastic) gradient descent, there are many other popular gradient-based optimization methods, such as Adam [57] and RMSprop [58]. However, the choice of optimization method is not the focus of this thesis, and we refer the reader to [2] for more information.

The supervised learning framework is one of the most popular ways to train deep neural networks. Compared to traditional algorithm design, where practitioners need to know *how* to solve a task, supervised learning only requires samples of *what* the task is. With enough high-quality data, a sufficiently expressive model, and a good optimization method, the model can learn to make accurate predictions on new, unseen data. Given the ever-increasing availability of computational resources, the bottleneck of this process is often the amount of data available. Collecting the ground-truth labels generally requires a lot of human annotation, which is both time-consuming and expensive. Further, the annotators must have a good understanding of the task at hand to be able to provide high-quality labels. While annotation software can aid the annotators to speed up the process and improve the quality, it is still a manual process that requires a lot of time and effort.

3.2 Modeling Uncertainty

When describing the ultimate goal of deep neural networks, we often do so in terms of performance. Naturally, we want them to perform as well as possible in as many situations as possible. In the best of worlds, a deep neural network would never err. However, in reality, we are often constrained by things such as the amount of available data for training our networks, or, in terms of computational power, limiting the expressiveness of the models. As the perfect neural network remains a lucid dream, it becomes important to understand the limitations of our trained models to understand what they do not understand. This is especially valuable for safety-critical applications such as AD, where, for instance, uncertainty about the location of an oncoming vehicle should reflect upon the action taken. To formalize the concept of reasoning about the limitations of our model, we rely on uncertainty modeling.

When training and using a deep neural network, there are multiple sources of uncertainty. First, the input data can contain noise. For instance, a water drop on the camera lens introduces severe distortions. Second, the labels used for supervision can be noisy. Sources can be imperfections, or even errors, due to human errors in the annotation process, or be due to inherent uncertainty in the task. Third, there can be uncertainty in the model itself. Due to limited data, we cannot be certain

about the model parameters. The first two sources of uncertainty are often referred to as aleatoric uncertainty, whereas the latter is known as epistemic uncertainty [59]. Aleatoric uncertainty is inherent to the data, and cannot be reduced by increasing the amount of training data. In contrast, epistemic uncertainty can be reduced by increasing the amount of training data and arises from the model's lack of knowledge about the data.

A common approach to capture epistemic uncertainty for neural networks is to use a Bayesian approach, where the parameters are modeled as random variables [60], [61]. Specifically, we put a prior distribution over the parameters θ, such as a Gaussian distribution $\theta \sim \mathcal{N}(\mu, I)$. Then, given the data X and labels Y, the aim is to find the posterior distribution over the parameters $p(\theta \mid X, Y)$. This way, we can find all plausible values for the parameters given the current data. While Bayesian neural networks are easy to formulate, it is challenging to perform inference in them, and hence most methods rely on different approximations [62], [63] or are restricted to simple models.

To model aleatoric uncertainty, practitioners often apply a distribution over the output of the network [59]. For instance, for a regression task, *i.e.*, predicting a continuous value such as the location of a car, the output of the network can be modeled to be corrupted by Gaussian noise. Aleatoric uncertainty can further be subdivided into homoscedastic and heteroscedastic. In the former, the corrupting noise is assumed constant and independent of the input. For the latter, we are instead interested in the variance of this distribution for different inputs and try to learn these from the data.

To give an example, consider a regression task where the goal is to predict the location of a traffic participant. In the homoscedastic case, if we model the output to be corrupted by Gaussian noise we treat the output $\hat{y}$ as the expected mean of the distribution. If our loss function is the negative log-likelihood, we get

$$L(y, \hat{y}(x)) = -\log\left(\mathcal{N}(y; \hat{y}(x), \sigma^2)\right) \tag{3.3}$$

$$= \frac{1}{2\sigma^2}(y - \hat{y}(x))^2 + \frac{1}{2}\log(2\pi\sigma^2), \tag{3.4}$$

where σ^2 is the variance of the noise. If the noise σ^2 is constant, we can treat it as a constant factor, and hence the loss function is equivalent to the L2 loss

$$L(y, \hat{y}(x)) = (y - \hat{y}(x))^2. \tag{3.5}$$

For the heteroscedastic case, we instead model the variance of the noise to be depen-

dent on the input, and hence the loss function becomes

$$L(y, \hat{y}(x)) = \frac{1}{2\sigma(x)^2}(y - \hat{y}(x))^2 + \frac{1}{2}\log(2\pi\sigma(x)^2), \tag{3.6}$$

where $\sigma^2(x)$ is the variance of the noise at input x. While a larger variance will decrease the first term, it is regularized by the second term.

The negative log-likelihood is an example of a proper scoring rule [64]. A scoring rule is a function that maps the predicted probability of an event and the observed outcome (label) to a scalar score. A scoring rule is proper, if the expected score is minimized only when the predicted probability matches the true probability, *i.e.*, if the learned model and the underlying distribution match. This makes them useful both for training and for evaluating the performance of a model.

3.3 Object Detection

Object detection is a common task in perception systems and is often used as a building block for other tasks, such as tracking. Formally, the task is to, given some sensor data, predict the set of objects present in the scene $\mathcal{O} = \{(c_i, l_i)\}_{i=1}^{N}$, where c_i is the class of the object and l_i is the location of the object in the scene. In practice, most object detection models also provide a confidence score for each prediction, which can be used to filter out low-confidence predictions. However, for the ground truth, the class is a discrete value, such as "car", and is typically part of a predefined set of classes $c_i \in \mathcal{C}$. For image and video data, the location is commonly represented using a 2D bounding box, i.e., a rectangle that bounds the object in the image. Other representations, such as 3D bounding boxes or segmentation masks, are possible, and which to use depends on the task at hand and downstream applications.

As for other deep learning tasks, the progress in object detection has often been measured using public benchmarks. One of the first major benchmarks for 2D object detection was the PASCAL Visual Object Classes (VOC) challenge [65]. While held for the first time in 2005, the 2007 version increased the number of classes and provided a standardized evaluation metric, paving the way for rapid progress in the field. Since then, many other object detection benchmarks have been proposed, such as the COCO dataset [66], KITTI [67], or Open Images [68], introducing larger datasets, new sensors, or a wider range of objects.

Detection Models

A key challenge with the object detection task is that the number of objects in the scene is usually unknown beforehand. To address this, many early works relied on predicting a large, fixed number of bounding boxes. Common examples include the YOLO [69] and SSD [70] models, which are so-called one-stage detectors. Both of these models divide the image into a grid of cells, and predict a set of bounding boxes and class probabilities for each cell. However, the YOLO model does so directly, while SSD predicts offsets from predefined proposal boxes. In contrast, two-stage detectors such as the R-CNN suite of models [71]–[73] first generate a set of proposal boxes, and then run a second stage to classify each box and refine the location of the bounding boxes. Nonetheless, both types of detectors result in many overlapping bounding boxes. At inference, the predictions are thresholded based on their confidence score and filtered using non-maximum suppression (NMS) to remove overlapping duplicates.

While non-maximum suppression is a simple and effective way to remove duplicates, it is a hand-crafted method based on greedy clustering, raising the question about its optimality. In line with the reoccurring trend in deep learning of aiming to learn the function end-to-end, multiple works have tried to either learn NMS [74] or design the model to predict the set of objects directly [75]. However, not until the introduction of the Transformer architecture [76] did direct prediction of the set of objects become a competitive approach. The Detection Transformer (DETR) [77] still predicts a fixed number of objects, however, this number is much smaller than one- or two-stage detectors. Further, the predictions are supervised to contain no duplicates by assigning at most one prediction to each ground-truth object and assigning the remaining predictions to the background class. The assignment is done optimally with respect to a cost function, ultimately enabling the set of objects to be predicted directly. Since its original formulation, DETR has given rise to a suite of set-based detectors through various improvements such as multiscale processing or improved supervision [78]–[80].

Evaluation Metrics

The most common evaluation metric for object detection is the mean Average Precision (mAP). The metric relies on precision and recall curves, where precision and recall are defined as

$$P(c) = \frac{\mathrm{TP}(c)}{\mathrm{TP}(c) + \mathrm{FP}(c)}, \tag{3.7}$$

and

$$R(c) = \frac{\text{TP}(c)}{\text{TP}(c) + \text{FN}(c)}, \tag{3.8}$$

where TP and FP are the number of true positives and false positives, respectively, and FN is the number of false negatives. Intuitively, precision measures quality, *i.e.*, how many of the predictions are correct, while recall measures quantity, *i.e.*, how many of the labels have correctly been identified. The average precision is computed as the area under the precision-recall curve, *i.e.*, the precision we can expect at a given recall. The mean average precision is then the mean of the average precision over all classes.

To define a true positive, mAP, in the context of 2D object detection, relies on the Intersection over Union (IoU) metric. As the name indicates, it is the area of the intersection of the predicted bounding box and the ground-truth bounding box divided by the area of their union. A prediction is considered a true positive if its IoU is greater than a predefined threshold, typically 0.5, and is of the same class as the ground truth object. Predictions are assigned to the ground truth object greedily, *i.e.*, the prediction with the highest confidence is assigned first. Notably, some benchmarks also use multiple IoU thresholds and average the multiple mAPs. Additional breakdowns are also common, such as by object size.

One of the perks of mAP is that it does not require any confidence threshold, as the precision and recall curves are invariant to the scale of the confidence scores. The only thing of importance is that the detections are ranked correctly by the detector, *i.e.*, the correct detections have higher confidence scores than the incorrect ones. However, this, in combination with treating each class independently, also means that the metric can behave in unexpected ways, as highlighted in [81]. The confidence scores for one class can all be within the range [0.1, 0.11], while the scores for another class can be all within the range [0.9, 0.91], and the model still achieves a high mAP. Thus, for deployment, relying on a single mAP score can be misleading, and models typically need to be calibrated or use class-specific thresholds.

In the 3D object detection community, mAP is also used, but often evaluated for two types of IoU metrics. The first is the 3D IoU, which is the volume of the intersection of the predicted bounding box and the ground truth bounding box divided by the volume of their union. The second is the bird's-eye view (BEV) IoU, which is the 2D IoU from a top-down view of the bounding boxes. While 3D IoU might seem more interesting for the perception task, BEV mAP is popular as it is more lenient toward vertical misalignments than 3D mAP and it aligns more closely to planning and control communities. For many such benchmarks, 3D information is disregarded, and hence BEV mAP is a more suitable metric. Other distance measures can also be

used for defining true positives, such as centerpoint distance. Beyond mAP, several AD datasets [46], [47] also use custom metrics to isolate different aspects of the task. This includes translational, scaling, and orientation errors.

Probabilistic Object Detection

The task of probabilistic object detection, as described in [4], is to detect objects while also accurately quantifying the spatial and semantic uncertainty of the predictions. The authors of [4] propose to use a categorical distribution over all class labels and represent each bounding box as separate Gaussian distributions for the corners of the bounding box. For images, as explored in [4], the two corners are described using two 2D Gaussian distributions. To evaluate the performance of the model, the authors propose using the Probability-based Detection Quality (PDQ) metric. The metric, however, is biased toward predictions of a smaller extent [5], and can therefore not be used for a fair comparison between models.

The authors of [5] instead propose to train and evaluate the predictive uncertainty of the model using proper scoring rules. In the paper, three different types of scoring rules are considered for learning the uncertainty of the bounding boxes, namely negative log-likelihood, energy score, and direct moment matching. For the classification, cross-entropy loss is used. This is applied to three different types of detectors, a one-stage detector, a two-stage detector, and a set-based detector. For evaluation, the authors use multiple scoring rules, such as the Brier score, the energy score, and the negative log-likelihood. While their evaluation is extensive and makes a good case for using multiple scoring rules to better understand the model behavior, it also overlooks a critical aspect of the evaluation. Namely, the assignment between the prediction and ground truth is done without any regard to the localization uncertainty, and based on ad-hoc rules.

As we argue in Paper A [6], the assignment will color the evaluation and any conclusions drawn from it. Consider Fig. 3.1, where we have one ground-truth bounding box in blue, and two predictions in red and green. The red prediction has a larger IoU with the ground truth and hence is considered a true positive by mAP and methods in [5]. However, if considering the bounding box uncertainty, the green prediction is a more probable match to the ground truth. Thus, the assignment problem should be rigorously modeled for any evaluation.

Random Finite Sets To properly model the uncertainty in object detection, we would like to have a distribution that captures all aspects of the task. For this, we need

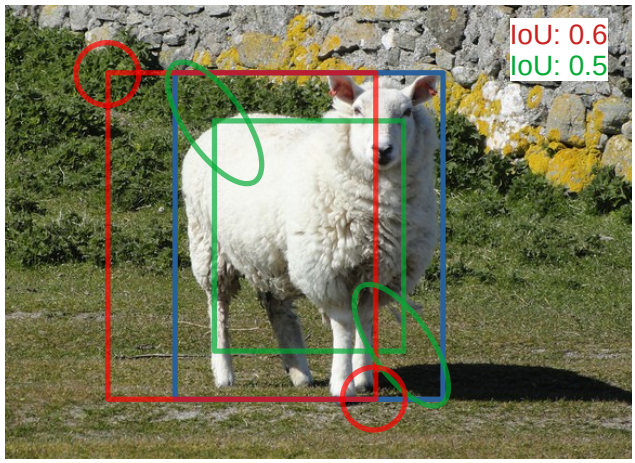

Figure 3.1: Predicted mean and covariance for two detections (red and green). mAP and method proposed in [5] prefer the red prediction with larger IoU. Our method considers uncertainties and multiple assignments and finds the green prediction a more probable match to the blue ground truth.

a distribution over the variable we want to predict. As can be seen above, existing work often considers each object as a separate variable and disregards their interaction among them. However, looking back at the task description of object detection, the goal is to predict a *set* of objects. As such, the natural choice of distribution to use for modeling object detection uncertainty should be a distribution over sets. Fortunately, the random finite set (RFS) formalism provides a framework for exactly this [7].

Random finite sets are probability density functions whose outcomes are sets. They view the set as a single random variable and enable us to capture uncertainties both in terms of cardinality and for describing the individual elements of the set. Further, when evaluating the likelihood of a given set, *e.g.*, we have predicted the distribution $f(\mathbb{Y}|X)$ and are interested in how well it describes the observed set $\mathbb{Y}$, RFSs explicitly model the assignment problem and consider all possible assignments and their individual likelihoods. Random finite sets have long been used in the model-based multi-object tracking community, as they provide theoretically grounded tools to reason about the existence and properties of multiple objects [82], [83]. Despite their appealing properties, they have not received any attention from the object detection community until recently [6].

To provide some intuition, we introduce some of the commonly used RFSs for

describing a set of objects. One of the simplest RFSs is the Bernoulli RFS which can be used to model a single object. Its density is

$$f_{\mathrm{B}}(\mathbb{Y}) = \begin{cases} 1-r & \text{if } \mathbb{Y} = \emptyset, \\ rp(y) & \text{if } \mathbb{Y} = \{y\}, \\ 0 & \text{if } |\mathbb{Y}| > 1, \end{cases} \tag{3.9}$$

where r is the probability of existence and $p(y)$ is the single-object density, *i.e.*, the density describing the distribution over the object itself, conditioned on its existence. For instance, $p(y)$ could consist of a class distribution and a distribution over the object's location and extent, similar to what is used in existing work. With probability $1-r$, the set is instead empty and the object does not exist. Further, the Bernoulli RFS assigns no density to the event that $|\mathbb{Y}| > 1$, *i.e.*, it can only model a single object.

To model multiple objects, we can take the union of multiple Bernoulli RFSs and combine them into a multi-Bernoulli (MB) RFS. Formally, let $\mathbb{Y}_1, \ldots, \mathbb{Y}_m$ be m independent Bernoulli RFSs with densities $f_{\mathrm{B}_1}(\mathbb{Y}_1), \ldots, f_{\mathrm{B}_m}(\mathbb{Y}_m)$, existence probabilities $r_1, \ldots, r_m$, and single-object densities $p_1(y), \ldots, p_m(y)$. Then $\mathbb{Y} = \cup_{i=1}^{m} \mathbb{Y}_i$ is an MB RFS with multi-object density

$$f_{\mathrm{MB}}(\mathbb{Y}) = \sum_{\uplus_{i=1}^{m} \mathbb{Y}_i = \mathbb{Y}} \prod_{j=1}^{m} f_{\mathrm{B}_j}(\mathbb{Y}_j), \tag{3.10}$$

where $\sum_{\uplus_{i=1}^{m} \mathbb{Y}_i = \mathbb{Y}}$ denotes the sum over all disjoint sets whose union is $\mathbb{Y}$. In other words, when evaluating the multi-object density $f_{\mathrm{MB}}(\mathbb{Y})$ of a set $\mathbb{Y}$ we sum the multi-object densities of all possible assignments between elements in $\mathbb{Y}$ and Bernoulli components in f_{MB}. This way, RFS explicitly reasons over possible assignments between predicted densities and observed set elements.

The Poisson Point Process (PPP) is an RFS that unlike the MB RFS has no upper limit on its cardinality. Instead, the cardinality follows a Poisson distribution, assigning a non-zero probability to any cardinality. To achieve this, the PPP RFS relies on an intensity function $\lambda(y)$, which is similar to a density function, with the exception that it does not have to sum to one. The resulting multi-object density is

$$f_{\mathrm{PPP}}(\mathbb{Y}) = \exp\left(-\bar{\lambda}\right) \prod_{y \in \mathbb{Y}} \lambda(y), \tag{3.11}$$

where $\bar{\lambda} = \int \lambda(x')dx'$ is the expected cardinality of the set. In the multi-object tracking community, the PPP RFS is often used to model objects that have not yet been detected.

We can combine the PPP and MB RFS into a Poisson multi-Bernoulli PMB RFS by taking the union of a PPP and an MB RFS. The resulting multi-object density is

$$f_{\text{PMB}}(\mathbb{Y}) = \sum_{\mathbb{Y}^{\text{U}} \uplus \mathbb{Y}^{\text{D}} = \mathbb{Y}} f_{\text{PPP}}(\mathbb{Y}^{\text{U}}) f_{\text{MB}}(\mathbb{Y}^{\text{D}}), \tag{3.12}$$

where $\mathbb{Y}^{\text{U}} \uplus \mathbb{Y}^{\text{D}}$ refers to summing over all possible ways of partitioning $\mathbb{Y}$ into two disjoint sets.

One of the main issues for using RFSs in an object detection context is that evaluating the likelihood requires considering *all* possible assignments. This is often computationally expensive when the number of objects is in the hundreds or even thousands. The authors of [84] propose to instead approximate the likelihood by only considering the K most likely assignments. These can be found efficiently by using, for instance, Murty's algorithm [85]. This enables using the negative log-likelihood to evaluate a predicted RFS density in an object detection context, as we show in Paper A.

CHAPTER 4

Finding the Right Data

This chapter introduces different concepts that are commonly used to reduce the need for expensive human annotations. First, we cover the two paradigms self-supervised learning (SSL) and weakly supervised learning (WSL). In self-supervised learning, one aims to learn meaningful representations directly from the data. Weakly supervised learning instead aims to learn from cheaper labels or use more abstract supervision. Both of these methods enable models to be trained at scales far exceeding what is typically feasible for supervised learning. This way, the models can be exposed to a wider range of data and scenarios, making SSL and WSL attractive alternatives for AD practitioners. We introduce both these learning concepts closer below, with a special emphasis on learning from language for the weak supervision setting. Further, we cover auto-labeling, a technique that instead of avoiding labels emphasizes ways to efficiently create labels automatically.

4.1 Self-Supervised Learning

Self-supervised learning can be defined in multiple ways [86], but broadly speaking, it refers to any method that learns representations without using human-annotated labels. Instead of human labels, the model is trained on a pretext task, *i.e.*, a task that

is related to the ultimate goal, but is easier to obtain labels. Examples of pretext tasks include predicting a masked-out word in a sentence [87] or part of an image [15]. Following the pre-training, the model is then fine-tuned towards the ultimate goal. What makes SSL-trained models particularly useful, is that they can be fine-tuned to a range of tasks and domains, and often reach competitive performance with limited amount of annotated data. Due to their wide adaptability, such models are commonly referred to as foundation models.

The approach of training a model self-supervised on a huge unlabeled dataset, followed by task-specific fine-tuning was first shown to be effective at scale in the natural language processing (NLP) domain [87]. It remains one of the success factors of large language models (LLMs), as it enables training them at unprecedented scales. Similarly, this approach has been successful in other domains as well, such as vision. Below we introduce the most popular SSL methods used in the image domain, but note that they are applicable to other perceptual domains as well.

Contrastive Learning One of the most popular SSL methods is contrastive learning [16], [88], [89]. The idea of contrastive learning is to map high-dimensional data, such as images, to a low-dimensional latent space, where the latent space is ordered such that semantically similar data points are close to each other. For training, the model is fed positive pairs that should attract and negative examples that should repel. Positive pairs are created by augmenting the same image in two different ways, as this ensures that there is similar semantic content in both images. Negative pairs are constructed by pairing these augmented views with other images in the dataset, which are assumed to be dissimilar. For instance, from a mini-batch of N images, one can create N positive pairs, resulting in $2N$ total data points. Further, each data point can be paired with each of the $2(N-1)$ negative examples. The loss function for a positive pair (z_i, z_j) can then be defined [89] as:

$$l_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}(k \neq i) \exp(\text{sim}(z_i, z_k)/\tau)} \tag{4.1}$$

where $\text{sim}(z_i, z_j)$ is the similarity between samples z_i and z_j, and τ is a temperature parameter. The denominator is the sum of similarities between z_i and all negative examples for z_i. Often, the similarity function is the cosine similarity, but other similarity functions can also be used. The loss function encourages the latent encoding of the positive pair to be closer to each other than to any of the negative pairs.

While conceptually simple, contrastive learning requires careful design to learn meaningful representations, and the field has undergone a number of developments.

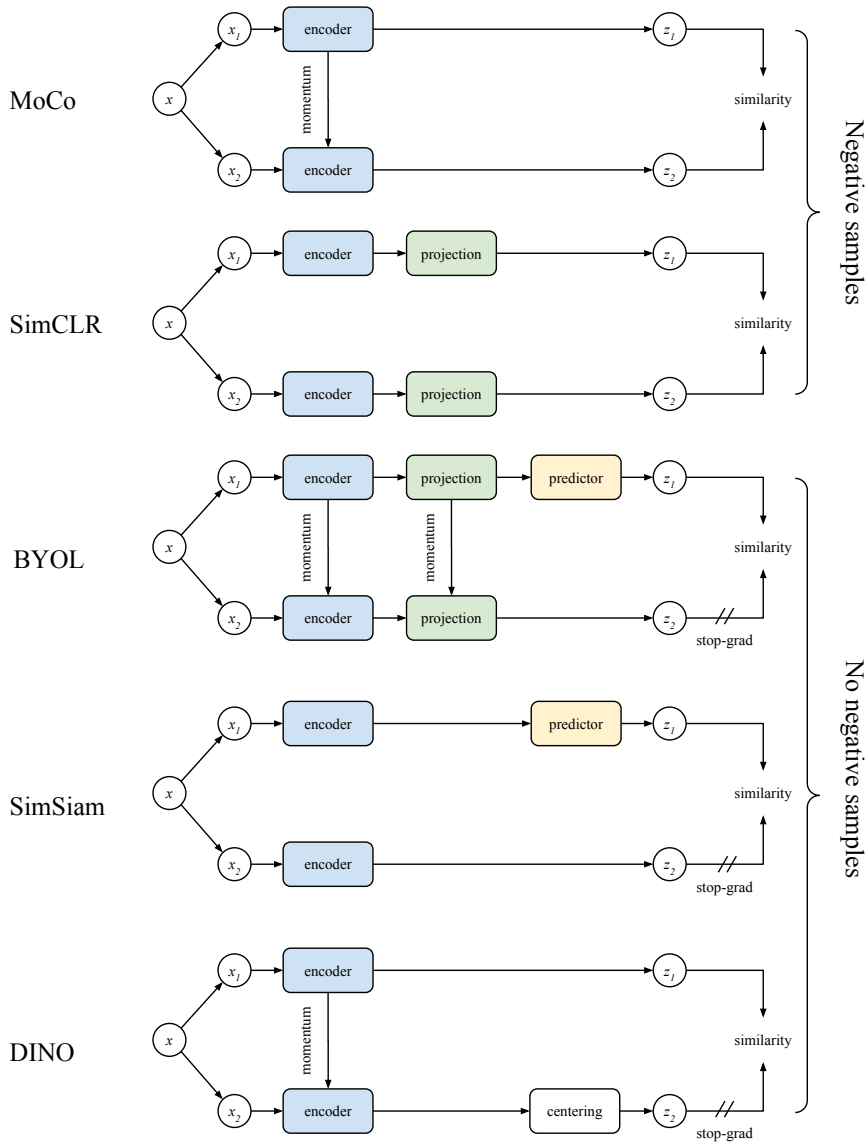

Figure 4.1: Overview of different contrastive learning (needs negative samples) and self-distillation (no need for negative samples) methods.

MoCo [16] proposes to use two models, a query model and a key model. The query model sees one version of the augmented data and is the one being optimized. The key model sees a different version of the augmented data and is a moving average of the query model. SimCLR [89] simplifies contrastive learning by using the same model for encoding both augmented data. Instead, they introduce a small MLP after the encoder, which maps the encoded data to a different latent space before applying the loss function. Note that this MLP is discarded after training. Further, the authors study the effect of different augmentations and note that SSL benefits from stronger augmentations compared to what is used in supervised learning, and from larger batch sizes and longer training times. See Fig. 4.1 for a visual comparison.

Self-Distillation Self-distillation is a self-supervised learning method that shares many similarities with contrastive learning, but steps away from the need for negative examples. One of the first works to demonstrate the perks of self-distillation at scale was Bootstrap Your Own Latent (BYOL) [90]. Similar to MoCo, BYOL uses two models, one that is being optimized (the student) and one that is a moving average of the optimized model (the teacher). The model itself consists of an encoder and a mapping MLP, similar to SimCLR. Further, each version of the model is fed an augmented version of the same data, where the student's prediction should be close to the teacher's representation. However, compared to SimCLR, BYOL introduces an additional MLP to the student model (not used for the teacher), which is tasked to predict the output of the teacher.

Since BYOL does not rely on negative examples, one would expect it to be prone to collapse to a single point, *i.e.*, all inputs are mapped to the same representation. The authors hypothesize that the combination of the additional MLP and the slow-moving average of the model encourages the encoding of more and more information, rather than collapsing. Follow-up works such as SimSiam [91] investigate this closer and show that the main reason for avoiding collapse is the use of a stop-gradient operation, *i.e.*, the teacher and student can be identical models as long as only the student is updated during gradient descent.

A similar approach to BYOL is DINO, knowledge **di**stillation with **no** labels [14]. DINO uses a student-teacher setup, with the teacher being a moving average of the student, and augments the data to create input pairs. Compared to BYOL, the architecture is simplified, with all extra MLPs removed, and the loss function is different. Instead, the latents of the teacher model are centered using a moving average of teacher latents and sharpened. These two operations are sufficient to stop the model from collapsing. Further, rather than using cosine similarity, DINO uses the

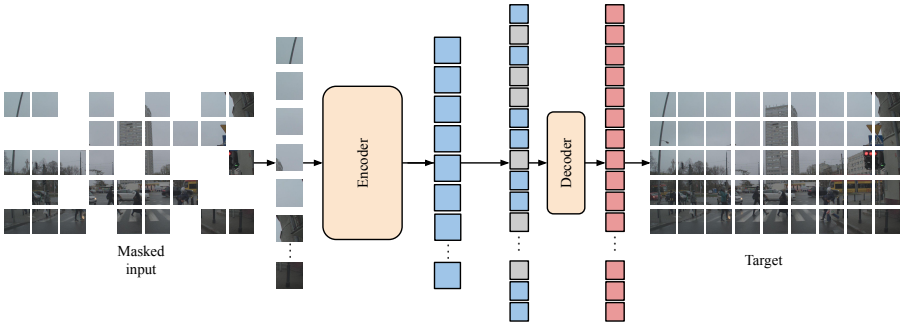

Figure 4.2: Masked autoencoder method overview. The encoder embeds masked inputs. From this context, a lightweight decoder is tasked to reconstruct the masked parts.

cross-entropy loss to compare the student and teacher latents, inspired by SwAV [92].

Besides providing models with strong representational capabilities, DINO also sheds some light on why their teacher-student setup is so successful. Their experiments show that the teacher consistently outperforms the student during training. The proposed interpretation is that the moving averaging is a form of Polyak-Ruppert averaging with exponential decay. This is a popular technique to simulate ensembling multiple models, and thus improve final performance. Thus, using momentum can be seen as constantly building an ensembled model during training.

Masked Autoencoders While contrastive learning and self-distillation both result in models with powerful representations, they come with practical issues. For contrastive learning, the requirement to use large batch sizes with many negative examples hinders applications with high-dimensional input. The self-distillation approach does not suffer from this, but instead requires keeping two copies of model weights in memory, limiting the size of model that can be trained. Masked autoencoders (MAE) [15] provide an SSL method that can handle both high-dimensional data and train models of larger size.

The task in MAE is to, given incomplete input data, predict the missing data, see Fig. 4.2. Specifically, a small fraction of the image ($\sim 30\%$) is processed by an encoder and the resulting embeddings are provided as context to a lightweight decoder, which attempts to predict the content of the masked parts. This way, the encoder is forced to learn to embed enough context from incomplete information. For example, the decoder should be able to predict what a car looks like, even if only part of the trunk is visible.

The key enablement of MAE is the use of Vision Transformers (ViTs) [93], since they, compared to CNNs, are more computationally efficient for sparse inputs. The extensive masking drastically reduces the amount of computation required to train the encoder, leading to faster training times compared to other SSL methods like contrastive and self-distillation. Likewise, at a similar computing budget, MAE enables the training of larger models.

4.2 Weakly Supervised Learning

Weakly supervised learning is a branch of machine learning where the supervision signal is either incomplete, inexact, or inaccurate [94], [95]. For incomplete supervision, only a subset of the training data has associated labels. This is also known as semi-supervised learning. The inexact supervision refers to coarse-grained labels, *e.g.*, only indicating whether an object is present or not, rather than its exact location in an image. Inaccurate labels refer to the labels not necessarily being ground truth and correct. Compared to the fully supervised setting, these types of supervision signals are usually less expensive to acquire and can therefore be applied at a greater scale. Below we examine WSL through two weakly supervised settings common in the autonomous driving context. The first example shows how web-scale text-image pairs can be used as inexact and inaccurate supervision to learn powerful language-vision representations. As a second example, we describe auto labeling, *i.e.*, different ways of automating the creation of labels for inaccurate supervision.

Learning from Language

Contrastive Language–Image Pre-training (CLIP) [18] is a method for learning powerful visual models from natural language supervision. CLIP is trained using web-scale image-text pairs, *i.e.* images and their captions. The model consists of a text encoder and an image encoder, where each encoder maps their respective modality to a single vector. The encoders are trained jointly using a contrastive loss, which encourages latent representations of pairs to attract each other while repelling representations of other pairs.

Conceptually, the training of CLIP has commonalities with self-supervised contrastive learning. The difference is that CLIP uses modality-specific encoders, and relies on pairs being aligned when collected, *i.e.*, that the captions and images are sensible matches, rather than creating pairs through augmented views. Thus, instead of viewing CLIP as an SSL method, we consider the image captions to be a form of

inexact supervision.

Using CLIP, one can compare the similarity between an image and arbitrary text descriptions, *e.g.*, "a photo of a cat" or "a photo of a dog". By selecting the most similar text description, one can then use the model to perform zero-shot classification. CLIP's zero-shot classification capabilities are on par with SSL models fine-tuned in a few-shot setting, *i.e.*, using a small amount of labeled data. Further, using CLIP in a few-shot setting, performance can be improved even further.

Vice versa, one can compare multiple images to a single text description and select the image that is most similar to the text description. This task, known as retrieval, is highly relevant for AD, where large amounts of unlabeled data are available, and practitioners are often interested in finding certain types of images. This enables one to quickly find rare cases, *e.g.*, data with animals on the road or data collected in severe weather conditions. Further, one can use this to retrieve data points that are similar to a given image that an object detector is struggling with.

Auto-Labeling

A different way to reduce the need for human annotations is to use automatically generated labels for the given task. There are many ways of building a pipeline for auto labeling, but generally such, pipelines contain large models that do not have real-time requirements. This can mean using huge networks or ensembles of models and letting these use tricks such as hindsight. For example, a large ensemble of models might accurately classify nearby traffic signs. For video data, these predictions can be turned into accurate labels when the sign is far away by tracking its location backward in time.

A popular application for automatic labeling is offline or offboard 3D object detection [9], [10]. Automatically annotating objects in 3D enables more scalable data generation for modules such as motion prediction and planning, which often rely on object-level data in a modular stack. Most offline 3D object detectors follow the recipe of using a large detector, tracking the detections forward and backward in time, and refining the predictions based on the resulting tracks. In particular, recent methods [10] claim to outperform human performance using this approach. Nonetheless, automatic labeling in general has some fundamental drawbacks.

First, training any large and powerful network will require a significant amount of labeling. As highlighted above, self-supervised learning can alleviate some of this, but for the model to reliably perform a task as intended labels will be needed. Second, it remains challenging to automatically verify the generated labels. To be completely

certain that no erroneous labels are created, some human-in-the-loop verification is likely needed.

4.3 Applications to Autonomous Driving

Much of the driving force behind the development of SSL methods has come from the NLP and vision domains. Both of these domains enjoy the benefits of readily available large-scale data, with text, images, and video being abundant. However, autonomous driving is a multi-modal domain, where multiple types of sensors are used to perceive the surroundings. This type of multi-modal data is often more scarce, at least publicly. Further, while many SSL methods are trained on data collected from different types of cameras (in the vision context) or different authors (in the NLP context), the diversity in certain AD sensors, such as lidars, is rather limited with only a handful of manufacturers and models. Similarly, the data itself can be considered to be more homogenous than web-scale datasets used to train large language or vision model. Thus, it is natural to ask whether SSL methods can be applied in the autonomous driving setting and what gains they bring.

Lidar-based models have been trained using a range of SSL methods. Multiple works have applied contrastive learning to learn representations from outdoor point clouds [96]–[100]. Compared to image-based models, these generally use more complex matching strategies, where subregions of the point clouds are matched across different views. For instance, SegContrast [97] uses pre-processing to find segments in the point clouds, and then contrasts these segments against each other.

The masked autoencoder framework has also proven a successful pretext task for point cloud understanding [101]–[103]. Typically, these methods mask out entire voxels in the point cloud and aim to predict the missing data. The reconstructive loss differs between methods and can contain per-voxel point cloud reconstruction, voxel-level occupancy prediction, or a combination of both. Compared to pure supervised training, these methods are able to drastically reduce the amount of annotated data needed to train a perception model.

Beyond single-modal SSL for lidar, multiple works have tried to learn from image-lidar data [104]–[107]. UniPAD [106] uses an MAE-style reconstruction pretext task. There, images and lidar point clouds as masked and encoded to a joint space, from which images and point clouds can be decoded using differentiable volume rendering. The other works [104], [105] instead rely on pre-trained image models for supervision, and distill their knowledge into the lidar encoders. Their reasoning is based on image data being more abundant than lidar data, and hence the image

model should have a wider range of knowledge. In LidarCLIP [21] (Paper B), we follow a similar reasoning but emphasize the use of a vision-language model, which provides the trained LidarCLIP model with a direct connection to the language domain. GASP [107] combines both using visual foundation models with a reconstruction-based pretext. Given a collection of lidar scans, a lidar encoder is tasked to predict occupancy and DINOv2 [108] features at future frames. Although they do not have a direct connection to language, the DINOv2 features still provide semantic supervision to the lidar encoder.

As evident, there exists a range of tools for self-supervised learning in an autonomous driving context. More so, these often showcase improved performance compared to methods trained in a fully supervised fashion, especially when the number of annotated samples is small. This is in line with what is observed for self-supervised methods for general image classification as well. However, the conclusions drawn in the AD context are often based on relatively small sample sizes, as public AD datasets are much smaller than web-scale text or image datasets used to train language or vision foundation models. Thus, it remains an open question what the scaling properties are of SSL for AD, and what gains to expect in a more industrial setting.

CHAPTER 5

Going Beyond Real Data

As evident from the previous chapters, much of the development of autonomous vehicles revolves around high-quality data. Training data-driven models for AD requires collecting data and, to some extent, annotating it. Evaluating the performance of AD systems and their subsystems also requires diverse data and annotations. As such, the performance of an AD system, and practitioners' confidence in the system's expected behavior, is strongly connected to the amount and quality of data available.

However, collecting sufficient amounts of real-world data for training and evaluation is a challenging undertaking. Generally, practitioners are interested in scenarios where the AD system is struggling. However, as the performance of the AD system improves, the time between relevant events increases. Furthermore, certain events, especially safety-critical ones, are rare and difficult to collect. Some are even impossible to observe without risking human or property damage. As a result, data collection and curation become increasingly cumbersome as the system matures.

Another factor complicating data collection is that testing the entire system from pixel to torque requires the data to be dynamic and reactive to the decisions made by the AD system. The easiest way to achieve this is to let the system drive in the real world. This, however, comes with multiple drawbacks. Beyond scaling poorly, as highlighted above, real-world testing is not reproducible, *i.e.*, one cannot evaluate different versions of software in the exact same scenario. It is possible to show the

collected data to a new version of the AD stack, but its actions would not be actuated. As such, it is hard to tell whether a bug observed in a specific scenario has been resolved by a software update.

Simulation has been a long-standing solution to the problem of data scarcity [109], [110]. It allows for the generation of diverse and dynamic data in a controlled environment. Beyond data generation, simulation can also be used to test the performance of AD systems in closed loop in a safe and cost-effective manner. Moreover, for many simulation methods, the generated ground truth is known and there is no need for human annotations. Nonetheless, current simulation tools have not been able to provide a scalable path toward robust autonomy.

The main problem with simulation is that it fails to provide sensor-realistic and diverse data at scale. Physics-based simulation, like game engines, has come a long way but struggles to reach sensor realism unless at the cost of computational resources and extensive modeling effort. Diversity, in turn, is bottlenecked by asset creation, which often involves humans in the loop.

Novel View Synthesis (NVS) is the task of generating sensor data from new viewpoints given a finite collection of observations from a scene. The field has received increasing attention from the AD simulation community as it has the potential to overcome the limitations of physics-based simulation. Pioneering works such as Neural Radiance Fields (NeRFs) [25] can turn multi-view images of a static scene into a representation that allows practitioners to generate new images from arbitrary viewpoints. With the potential to extract assets needed for AD simulation directly from the data, NeRFs have sparked the development of methods that can do the same for large-scale dynamic environments with multiple types of sensors. Below we introduce the NVS task, recent advances in the field, and their adaptations to AD simulation.

5.1 Novel View Synthesis

Novel View Synthesis (NVS) is a task in computer vision and graphics that involves generating new images of a scene from viewpoints that were not present in the original set of images. Formally, given a set of images captured from different viewpoints, NVS aims to synthesize images from novel viewpoints by learning the underlying 3D structure and appearance of the scene. This process typically involves estimating the geometry, lighting, and material properties of the scene to produce photorealistic renderings from any desired viewpoint.

Neural Radiance Fields

Neural Radiance Field (NeRF) [25] is a method that aims to learn an implicit scene representation from a set of posed camera images. It sidesteps the explicit lighting and material modeling by only considering the radiance, *i.e.*, the radiant flux emitted, reflected, or transmitted from a surface. In NeRF, the underlying scene is represented as a continuous volumetric function of 3D coordinates $\mathbf{x} = (x, y, z)$ and viewing direction $\mathbf{d} = (\theta, \phi)$. The function is parameterized as an MLP that produces a density $\sigma(\mathbf{x})$ and a color $\mathbf{c}(\mathbf{x}, \mathbf{d}) = (c_r, c_g, c_b)$ for each 3D coordinate and viewing direction. Notably, the MLP input is embedded using positional encodings to ensure that the network can represent high-frequency functions [111]. To produce a pixel value at a given image coordinate, NeRFs cast a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ from the camera center $\mathbf{o}$ through the image plane and query the MLP along the ray. From classical volume rendering [112], the expected color $C(\mathbf{r})$ with bounds t_n and t_f is given by:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})\, dt, \tag{5.1}$$

where $T(t)$ is the transmittance along the ray expressed as:

$$T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{r}(s))\, ds\right). \tag{5.2}$$

In practice, the integral is approximated using the quadrature rule:

$$C(\mathbf{r}) \approx \sum_{i=1}^{N} T_i(1 - \exp(-\sigma_i \Delta t_i))\mathbf{c}_i, \tag{5.3}$$

where T_i is the transmittance at the i-th sample, σ_i is the density, and Δt_i is the distance between the i-th and $(i+1)$-th sample.

The search for optimal MLP parameters is posed as an optimization problem. Relying on differentiable rendering, the parameters are found by minimizing the difference between the synthesized and the ground-truth images. The loss function is defined as:

$$L = \sum_{\mathbf{r}\in\mathcal{R}} \left\| C(\mathbf{r}) - \hat{C}(\mathbf{r}) \right\|_2^2, \tag{5.4}$$

where $\mathcal{R}$ is the set of rays, potentially sampled from multiple images, $C(\mathbf{r})$ is the ground truth color of the pixel, and $\hat{C}(\mathbf{r})$ is the synthesized color. Given enough images from different viewpoints, NeRFs successfully learn a sensible geometry of the scene, and its radiance, producing high-quality renderings.

Since its introduction, NeRFs have been improved in multiple ways. The authors of [113] note that when training and testing images have different resolutions, NeRFs tend to produce blurry or aliased images because they sample the scene with a single ray per pixel. To efficiently emulate supersampling with multiple rays per pixel, they propose to use the expected positional embedding if one were to represent a pixel with multiple rays. Their proposed Mip-NeRF boosts the performance of NeRFs by a significant margin, without introducing any additional parameters or training time.

However, NeRFs still struggle to handle large, unbounded scenes and produce blurry or low-quality images in these cases. In [114] three main challenges are identified: (1) parameterization, NeRFs require 3D scene coordinates to be bounded, (2) efficiency, larger scenes require more network capacity, (3) ambiguity, the content of unbounded scenes can lie at any distance. First, Mip-NeRF 360 [114] introduces a non-linear coordinate transformation to contract the space beyond a certain distance. Second, Mip-NeRF 360 uses an online distillation method that relies on multiple MLPs to represent the scene on several scales. Last, the authors propose a new regularization loss that encourages density along rays to be concentrated in a narrow band.

In addition to improving the quality and modeling capabilities of NeRFs, another line of work focuses on improving the efficiency of NeRFs [115]–[122]. The original NeRF formulation takes on the order of days to optimize the MLP for a single scene and can run inference at < 1 FPS. The methods trying to address this can roughly be grouped into two categories, (1) changing the scene parameterization to accelerate both training and inference speed, and (2) baking the scene by pre-computing values for a different scene representation. For instance, Instant-NGP [115] proposes to exchange the positional encoding with a multi-resolution hash encoding, which allows for a drastically smaller MLP. The method trains in seconds to minutes and can run inference at real-time speeds. In contrast, Plenoctrees [117] first train a slightly modified NeRF, and then convert the NeRF into an octree-based representation. While this does not accelerate training, it allows for > 100 FPS inference.

3D Gaussian Splatting

3D Gaussian Splatting (3DGS) [123] is an NVS method that both trains fast and runs inference at real-time speeds. Instead of learning a neural network to represent the scene, 3DGS uses an explicit representation of 3D Gaussians to represent the geometry and appearance. The key to optimizing their shape and appearance is (1) a fast GPU-based rasterization-based renderer and (2) heuristics for adaptive density

control of the Gaussians. To render an image, all Gaussians are first projected onto the image plane and approximated as 2D Gaussians. The rasterizer divides the image into 16x16 pixel tiles and each Gaussian is assigned to the tile(s) they overlap with. Before alpha-blending, the visible Gaussians are sorted globally by their midpoint's distance to the camera. Last, each 16x16 tile launches one thread per pixel, and these collaboratively render the tile by traversing the sorted Gaussians front to back. As for NeRFs, the loss function is defined as the difference between the synthesized and the ground-truth images.

3DGS has spurred a range of follow-up works focused on improving different aspects of the method. For instance, Mip-Splatting [124] adjusts the rendering equation to overcome the aliasing artifacts of 3DGS. 3DGS-MCMC [125] proposes to see the 3DGS optimization as a Markov Chain Monte Carlo (MCMC) problem, and proposes methods to remove heuristic density control. StopThePop [126] improves the renderer by replacing the global sorting with a hierarchical one. This remedies popping artifacts when rotating the camera, *i.e.*, small viewpoint changes resulting in switches in the sorting order. All of these improve the quality of the rendered images without introducing any computational overhead.

Some works have taken a different path, where they rely on 3D Gaussians to represent the scene, but use ray-tracing instead of rasterization to render the images. This allows them to model secondary effects, such as depth of field, motion blur, rolling shutter, shadows, and specular highlights, which are hard to capture with rasterization. Further, while the 3DGS rendering assumes a pinhole camera for 3D to 2D projection, the ray-based formulation makes it easier to adapt other lenses, such as fisheye lenses. 3D Gaussian Ray Tracing [127] is an example of this line of work. There, 3D Gaussians are first converted to a proxy geometry and inserted into a bounding volume hierarchy (BVH). Then, using NVIDIA's general-purpose ray-tracing engine, OptiX [128], rays are traced to compute the visibility of the 3D Gaussians. EVER [129] also relies on BVH and OptiX for sorting Gaussians along each ray, but also proposes a new rendering equation where the interaction between overlapping Gaussians is modeled. While both these works enable novel applications, they come at the cost of increased computational complexity, and run $2-10\times$ slower than 3DGS.

Generative Modeling for NVS

Generative modeling is another line of work for solving NVS. Instead of relying on NeRF-style per-scene optimization, generative models are trained over large datasets.

By learning the underlying distribution of the data, these models have far better generalization capabilities than NeRFs or 3DGS. For instance, multiple such models are capable of predicting plausible 3D scenes from a single image [130], and are even able to generate 3D scenes from text descriptions [131]. Such problems are unsolvable with NeRFs or 3DGs alone.

The drawback of generative methods is that they often lack built-in multiview consistency, are harder to control with the same precision as rendering-based methods, and run much slower at inference. Because of this, generative models are often coupled with rendering-based methods [132], [133]. The rendering-based method is used to encapsulate the current scene, provide multiview consistency, and query the scene. The generative model is used to extend or regularize the existing scene, *e.g.*, generate unseen views, or generate completely new scenes.

This style of approach has received increasing attention in the AD simulation community as well [134]–[137]. Indeed, these methods are able to improve image fidelity when shifting the ego-vehicle to a different lane, for instance. However, generative models for AD are still in their infancy and the full potential of this approach has yet to be unlocked. Important questions to address moving forward include how to utilize generative models efficiently, rather than simply selecting views for generation based on heuristics. Further, there is a lack of tools for multi-modal data generation, *e.g.*, how to extend the lidar properties in a scene remains an open challenge.

5.2 Neural Rendering in AD Simulation

Neural rendering, *i.e.*, the collection of methods that learn a 3D scene representation from 2D images, has enormous potential for scalable AD simulation. However, AD data come with unique challenges for novel view synthesis. First, AD scenes can cover hundreds of meters within seconds, whereas standard NVS datasets are often collected walking around with a handheld camera. Second, AD scenes are dynamic, breaking one of the core assumptions of NeRF- and 3DGS-based methods that the scene is static. Last, data are often collected using not only multiple cameras but also other types of sensors. While several works separately address scale [138], dynamics [139], [140], multi-modal data [141], [142], or some combination of them [143], [144], the challenge in developing a simulator for AD is to handle all these aspects simultaneously.

Modeling Dynamics One of the first works that use NeRFs for AD scene reconstruction is Neural Scene Graphs (NSG) [145]. There, the scene is decomposed into a static background and a set of rigidly moving objects. By knowing how the objects traverse the scene, the static and dynamic parts can be composed at any given time. Further, this allows easy reconfiguration of the scene, *i.e.*, changing the location of the ego-vehicle or the objects, which is important for using the method in a simulator. Other works have a more implicit separation of static and dynamic scenes, such as EmerNeRF [24], where static and dynamic are two separate networks. Although this removes dependencies of annotations for the dynamics, it also limits the possibilities for reconfiguring the scene.

To model dynamic actors as rigid has since become a popular approach in the AD simulation community [23], [142], [146], [147]. The main drawback is that this is not a good approximation for deformable actors such as pedestrians and cyclists. Recently, this limitation was overcome by OmniRe [148], a 3DGS-based method that can handle non-rigid dynamic actors. Here, the dynamic actors are subdivided into rigid, general deformable, and SMPL nodes. SMPL [149] is a parametric model for representing the pose and shape of humans. The deformable nodes instead rely on learning a small neural network to deform their parts in a more flexible but less structured manner. This approach greatly improves the quality of the rendered images, especially for deformable actors. However, while conceptually simple, OmniRe relies on external tools [150] to track pedestrians before starting the optimization, thereby adding significant processing time.

Multi-Modal Data While early work focused on camera data only, multiple methods exist for lidar data [141], [142], or for jointly modeling camera and lidar data [26], [28]. Since many AD systems rely on lidar data for 3D understanding, the ability to faithfully render lidar data is crucial for realistic simulation. Thankfully, since lidar is a light-based sensor, it is possible to adapt existing NeRF-based methods to render lidar data. However, the peculiarities of lidar data require careful modeling of the sensor.

In NFL [142], the authors adjust the volume rendering equation to account for the emitted beam traveling from the sensor and back, and the associated loss in energy. In addition, they model multiple important sensor characteristics. For instance, beam divergence, *i.e.*, the spread of the emitted laser, is captured by tracing multiple rays per lidar point. Further, instead of RGB, the learned scene representation contains reflectance values and probabilities for a lidar beam not scattering back toward the sensor (ray drop probability). This type of accurate modeling is important for a

perception system to have the same behavior in simulation as in the real world [144], [151].

When it comes to modeling camera and lidar data jointly, there are only a few works that address this [26], [28]. It is common practice to use the lidar for supervising the scene representation since it provides valuable depth information, but less common to accurately render the lidar signal for novel views. UniSim [23] was one of the first to report metrics for both image and lidar rendering. However, as we discuss in Paper C, UniSim overlooks certain aspects such as beam divergence, ray drop probability, and rolling shutter effects, limiting their performance.

One modality that has received little attention is radar data [152]–[155]. Since radars are not light-based, it is challenging to adapt existing NeRF-based methods to render radar data. Further, radar detections are generally sparse and noisy in comparison to lidar or camera data, making the pure reconstruction-based optimization approach more difficult. DART [152] tries to overcome the challenge of low spatial resolution by using the range-Doppler domain. Unfortunately, none of the large-scale AD datasets provide range-Doppler data, limiting the applicability of the method.

Applications The most common motivation for using NVS in AD simulation is the ability to conduct realistic simulations at scale. NeuroNCAP [27] explores exactly this by using a NeRF-based renderer [26] to explore safety-critical scenarios created from real-world data, see an example in Fig. 5.1. In their closed-loop simulation, they evaluate two end-to-end AD systems [40], [41] that previously have reported state-of-the-art results on *open-loop* evaluation. The NeuroNCAP results show that both methods fail to handle even simple scenarios, such as stationary cars parked in the middle of the road. The observation that closed-loop evaluation does not correlate with open-loop performance is in line with previous observations [156], [157], and merits the development of tools for closed-loop evaluation.

However, as for traditional game-engine-based simulators, questions often arise revolving around the realism of the rendered scenes and if results achieved in simulation can be trusted and used to draw conclusion about real-world performance. Works like UniSim [23] and NeuRAD [26] include some results to assess this real-to-sim gap. There, perception models are first trained on real data and then evaluated on both real and simulated data. By evaluating if the perception models' predictions are the same on both types of data, it is possible to assess the realism of the simulated scenes. UniSim and NeuRAD show that perception models such as 3D object detector BEVFormer [158] and monocular depth estimator DepthAnything [159] reach similar performance on real and simulated data.

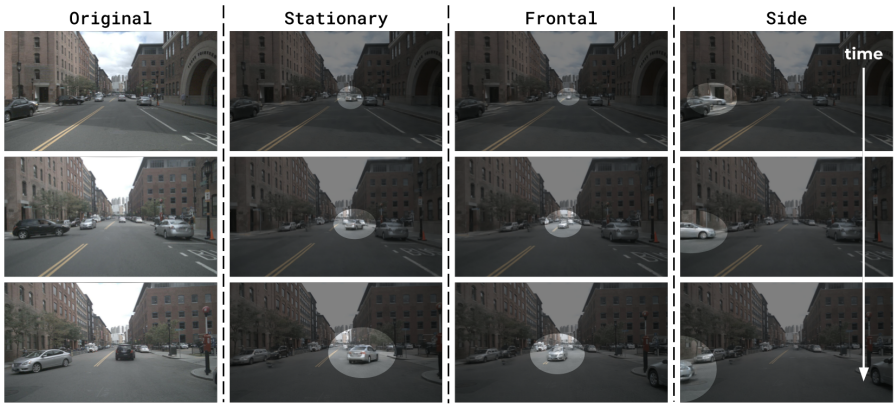

Figure 5.1: Example of how NeuroNCAP [27] uses NeuRAD (Paper C) to convert a collected real-world log (left) to three safety-critical scenarios: a parked car (second from left), potential frontal collision (second from right), and potential side collision (right). Image used with permission from [27].

These small-scale experiments are further extended in [160], where the authors explore the real-to-sim gap on a larger scale. There, the gap is evaluated for nuScenes data, using multiple 3D object detectors and an online mapping model. The results show a clear decline in perception performance when applying the same object detectors to the simulated data. However, this gap can be reduced by extending the data augmentation pipeline with either simulated data or data with a similar appearance as simulated data. Moreover, blending rare scenarios with simulated data can increase the performance on real-world data, indicating potential applications for neural rendering beyond verification and testing.

CHAPTER 6

Summary of included papers

This chapter provides a summary of the included papers.

6.1 Paper A

Georg Hess, Christoffer Petersson, Lennart Svensson
Object Detection as Probabilistic Set Prediction
Published in European Conference on Computer Vision (ECCV),
pp. 550–566, 2022.

DOI: 10.1007/978-3-031-20080-9_32.

We propose to view the task of object detection as probabilistic set prediction. Specifically, we model the predictions from an object detector as a Random Finite Set (RFS). We demonstrate how to acquire the required parameters from existing probabilistic object detectors. Furthermore, we propose to use the negative log-likelihood of the predicted RFS as a new object detection evaluation metric. This has appealing theoretical properties, as it is minimized only when the RFS matches the true data-generating distribution and further accounts for ambiguities when assigning predictions to ground truth.

The original idea behind the project came from LS and was refined in project meetings by me, LS, and CH. I implemented baselines, conducted experiments, and summarized the results. The majority of the paper was written by me, with a large amount of valuable feedback from LS and CH.

6.2 Paper B

Georg Hess†, Adam Tonderski†, Christoffer Petersson, Kalle Åström, Lennart Svensson
LidarCLIP or: How I Learned to Talk to Point Clouds
Published in IEEE/CVF Winter Conference on Applications of Computer Vision (WACV),
pp. 7423–7432, 2024.
DOI: 10.1109/WACV57701.2024.00727

We propose a method for relating 3D point clouds and text, without any need for point cloud-text pairs. Instead, we use image-point cloud pairs to supervise a point cloud encoder with image CLIP embeddings, effectively creating a text and point cloud coupling with the image domain as an intermediary. We find that our learned lidar encoder is on par with the image encoder for retrieval in the autonomous driving context. Furthermore, we propose to use both encoders for multi-modal retrieval. This method is more robust to adverse conditions and can further be used to explicitly find these types of scenarios.

Me and AT jointly conceptualized the project, with AT first proposing the core idea. Together, we implemented the method, ran experiments, analyzed the results, and wrote the paper. KÅ, LS, and CP provided feedback.

6.3 Paper C

Adam Tonderski†, Carl Lindström†, **Georg Hess**†, William Ljungbergh, Lennart Svensson, Christoffer Petersson
NeuRAD: Neural Rendering for Autonomous Driving

Published in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),
pp. 14895–14904, 2024.
DOI: 10.1109/CVPR52733.2024.01411

We propose a new method for novel view synthesis in an autonomous driving context. Our method is the first to jointly model lidar and multi-camera data. Furthermore, we were the first to emphasize the modeling of rolling shutter effects for accurate novel view synthesis. By combining this with modeling of other sensor characteristics, such as camera antialiasing and lidar beam divergence, our method set a new state of the art across five popular autonomous driving datasets.

AT and I came up with the original idea for the project in discussions with LS and CP. The method was then developed by AT, me, and CL, in discussion with LS and CP. The implementation was done by AT, me, and CL, with assistance from WL. The final model architecture was proposed by AT. Experiments were run by CL, AT, and me. I created much of the first draft of the paper, and writing was then done mainly by me, CL, and AT, with support and feedback from all co-authors.

6.4 Paper D

Georg Hess[†], Carl Lindström[†], Maryam Fatemi, Christoffer Petersson, Lennart Svensson
SplatAD: Real-Time Lidar and Camera Rendering with 3D Gaussian Splatting for Autonomous Driving
To be published in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2025.

Training NeuRAD takes several hours and using it for simulation requires a significant amount of computational resources, as it does not run in real-time. Here, we propose a new method for novel view synthesis based on 3D Gaussian Splatting, which enables real-time simulation and faster training times. After a few minutes of training, SplatAD reaches a performance comparable to that of the previous state-of-the-art. Furthermore, longer training leads to SplatAD clearly surpassing the previous state of the art. To enable this, we extend the original 3D Gaussian Splatting formulation

from modeling single camera only, to jointly modeling multi-camera and lidar data.

I proposed the project in discussion with LS and CP. Me and CL designed and implemented the method together. Me and CL ran experiments with feedback from LS, CP, and MF on which experiments to prioritize. Me and CL wrote the paper together, where I wrote a majority of the first draft, and LS, CP, and MF, provided feedback.

CHAPTER 7

Concluding Remarks

Throughout this thesis, we have addressed three different challenges in the field of autonomous driving perception: uncertainty estimation in object detection, self-supervised learning for reduced need for annotation, and neural rendering for scalable development. With these contributions, perception systems can reason about occluded objects, utilize language to learn from unlabeled data, and be trained and evaluated in realistic simulation environments. While we believe that these contributions are a step in the right direction, there is still a lot of work to be done before we have affordable and safe autonomous driving in every corner of the world.

Uncertainty Estimation Evaluation Our proposed framework for modeling object detection uncertainty enables fair and principled evaluation of different object detectors. However, as highlighted by the emerging end-to-end AD systems, it is not clear how these performance measures align with system-level performance. For instance, in many perception metrics, all objects are treated as equally important, but in reality, only a handful of objects, such as the ones closest to the ego vehicle, truly impact the decision-making process. To figure out the impact of poor detections, we would have to evaluate multiple "what if" scenarios. While this is not feasible in a real-world setting, we hope that our work on novel view synthesis will enable such features in the future.

Scaling LidarCLIP and Introducing Additional Modalities LidarCLIP demonstrated an impressive semantic understanding of 3D lidar data and decent cross-dataset generalization by training on 7 million image-lidar pairs. Nonetheless, the amount of data is still orders of magnitude smaller than what is used for training most image-language models. Also, most AD systems rely on additional sensors, such as radars, which is not handled by LidarCLIP. It would be interesting to see emerging properties when scaling LidarCLIP to even larger datasets and model sizes. Both NLP and CV fields have repeatedly shown how scaling data and models can unlock new capabilities. Further, approaches similar to Imagebind [161] could enable methods to include additional modalities, and, perhaps more importantly, enable the image model to learn from these modalities. This way, instead of mapping 3D data to a semantic embedding space, the embedding space will also contain 3D information.

While self-supervised learning reduces the need for costly human annotation, it often requires massive datasets, leading to significant data collection and storage costs. For reference, the best-performing LLMs such as LLama 3 [162] are trained on 15T tokens. Translating a standard Full-HD image to about 8K tokens[1], the equivalent would be 2B images. Furthermore, text is a more readily available resource than synchronized multi-sensor data of AD systems. This raises questions on how to utilize the data more efficiently. Potential avenues include continual learning [163] and meta-learning [164]. While not addressing the data collection problem, these approaches can enable the model to learn from new data without forgetting previously learned information.

Extending Neural Rendering Applications Both NeuRAD and SplatAD provide high-fidelity camera-lidar data for novel views close to the original training views. In simulation, however, the ego-vehicle can diverge significantly from the original trajectory. To extend the applicability of these methods, additional regularization is needed. The most promising direction is to rely on the fast progress in generative models and use them to extend the learned scene representations [132], [133]. An open research problem is how to extend these methods to include additional modalities, such as lidars and radars. Perhaps an SSL model with multi-modal embedding space, as discussed above, can be a key to unlocking these applications.

Onboard Compute Irrespective of the amount of high-quality data, an AD system will be bound by the amount of available onboard computing power. While there are

[1] A 16:9 Full-HD image has 1920x1080 pixels. Using the common 16x16 tokenization of Vision Transformers, this would be 8100 tokens.

limited studies on scaling laws for AD, other DL-driven domains clearly show that model size and dataset size should be scaled in tandem for increasing performance [165]. Thus, to improve AD systems, we likely need more powerful and affordable onboard compute. Further, access to compute that is capable beyond running the model just-in-time could open up new directions, similar to the test-time reasoning capabilities observed in recent LLMs [166]–[168]. This way, the model could use more compute power for complicated scenarios, and less for nominal driving.

References

[1] L. Di Lillo, T. Gode, X. Zhou, J. M. Scanlon, R. Chen, and T. Victor, "Do autonomous vehicles outperform latest-generation human-driven vehicles? a comparison to waymo's auto liability insurance claims at 25.3m miles," Waymo Research, Tech. Rep., 2024.

[2] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[3] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, vol. 111, no. 3, pp. 257–276, 2023.

[4] D. Hall, F. Dayoub, J. Skinner, H. Zhang, D. Miller, P. Corke, G. Carneiro, A. Angelova, and N. Sünderhauf, "Probabilistic object detection: Definition and evaluation," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2020, pp. 1031–1040.

[5] A. Harakeh and S. L. Waslander, "Estimating and evaluating regression predictive uncertainty in deep object detectors," in *International Conference on Learning Representations (ICLR)*, 2021.

[6] G. Hess, C. Petersson, and L. Svensson, "Object detection as probabilistic set prediction," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer, 2022, pp. 550–566.

[7] R. Mahler, "PHD filters of higher order in target number," *IEEE Transactions on Aerospace and Electronic systems*, vol. 43, no. 4, pp. 1523–1543, 2007.

[8] P. Ren, Y. Xiao, X. Chang, P.-Y. Huang, Z. Li, B. B. Gupta, X. Chen, and X. Wang, "A survey of deep active learning," *ACM computing surveys (CSUR)*, vol. 54, no. 9, pp. 1–40, 2021.

[9] C. R. Qi, Y. Zhou, M. Najibi, P. Sun, K. Vo, B. Deng, and D. Anguelov, "Offboard 3d object detection from point cloud sequences," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 6134–6144.

[10] L. Fan, Y. Yang, Y. Mao, F. Wang, Y. Chen, N. Wang, and Z. Zhang, "Once detected, never lost: Surpassing human performance in offline lidar based 3d object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 19 820–19 829.

[11] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[12] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[13] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 1877–1901, 2020.

[14] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 9650–9660.

[15] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 16 000–16 009.

[16] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick, "Momentum contrast for unsupervised visual representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[17] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 119, PMLR, Jul. 2020, pp. 1597–1607.

[18] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," in *Proceedings of the International Conference on Machine Learning (ICML)*, ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 8748–8763.

[19] R. Zhang, Z. Guo, W. Zhang, K. Li, X. Miao, B. Cui, Y. Qiao, P. Gao, and H. Li, "Pointclip: Point cloud understanding by CLIP," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 8552–8562.

[20] T. Huang, B. Dong, Y. Yang, X. Huang, R. W. Lau, W. Ouyang, and W. Zuo, "Clip2point: Transfer clip to point cloud classification with image-depth pre-training," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[21] G. Hess, A. Tonderski, C. Petersson, K. Åström, and L. Svensson, "LidarCLIP or: How i learned to talk to point clouds," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2024, pp. 7438–7447.

[22] C. Gulino, J. Fu, W. Luo, G. Tucker, E. Bronstein, Y. Lu, J. Harb, X. Pan, Y. Wang, X. Chen, *et al.*, "Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[23] Z. Yang, Y. Chen, J. Wang, S. Manivasagam, W.-C. Ma, A. J. Yang, and R. Urtasun, "Unisim: A neural closed-loop sensor simulator," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 1389–1399.

[24] J. Yang, B. Ivanovic, O. Litany, X. Weng, S. W. Kim, B. Li, T. Che, D. Xu, S. Fidler, M. Pavone, and Y. Wang, "EmerneRF: Emergent spatial-temporal scene decomposition via self-supervision," in *International Conference on Learning Representations (ICLR)*, 2024.

[25] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Cham: Springer International Publishing, 2020, pp. 405–421, ISBN: 978-3-030-58452-8.

[26] A. Tonderski, C. Lindström, G. Hess, W. Ljungbergh, L. Svensson, and C. Petersson, "Neurad: Neural rendering for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 14 895–14 904.

[27] W. Ljungbergh, A. Tonderski, J. Johnander, H. Caesar, K. Åström, M. Felsberg, and C. Petersson, "NeuroNCAP: Photorealistic closed-loop safety testing for autonomous driving," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer, 2025, pp. 161–177.

[28] G. Hess, C. Lindström, M. Fatemi, C. Petersson, and L. Svensson, "Splatad: Real-time lidar and camera rendering with 3d gaussian splatting for autonomous driving," *arXiv preprint arXiv:2411.16816*, 2024.

[29] I. A. Bârsan, "Learning rich representations for robot state estimation," Ph.D. dissertation, University of Toronto (Canada), 2024.

[30] Tesla, *Tesla vision update: Replacing ultrasonic sensors with tesla vision*, `https://web.archive.org/web/20241225130228/https://www.tesla.com/support/transitioning-tesla-vision`, 2024.

[31] G. Chen, H. Cao, J. Conradt, H. Tang, F. Rohrbein, and A. Knoll, "Event-based neuromorphic vision for autonomous driving: A paradigm shift for bio-inspired visual sensing and perception," *IEEE Signal Processing Magazine*, vol. 37, no. 4, pp. 34–49, 2020.

[32] B. Miethig, A. Liu, S. Habibi, and M. v. Mohrenschildt, "Leveraging thermal imaging for autonomous driving," in *2019 IEEE Transportation Electrification Conference and Expo (ITEC)*, IEEE, 2019, pp. 1–5.

[33] M. Alibeigi, W. Ljungbergh, A. Tonderski, G. Hess, A. Lilja, C. Lindström, D. Motorniuk, J. Fu, J. Widahl, and C. Petersson, "Zenseact open dataset: A large-scale and diverse multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 20 178–20 188.

[34] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, "End-to-end autonomous driving: Challenges and frontiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[35] T. Meinhardt, A. Kirillov, L. Leal-Taixe, and C. Feichtenhofer, "Trackformer: Multi-object tracking with transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 8844–8854.

[36] M. Gulzar, Y. Muhammad, and N. Muhammad, "A survey on motion prediction of pedestrians and vehicles for autonomous driving," *IEEE Access*, vol. 9, pp. 137 957–137 969, 2021.

[37] S. Teng, X. Hu, P. Deng, B. Li, Y. Li, Y. Ai, D. Yang, L. Li, Z. Xuanyuan, F. Zhu, *et al.*, "Motion planning for autonomous driving: The state of the art and future perspectives," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 6, pp. 3692–3711, 2023.

[38] I. Batkovic, "Enabling safe autonomous driving in uncertain environments: Based on a model predivtive control approach," Ph.D. dissertation, Chalmers University of Technology, 2022.

[39] S. Gould, R. Hartley, and D. Campbell, "Deep declarative networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 8, pp. 3988–4004, 2021.

[40] 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 (CVPR)*, 2023, pp. 17 853–17 862.

[41] B. Jiang, S. Chen, Q. Xu, B. Liao, J. Chen, H. Zhou, Q. Zhang, W. Liu, C. Huang, and X. Wang, "Vad: Vectorized scene representation for efficient autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 8340–8350.

[42] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *arXiv preprint arXiv:1708.06374*, 2018.

[43] A. Sadat, S. Segal, S. Casas, J. Tu, B. Yang, R. Urtasun, and E. Yumer, "Diverse complexity measures for dataset curation in self-driving," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 8609–8616.

[44] L. Li, W. Shao, W. Dong, Y. Tian, Q. Zhang, K. Yang, and W. Zhang, "Data-centric evolution in autonomous driving: A comprehensive survey of big data system, data mining, and closed-loop technologies," *arXiv preprint arXiv:2401.12888*, 2024.

[45] J. Lin, Z. Liang, S. Deng, L. Cai, T. Jiang, T. Li, K. Jia, and X. Xu, "Exploring diversity-based active learning for 3d object detection in autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[46] 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 (CVPR)*, 2020, pp. 11 621–11 631.

[47] B. Wilson, W. Qi, T. Agarwal, J. Lambert, J. Singh, S. Khandelwal, B. Pan, R. Kumar, A. Hartnett, J. K. Pontes, D. Ramanan, P. Carr, and J. Hays, "Argoverse 2: Next generation datasets for self-driving perception and forecasting," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[48] 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 (CVPR)*, 2020, pp. 2446–2454.

[49] C. M. Jiang, M. Najibi, C. R. Qi, Y. Zhou, and D. Anguelov, "Improving the intra-class long-tail in 3d detection via rare example mining," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer, 2022, pp. 158–175.

[50] Tesla, *Tesla ai day 2022*, `https://www.youtube.com/watch?v=ODSJsviD_SU`, 2022.

[51] European New Car Assessment Programme (Euro NCAP), *Euro ncap test and assessment protocols*, Accessed: 2025-02-21, 2025.

[52] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[53] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell, "Agent57: Outperforming the atari human benchmark," in *International conference on machine learning*, PMLR, 2020, pp. 507–517.

[54] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[55] M. Cusumano-Towner, D. Hafner, A. Hertzberg, B. Huval, A. Petrenko, E. Vinitsky, E. Wijmans, T. Killian, S. Bowers, O. Sener, *et al.*, "Robust autonomy emerges from self-play," *arXiv preprint arXiv:2502.03349*, 2025.

[56] H. Caesar, J. Kabzan, K. S. Tan, W. K. Fong, E. Wolff, A. Lang, L. Fletcher, O. Beijbom, and S. Omari, "Nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles," in *CVPR ADP3 workshop*, 2021.

[57] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[58] T. Tieleman, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, p. 26, 2012.

[59] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" In *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[60] J. Denker and Y. LeCun, "Transforming neural-net output levels to probability distributions," *Advances in neural information processing systems*, vol. 3, 1990.

[61] R. M. Neal, *Bayesian learning for neural networks*. Springer Science & Business Media, 2012, vol. 118.

[62] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors. arxiv 2012," *arXiv preprint arXiv:1207.0580*, 2012.

[63] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[64] T. Gneiting and A. E. Raftery, "Strictly proper scoring rules, prediction, and estimation," *Journal of the American statistical Association*, vol. 102, no. 477, pp. 359–378, 2007.

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

[66] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer, 2014, pp. 740–755.

[67] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[68] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, *et al.*, "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale," *International Journal of Computer Vision*, vol. 128, no. 7, pp. 1956–1981, 2020.

[69] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[70] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer, 2016, pp. 21–37.

[71] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 580–587.

[72] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 28, 2015.

[73] R. Girshick, "Fast r-cnn," *arXiv preprint arXiv:1504.08083*, 2015.

[74] J. Hosang, R. Benenson, and B. Schiele, "Learning non-maximum suppression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4507–4515.

[75] R. Stewart, M. Andriluka, and A. Y. Ng, "End-to-end people detection in crowded scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2325–2333.

[76] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, Curran Associates, Inc., 2017.

[77] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with Transformers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer, 2020, pp. 213–229.

[78] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable transformers for end-to-end object detection," in *International Conference on Learning Representations (ICLR)*, 2021.

[79] S. Liu, F. Li, H. Zhang, X. Yang, X. Qi, H. Su, J. Zhu, and L. Zhang, "DAB-DETR: Dynamic anchor boxes are better queries for DETR," in *International Conference on Learning Representations (ICLR)*, 2022.

[80] D. Meng, X. Chen, Z. Fan, G. Zeng, H. Li, Y. Yuan, L. Sun, and J. Wang, "Conditional DETR for fast training convergence," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 3651–3660.

[81] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[82] Á. F. García-Fernández, J. L. Williams, K. Granström, and L. Svensson, "Poisson multi-Bernoulli mixture filter: Direct derivation and implementation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 54, no. 4, pp. 1883–1901, 2018.

[83] B. Vo, B. Vo, and H. G. Hoang, "An efficient implementation of the generalized labeled multi-Bernoulli filter," *IEEE Transactions on Signal Processing*, vol. 65, no. 8, pp. 1975–1987, 2017.

[84] J. Pinto, Y. Xia, L. Svensson, and H. Wymeersch, "An uncertainty-aware performance measure for multi-object tracking," *IEEE Signal Processing Letters*, vol. 28, pp. 1689–1693, 2021.

[85] K. G. Murty, "An algorithm for ranking all the assignments in order of increasing cost," *Operations research*, vol. 16, no. 3, pp. 682–687, 1968.

[86] J. Gui, T. Chen, J. Zhang, Q. Cao, Z. Sun, H. Luo, and D. Tao, "A survey on self-supervised learning: Algorithms, applications, and future trends," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[87] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," *OpenAI Blog*, 2018.

[88] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, 2006, pp. 1735–1742.

[89] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *Proceedings of the International Conference on Machine Learning (ICML)*, PMLR, 2020, pp. 1597–1607.

[90] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, *et al.*, "Bootstrap your own latent-a new approach to self-supervised learning," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 21 271–21 284.

[91] X. Chen and K. He, "Exploring simple siamese representation learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 15 750–15 758.

[92] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin, "Unsupervised learning of visual features by contrasting cluster assignments," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[93] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations (ICLR)*, 2021.

[94] Z. Ren, S. Wang, and Y. Zhang, "Weakly supervised machine learning," *CAAI Transactions on Intelligence Technology*, vol. 8, no. 3, pp. 549–580, 2023.

[95] Z.-H. Zhou, "A brief introduction to weakly supervised learning," *National science review*, vol. 5, no. 1, pp. 44–53, 2018.

[96] S. Xie, J. Gu, D. Guo, C. R. Qi, L. Guibas, and O. Litany, "Pointcontrast: Unsupervised pre-training for 3d point cloud understanding," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer, 2020, pp. 574–591.

[97] L. Nunes, R. Marcuzzi, X. Chen, J. Behley, and C. Stachniss, "Segcontrast: 3d point cloud feature representation learning through self-supervised segment discrimination," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2116–2123, 2022.

[98] H. Liang, C. Jiang, D. Feng, X. Chen, H. Xu, X. Liang, W. Zhang, Z. Li, and L. Van Gool, "Exploring geometry-aware contrast and clustering harmonization for self-supervised 3d object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 3293–3302.

[99] J. Yin, D. Zhou, L. Zhang, J. Fang, C.-Z. Xu, J. Shen, and W. Wang, "Proposalcontrast: Unsupervised pre-training for lidar-based 3d object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer, 2022, pp. 17–33.

[100] C. Sautier, G. Puy, A. Boulch, R. Marlet, and V. Lepetit, "Bevcontrast: Self-supervision in bev space for automotive lidar point clouds," in *International Conference on 3D Vision (3DV)*, IEEE, 2024, pp. 559–568.

[101] G. Hess, J. Jaxing, E. Svensson, D. Hagerman, C. Petersson, and L. Svensson, "Masked autoencoder for self-supervised pre-training on lidar point clouds," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2023, pp. 350–359.

[102] H. Yang, T. He, J. Liu, H. Chen, B. Wu, B. Lin, X. He, and W. Ouyang, "Gd-mae: Generative decoder for mae pre-training on lidar point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 9403–9414.

[103] G. Krispel, D. Schinagl, C. Fruhwirth-Reisinger, H. Possegger, and H. Bischof, "Maeli: Masked autoencoder for large-scale lidar point clouds," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2024, pp. 3383–3392.

[104] C. Sautier, G. Puy, S. Gidaris, A. Boulch, A. Bursuc, and R. Marlet, "Image-to-lidar self-supervised distillation for autonomous driving data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 9891–9901.

[105] Y. Liu, L. Kong, J. Cen, R. Chen, W. Zhang, L. Pan, K. Chen, and Z. Liu, "Segment any point cloud sequences by distilling vision foundation models," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, pp. 37 193–37 229, 2023.

[106] H. Yang, S. Zhang, D. Huang, X. Wu, H. Zhu, T. He, S. Tang, H. Zhao, Q. Qiu, B. Lin, *et al.*, "Unipad: A universal pre-training paradigm for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 15 238–15 250.

[107] W. Ljungbergh, A. Lilja, A. T. Ling, C. Lindström, W. Verbeke, J. Fu, C. Petersson, L. Hammarstrand, M. Felsberg, *et al.*, "Gasp: Unifying geometric and semantic self-supervised pre-training for autonomous driving," *arXiv preprint arXiv:2503.15672*, 2025.

[108] M. Oquab, T. Darcet, T. Moutakanni, H. V. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. HAZIZA, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski, "DINOv2: Learning robust visual features without supervision," *Transactions on Machine Learning Research*, 2024, ISSN: 2835-8856.

[109] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," in *Proceedings of the Conference on Robot Learning (CORL)*, PMLR, 2017, pp. 1–16.

[110] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics: Results of the 11th International Conference*, Springer, 2018, pp. 621–635.

[111] M. Tancik, P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. Barron, and R. Ng, "Fourier features let networks learn high frequency functions in low dimensional domains," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 7537–7547, 2020.

[112] J. T. Kajiya and B. P. Von Herzen, "Ray tracing volume densities," *ACM SIGGRAPH computer graphics*, vol. 18, no. 3, pp. 165–174, 1984.

[113] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 5855–5864.

[114] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-NeRF 360: Unbounded anti-aliased neural radiance fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 5470–5479.

[115] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics*, vol. 41, no. 4, pp. 1–15, 2022.

[116] S. Fridovich-Keil, A. Yu, M. Tancik, Q. Chen, B. Recht, and A. Kanazawa, "Plenoxels: Radiance fields without neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 5501–5510.

[117] A. Yu, R. Li, M. Tancik, H. Li, R. Ng, and A. Kanazawa, "Plenoctrees for real-time rendering of neural radiance fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 5752–5761.

[118] S. Fridovich-Keil, G. Meanti, F. R. Warburg, B. Recht, and A. Kanazawa, "K-planes: Explicit radiance fields in space, time, and appearance," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 12 479–12 488.

[119] A. Chen, Z. Xu, A. Geiger, J. Yu, and H. Su, "Tensorf: Tensorial radiance fields," in *Proceedings of the European Conference on Computer Vision (ECCV)*, Springer, 2022, pp. 333–350.

[120] Z. Chen, T. Funkhouser, P. Hedman, and A. Tagliasacchi, "Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 16 569–16 578.

[121] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Zip-NeRF: Anti-aliased grid-based neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 19 697–19 705.

[122] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec, "Baking neural radiance fields for real-time view synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 5875–5884.

[123] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, pp. 1–14, 2023.

[124] Z. Yu, A. Chen, B. Huang, T. Sattler, and A. Geiger, "Mip-splatting: Alias-free 3d gaussian splatting," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 19 447–19 456.

[125] S. Kheradmand, D. Rebain, G. Sharma, W. Sun, Y.-C. Tseng, H. Isack, A. Kar, A. Tagliasacchi, and K. M. Yi, "3d gaussian splatting as markov chain monte carlo," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[126] L. Radl, M. Steiner, M. Parger, A. Weinrauch, B. Kerbl, and M. Steinberger, "Stopthepop: Sorted gaussian splatting for view-consistent real-time rendering," *ACM Transactions on Graphics (TOG)*, vol. 43, no. 4, pp. 1–17, 2024.

[127] N. Moenne-Loccoz, A. Mirzaei, O. Perel, R. de Lutio, J. Martinez Esturo, G. State, S. Fidler, N. Sharp, and Z. Gojcic, "3d gaussian ray tracing: Fast tracing of particle scenes," *ACM Transactions on Graphics*, vol. 43, no. 6, pp. 1–19, 2024.

[128] S. G. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, *et al.*, "Optix: A general purpose ray tracing engine," *ACM Transactions on Graphics*, vol. 29, no. 4, pp. 1–13, 2010.

[129] A. Mai, P. Hedman, G. Kopanas, D. Verbin, D. Futschik, Q. Xu, F. Kuester, J. T. Barron, and Y. Zhang, "Ever: Exact volumetric ellipsoid rendering for real-time view synthesis," *arXiv preprint arXiv:2410.01804*, 2024.

[130] R. Liu, R. Wu, B. Van Hoorick, P. Tokmakov, S. Zakharov, and C. Vondrick, "Zero-1-to-3: Zero-shot one image to 3d object," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 9298–9309.

[131] C.-H. Lin, J. Gao, L. Tang, T. Takikawa, X. Zeng, X. Huang, K. Kreis, S. Fidler, M.-Y. Liu, and T.-Y. Lin, "Magic3d: High-resolution text-to-3d content creation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 300–309.

[132] R. Gao, A. Holynski, P. Henzler, A. Brussee, R. Martin-Brualla, P. P. Srinivasan, J. T. Barron, and B. Poole, "Cat3d: Create anything in 3d with multi-view diffusion models," *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[133] R. Wu, B. Mildenhall, P. Henzler, K. Park, R. Gao, D. Watson, P. P. Srinivasan, D. Verbin, J. T. Barron, B. Poole, *et al.*, "Reconfusion: 3d reconstruction with diffusion priors," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 21 551–21 561.

[134] Z. Yu, H. Wang, J. Yang, H. Wang, Z. Xie, Y. Cai, J. Cao, Z. Ji, and M. Sun, "Sgd: Street view synthesis with gaussian splatting and diffusion prior," *arXiv preprint arXiv:2403.20079*, 2024.

[135] C. Ni, G. Zhao, X. Wang, Z. Zhu, W. Qin, G. Huang, C. Liu, Y. Chen, Y. Wang, X. Zhang, *et al.*, "Recondreamer: Crafting world models for driving scene reconstruction via online restoration," *arXiv preprint arXiv:2411.19548*, 2024.

[136] L. Fan, H. Zhang, Q. Wang, H. Li, and Z. Zhang, "Freesim: Toward free-viewpoint camera simulation in driving scenes," *arXiv preprint arXiv:2412.03-566*, 2024.

[137] Z. Yang, Z. Pan, Y. Yang, X. Zhu, and L. Zhang, "Driving scene synthesis on free-form trajectories with generative prior," *arXiv preprint arXiv:2412.01717*, 2024.

[138] M. Tancik, V. Casser, X. Yan, S. Pradhan, B. Mildenhall, P. P. Srinivasan, J. T. Barron, and H. Kretzschmar, "Block-nerf: Scalable large scene neural view synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 8248–8258.

[139] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-nerf: Neural radiance fields for dynamic scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 10 318–10 327.

[140] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla, "Nerfies: Deformable neural radiance fields," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 5865–5874.

[141] J. Zhang, F. Zhang, S. Kuang, and L. Zhang, "Nerf-lidar: Generating realistic lidar point clouds with neural radiance fields," in *Proceedings of the national conference on Artificial Intelligence (AAAI)*, vol. 38, 2024, pp. 7178–7186.

[142] S. Huang, Z. Gojcic, Z. Wang, F. Williams, Y. Kasten, S. Fidler, K. Schindler, and O. Litany, "Neural lidar fields for novel view synthesis," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 18 236–18 246.

[143] H. Turki, J. Y. Zhang, F. Ferroni, and D. Ramanan, "SUDS: Scalable urban dynamic scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 12 375–12 385.

[144] H. Wu, X. Zuo, S. Leutenegger, O. Litany, K. Schindler, and S. Huang, "Dynamic lidar re-simulation using compositional neural fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 19 988–19 998.

[145] J. Ost, F. Mannan, N. Thuerey, J. Knodt, and F. Heide, "Neural scene graphs for dynamic scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 2856–2865.

[146] Z. Xie, J. Zhang, W. Li, F. Zhang, and L. Zhang, "S-neRF: Neural radiance fields for street views," in *International Conference on Learning Representations (ICLR)*, 2023.

[147] Y. Siddiqui, L. Porzi, S. R. Bulò, N. Müller, M. Nießner, A. Dai, and P. Kontschieder, "Panoptic lifting for 3d scene understanding with neural fields," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 9043–9052.

[148] Z. Chen, J. Yang, J. Huang, R. de Lutio, J. M. Esturo, B. Ivanovic, O. Litany, Z. Gojcic, S. Fidler, M. Pavone, *et al.*, "Omnire: Omni urban scene reconstruction," *arXiv preprint arXiv:2408.16760*, 2024.

[149] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, "SMPL: A skinned multi-person linear model," *ACM Transactions on Graphics*, vol. 34, no. 6, 248:1–248:16, 2015.

[150] S. Goel, G. Pavlakos, J. Rajasegaran, A. Kanazawa, and J. Malik, "Humans in 4d: Reconstructing and tracking humans with transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 14 783–14 794.

[151] S. Manivasagam, I. A. Bârsan, J. Wang, Z. Yang, and R. Urtasun, "Towards zero domain gap: A comprehensive study of realistic lidar simulation for autonomy testing," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 8272–8282.

[152] T. Huang, J. Miller, A. Prabhakara, T. Jin, T. Laroia, Z. Kolter, and A. Rowe, "Dart: Implicit doppler tomography for radar novel view synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 24 118–24 129.

[153] T. Ehret, R. Marí, D. Derksen, N. Gasnier, and G. Facciolo, "Radar fields: An extension of radiance fields to sar," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 564–574.

[154] Z. Lei, F. Xu, J. Wei, F. Cai, F. Wang, and Y.-Q. Jin, "Sar-nerf: Neural radiance fields for synthetic aperture radar multi-view representation," *IEEE Transactions on Geoscience and Remote Sensing*, 2024.

[155] M. Rafidashti, J. Lan, M. Fatemi, J. Fu, L. Hammarstrand, and L. Svensson, "Neuradar: Neural radiance fields for automotive radar point clouds," *arXiv preprint arXiv:2504.00859*, 2025.

[156] J.-T. Zhai, Z. Feng, J. Du, Y. Mao, J.-J. Liu, Z. Tan, Y. Zhang, X. Ye, and J. Wang, "Rethinking the open-loop evaluation of end-to-end autonomous driving in nuscenes," *arXiv preprint arXiv:2305.10430*, 2023.

[157] D. Dauner, M. Hallgarten, A. Geiger, and K. Chitta, "Parting with misconceptions about learning-based vehicle motion planning," in *Conference on Robot Learning*, PMLR, 2023, pp. 1268–1281.

[158] Z. Li, W. Wang, H. Li, E. Xie, C. Sima, T. Lu, Q. Yu, and J. Dai, "Bevformer: Learning bird's-eye-view representation from lidar-camera via spatiotemporal transformers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[159] L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng, and H. Zhao, "Depth anything: Unleashing the power of large-scale unlabeled data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 10 371–10 381.

[160] C. Lindström, G. Hess, A. Lilja, M. Fatemi, L. Hammarstrand, C. Petersson, and L. Svensson, "Are NeRFs ready for autonomous driving? towards closing the real-to-simulation gap," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2024, pp. 4461–4471.

[161] R. Girdhar, A. El-Nouby, Z. Liu, M. Singh, K. V. Alwala, A. Joulin, and I. Misra, "Imagebind: One embedding space to bind them all," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 15 180–15 190.

[162] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[163] L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: Theory, method and application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[164] C. Finn and S. Levine, "Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm," in *International Conference on Learning Representations (ICLR)*, 2018.

[165] Y. Zheng, Z. Xia, Q. Zhang, T. Zhang, B. Lu, X. Huo, C. Han, Y. Li, M. Yu, B. Jin, *et al.*, "Preliminary investigation into data scaling laws for imitation learning-based end-to-end autonomous driving," *arXiv preprint arXiv:2412.02689*, 2024.

[166] OpenAI, *Learning to reason with llms*, `https://openai.com/index/learning-to-reason-with-llms/`, 2024.

[167] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[168] V. Xiang, C. Snell, K. Gandhi, A. Albalak, A. Singh, C. Blagden, D. Phung, R. Rafailov, N. Lile, D. Mahan, *et al.*, "Towards system 2 reasoning in llms: Learning how to think with meta chain-of-though," *arXiv preprint arXiv:2501.04682*, 2025.