UNIVERSITÄT BONN

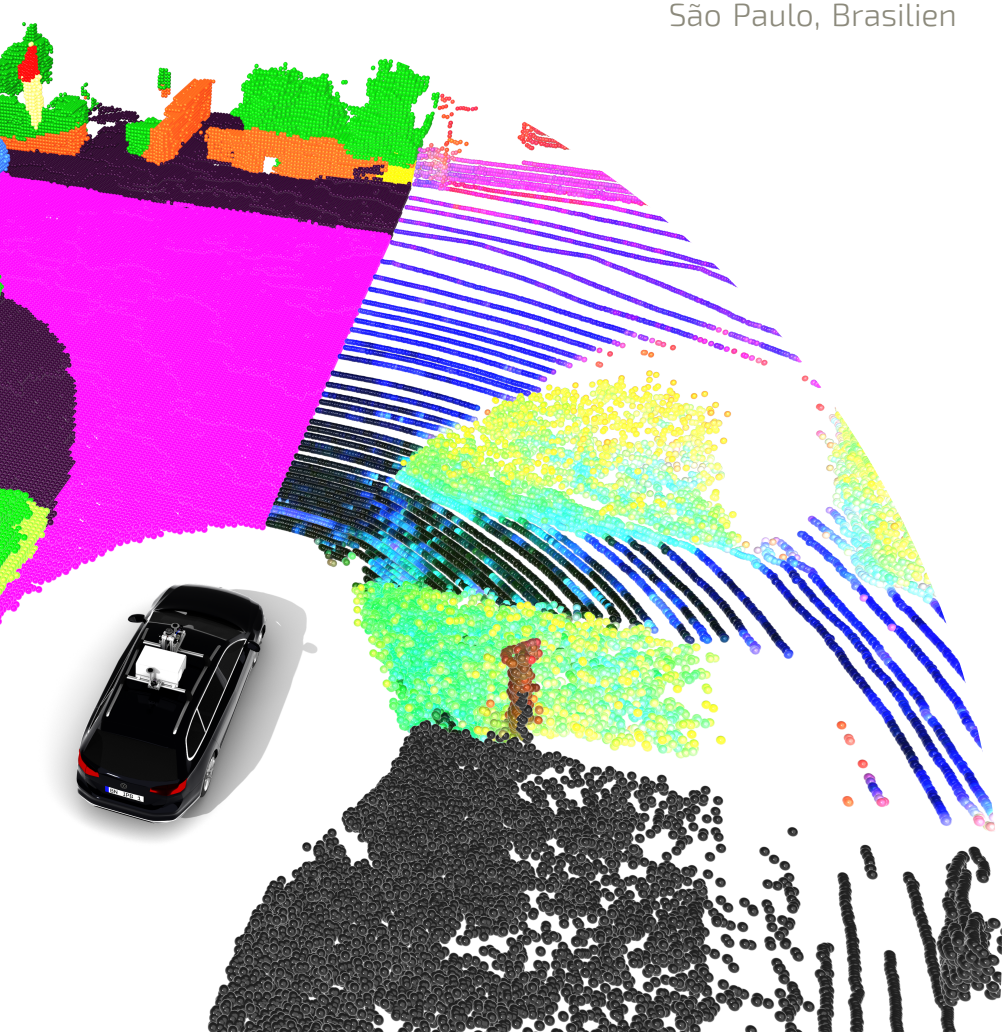
Dissertation
zur Erlangung des Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
Agrar-, Ernährungs- und Ingenieurwissenschaftliche Fakultät
der Rheinischen Friedrich-Wilhelms-Universität Bonn
Institut für Geodäsie und Geoinformation

Learning Discriminative Representations and Generative Approaches for Outdoor 3D LiDAR Data

von

Lucas Franco Nunes

aus
São Paulo, Brasilien



Bonn 2026

Referent:
Prof. Dr. Cyrill Stachniss, University of Bonn, Germany

1. Korreferent:
Prof. Dr. Abhinav Valada, University of Freiburg, Germany

2. Korreferent:
Prof. Dr. Ribana Roscher, University of Bonn, Germany

Tag der mündlichen Prüfung: 10.02.2026

Angefertigt mit Genehmigung der Agrar-, Ernährungs- und Ingenieurwissenschaftlichen Fakultät der Universität Bonn

Zusammenfassung

Mobile Roboter sind inzwischen Teil unseres Alltags. Autonome Systeme wurden entwickelt, um Aufgaben zu automatisieren, die für Menschen monoton, wiederholend oder gefährlich sind. Dennoch beschränken sich derzeit autonome Systeme auf spezifische, klar definierte Aufgaben in kontrollierten Umgebungen, wie das Reinigen von Häusern oder die Verwaltung von Lagerbeständen. Die Nutzung von autonomen Systemen in hochdynamischen und sich ständig ändernden Umgebungen, wie im autonomen Fahren, ist noch immer eine große Herausforderung. Um in diesen komplexen Umgebungen zu operieren, benötigen die Fahrzeuge robuste Wahrnehmungssysteme, um verschiedene Akteure wie Fußgänger, Radfahrer und Fahrzeuge zu erkennen und zu berücksichtigen. Fehlentscheidungen können katastrophale Folgen haben.

Eine zentrale Einschränkung für die Automatisierung solcher Aufgaben liegt in den Grenzen heutiger Wahrnehmungssysteme bei der Interpretation der Rohsensordaten der Fahrzeuge. Meist werden neuronale Netze eingesetzt, um aus Sensordaten, wie Punktwolken von 3D-LiDAR oder Bildern von Kameras, Informationen über die Fahrzeugumgebung zu gewinnen und Entscheidungen zu treffen. Das Training solcher Netze erfordert große, vielfältige Datensätze. Im autonomen Fahren betrifft dies unterschiedliche Objekte, den Einsatzort, Jahreszeiten, Wetterbedingungen und Lichtverhältnisse. Die dafür notwendigen Datenannotationen sind zeitaufwendig und teuer, vor allem bei 3D-LiDAR-Annotationen. Der manuelle Arbeitsaufwand lässt sich wegen der Komplexität und Datenmenge nur schwer skalieren. Es ist notwendig, die Systeme zu verbessern und zugleich den Bedarf an manuellen Annotationen zu senken.

Diese Arbeit unterteilt die Verbesserung von Wahrnehmungssystemen in drei Hauptprobleme in Bezug auf die Skalierbarkeit der Annotation. Die erste Herausforderung betrifft die Menge an annotierten Daten, sowohl in Quantität als auch in Vielfalt. Die zweite Herausforderung betrifft die Abdeckung der Klassen. In unstrukturierten Umgebungen ist es unmöglich, alle potenziellen Objektklassen zu definieren und zu annotieren. Das Verhalten des Systems gegenüber un-

gewöhnlichen Objekten wird im Test unvorhersehbar und kann gefährlich sein. Die dritte Herausforderung betrifft die Unterschiede zwischen Daten verschiedener Sensoren. Daten verschiedener LiDARs haben oft unterschiedliche Strukturen oder Auflösungen, was das Aussehen der Punktwolken stark verändert. Annotierte Punktwolken werden so sensorspezifisch und sind nicht generalisierbar.

Zur ersten Herausforderung schlagen wir neue, selbstüberwachte Trainingsmethoden vor. Wir optimieren ein Netz, das eine latente Repräsentation lernt, welche grobe Objekte in der Szene ohne Annotationen unterscheidet. Die Methode erweitern wir, indem wir Scan-Sequenzen nutzen und so eine Repräsentation ableiten, die auch gegenüber der Sensorperspektive invariant ist. Zudem setzen wir generative Modelle ein, um die Datenverteilung des annotierten Datensatzes zu lernen und neue gelabelte Punktwolken zu erzeugen. So steigt die Menge an Labels ohne manuelle Annotation. Beide Ansätze verbessern Wahrnehmungsaufgaben und reduzieren den Bedarf an Annotationen.

Für die zweite Herausforderung schlagen wir eine neue klassenunabhängige Instanzsegmentierung vor. Wir bauen einen Graphen aus der LiDAR-Punktwolke und gewichten die Kanten mit einem Netz, das mit unserer selbstüberwachten Strategie trainiert wurde, basierend auf punktweisen Ähnlichkeiten. Dann partitionieren wir den Graphen und trennen einzelne Instanzen klassenunabhängig. So kann das System alle Objekte erkennen, auch jene, die in den annotierten Klassen fehlen.

Für die dritte Herausforderung schlagen wir eine Methode vor, um aus einem einzelnen LiDAR-Scan eine dichte und vollständige Punktwolke vorherzusagen. Wir nutzen generative Modelle, um fehlende Regionen zu erzeugen und eine vollständige Szenenrepräsentation zu gewinnen. Die dichte Punktwolke dient als Proxy, aus dem sensorspezifische LiDAR-Scans erzeugt werden. Damit überbrücken wir die Domänenlücke und ermöglichen Transfer zwischen Sensoren.

Diese Arbeit behandelt verschiedene Herausforderungen, um Wahrnehmungssysteme bezüglich der Skalierbarkeit von Annotationen zu verbessern. Wir schlagen Methoden des Repräsentationslernens und der Generierung vor, um zuverlässige Systeme zu entwickeln und den Bedarf an Annotationen zu senken. Die Methoden dieser Arbeit wurden in begutachteten Zeitschriften und Konferenzen veröffentlicht, und ihre Implementierungen sind Open Source verfügbar, um weitere Forschung zu unterstützen.

Abstract

Mobile robots have become part of our everyday lives nowadays. Autonomous systems have been developed to automate tasks that are monotonous, repetitive, or dangerous to humans. Still, such automations are currently limited to specific and well-defined tasks in controlled environments, such as cleaning our houses or managing inventory in warehouses. The execution of tasks out of such controlled environments is limited due to the complexity of dealing with highly dynamic and constantly changing scenarios, as in the autonomous driving context. In such a scenario, the robot's perception system must be robust to deal with the different agents in the scene, such as pedestrians, cyclists, and other vehicles, constantly interacting. At the same time, wrong decisions can have catastrophic impacts.

A key limitation for automating tasks in contexts such as autonomous driving comes from the limitations of current perception systems in interpreting the raw sensor data onboard the vehicles. Commonly, neural networks are employed to extract information from sensor data about the vehicle's surroundings for decision-making. The training of these neural networks requires the dataset to be large in quantity and variability. In the autonomous driving context, this is related to different objects that may appear, the location of deployment, seasons of the year, weather, and light conditions. The requirement for data labeling becomes more challenging when considering 3D LiDAR data instead of images, given the sparsity of the point clouds, varying sensor resolutions, and the difficulty for humans to label such data compared to images. However, data annotation is not easily scalable due to its complexity and the amount of data required. Therefore, a more reliable approach towards developing perception systems is to focus on improving such systems while reducing the requirement for manual labeling.

This thesis divides the challenge of improving perception systems into three main challenges related to the scalability of data annotation. The first challenge is related to the amount of labeled data available to train perception models, accounting for quantity and variability. The second challenge is related to the coverage of classes defined in the annotated data. In unstructured environments, it is impossible to define and label all potentially occurring classes of objects

that might appear. The behavior of the perception system towards those unusual objects becomes unpredictable at test time, potentially leading to dangerous situations. Finally, the third challenge relates to the domain gap between data collected with different sensors. Different LiDARs often have different laser beam patterns or sensor resolutions, leading to drastic changes in the appearance of the collected scans. Such changes make the available labeled point clouds sensor-dependent, limiting the use of annotated data between different LiDARs.

To address the first challenge, in this thesis we propose novel self-supervised training methods. We optimize a network for learning a latent representation that can distinguish between coarse segments of objects in the scene without labels. We further extend this method by leveraging sequences of scans, deriving a representation also invariant to the sensor viewpoint. Additionally, we employ generative models to learn the data distribution of the annotated dataset and generate novel labeled point clouds, increasing the amount of labels without manual labeling. Both self-supervised training and labeled data generation lead to improvement on different perception tasks, reducing the requirement for annotated data.

Regarding the second challenge, we propose a new class-agnostic instance segmentation method in this thesis. We build a graph from the LiDAR point cloud and use a network trained with our self-supervised training strategy to weight the edges based on the computed point-wise feature similarities. We then partition this graph, separating individual instances in the point cloud in a class-agnostic manner. This method enables the perception system to identify all objects in the point cloud, including those not present in the annotated classes.

To address the third challenge, this thesis proposes a method to predict a dense and complete point cloud from a single LiDAR scan. We leverage generative models to generate the unseen regions in a sparse LiDAR measurement to achieve a complete scene representation. The dense scene point cloud serves as a proxy from which sensor-specific LiDAR scans can be sampled. This allows us to bridge the domain gap between different LiDARs, enabling transfer across sensors.

This thesis tackles different challenges to improve perception systems regarding data annotation scalability. We proposed different representation learning and generative methods to advance the development of reliable perception systems while reducing the requirement for data annotation. The methods presented in this thesis were published in peer-reviewed journals and conferences, and their implementations are open-source to support further research in the field.

Acknowledgements

EARNING a Ph.D. is more than just a degree, it is a journey of discovering new research directions, studying and developing innovative ideas, but also self-discovery and personal growth. As with any long journey, there will be for sure ups and downs, and often it may be exhausting or even seem impossible to accomplish. Gladly for me, I have not travelled alone during these five years.

First and foremost, I would like to thank my doctoral supervisor, Cyrill Stachniss, for giving me the opportunity to pursue my Ph.D. as one of his students. At the beginning of a Ph.D., one must first learn how to do research and, equally important, how to present it. Under his supervision, he taught me how to do both. I had the space to explore my own ideas, and whenever needed, he would arrange time for discussion and help with any doubts that I had. Other than research, I am also grateful for his guidance and mentorship regarding my academic path, also giving me advices about my next endeavors.

Next, I would like to thank Jens Behley, the post-doc who guided me during my Ph.D. years. Whenever I wanted to discuss a new research idea or when I had no idea why something wasn't working, he was there to "*provide knowledge and support*". I am also thankful for the long discussions about mathematical notations to make my papers clearer, which always led to drastic changes everywhere in the text, but it was worth it in the end.

I also want to thank my supervisors during my master's, Denis Fernando Wolf and Jefferson Rodrigo de Souza, who motivated and inspired me during my first contact with research. They showed me how working with research can be both challenging and at the same time rewarding.

Pursuing a Ph.D. often leads to periods of uncertainty and stress, which can be overwhelming without support. I had the pleasure of sharing the lab with many great people. I am glad to have worked closely with Rodrigo. It was great to share the office with him and often exchange times of motivation and excitement when something was working, as well as share the times when nothing was working as expected. Also, I am glad to have worked closely with Nacho. Apart from the huge technical knowledge and support Nacho provided, it was

always great to talk with him and Rodrigo and have a bit of the warmth from South America. It was great to have a piece of home close by, *gracias* for that.

I am also thankful to have worked with many close friends during this time. I am glad to have worked with Louis. It was challenging to discuss research with him simply because we would end up making jokes and laughing about nonsense, but it was always a reliable source of distress. Thanks for that. I also want to thank Elias and Matteo for providing valuable Italian culinary knowledge and, most importantly, long coffee breaks after lunch whenever needed, and sometimes it was much needed. Also, I am glad to have shared the office with Gianmarco and appreciate the random morning hugs, especially during deadlines those were really helpful. I also want to thank Linn for the exchange of knowledge about cats and plants, from which my chili plants manage to survive and even flourish.

I am also grateful to have worked with and learned from many other colleagues: Nived Chebrolu, Xieyuanli (Rhiney) Chen, Tiziano Guadagnino, Saurabh Gupta, Liren Jin, Federico Magistri, Haofei Kuang, Luca Lobefaro, Meher Malladi, Benedikt Mersch, Yue Pan, Lasse Peters, Julius Rückin, Niklas Trekel, Jan Weyler, Matthias Zeller, Xingguang (Starry) Zhong, Nicky Zimmerman, and Daniel Casado. I am grateful for the friendly relationship and the time we spent together. I would also like to thank Thomas Läbe, Perrine Aguiar, Birgit Klein and Kirsten Sadler for their technical and administrative support. As an expat, it's quite challenging to deal with the German bureaucracy. Thanks for dealing and helping me with that.

Moving to a new country, a new continent, far from everyone you know is a hard decision. Taking this decision to pursue my dream and handling this huge change in my life would not have been possible without the support of my wife, Fernanda Rani Malta. Thank you for supporting me always, from the decision to move to Germany in the middle of a pandemic, and throughout my Ph.D. years. Thank you for being my biggest fan, pushing me forward whenever I doubted myself, and being my partner in life. I am deeply grateful for sharing this chapter of my life with you and eager to share the chapters yet to come.

Despite being in a new country and making new friends, I am glad to have been able to receive support remotely from my friends in Brazil. I want to thank Caio, Denise, Diego, Kennedy, Lucas, Renan, and Kevin for sparing weekend hours with me, either playing something together online or just talking about everyday life over text messages. The lack of friends is something I cannot complain about.

I would also like to thank my family for their continuous support in following my dream. I would like to thank my parents, Emanuel and Damaris, for always supporting me and providing everything they could to help me follow my dreams. I would also like to thank my sister Agatha for being not only an example but also for motivating me to pursue my goals. I am also grateful for having Agatha

and her wife Thamires moving closer by. Living far from family and friends can be challenging, and I am glad to have you two living close. I'm grateful for the trips and weekends we spend together. Thanks my family for supporting and encouraging me to dare to dream and helping me to achieve it.

A common saying in academic life is that research is only possible by "standing on the shoulders of giants". Now, at the end of my Ph.D. I could not agree more, and I am deeply grateful for the ones who have supported me until here.

The work presented in this thesis is partially supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy, EXC-2070 – 390732324 – PhenoRob, by the European Union's Horizon 2020 research and innovation programme under grant agreement No 101017008 (Harmony), and by the German Federal Ministry of Research, Technology and Space (BMFTR) under the Robotics Institute Germany (RIG).

Contents

Zusammenfassung **iii**

Abstract **v**

Contents **xi**

1 Introduction **1**
1.1 Main Contributions 4
1.2 Publications 6
1.3 Further Scientific Contributions 7
1.4 Open Source Contributions 9

2 Basic Techniques **11**
2.1 Sparse Convolutional Neural Networks 12
2.1.1 2D Convolutional Neural Network 12
2.1.2 3D Convolutional Neural Networks 15
2.1.3 Sparse 3D Convolutions 16
2.1.4 MinkUNet 18
2.2 Self-Supervised Pre-Training with Contrastive Learning 19
2.3 Autoencoder Networks 21
2.3.1 Autoencoders 22
2.3.2 Variational Autoencoders 22

3 Representation Learning from Unlabeled 3D Data **25**
3.1 3D LiDAR Data Representation Learning 27
3.1.1 Our Approach 28
3.1.2 Unsupervised Segment Extraction 29
3.1.3 Segment Augmentation 29
3.1.4 Segment Contrastive Loss 30
3.1.5 Pre-Training Strategy 31
3.2 Experimental Evaluation 32
3.2.1 Implementation Details and Experimental Setup 32

3.2.2 Fine-Tuning for Semantic Segmentation . . . 33
3.2.2.1 Label Efficiency . . . 33
3.2.2.2 Linear Evaluation . . . 35
3.2.2.3 Generalization . . . 35
3.2.3 Fine-Tuning for Object Detection . . . 39
3.3 Related Work . . . 39
3.4 Conclusion . . . 42

4 Temporal Representations from Unlabeled 3D Data 45
4.1 Temporal Consistent 3D LiDAR Representation Learning . . . 47
4.1.1 Our Approach . . . 47
4.1.2 Temporal Objects Views . . . 48
4.1.3 Temporal Batch . . . 50
4.1.4 Implicit Clustering . . . 51
4.1.5 Transformer Projection . . . 53
4.1.6 Pre-Training Strategy . . . 53
4.2 Experimental Evaluation . . . 54
4.2.1 Implementation Details and Experimental Setup . . . 54
4.2.2 Fine-Tuning for Semantic Segmentation . . . 55
4.2.2.1 Label Efficiency . . . 55
4.2.2.2 Linear Evaluation . . . 56
4.2.2.3 Generalization . . . 57
4.2.3 Fine-Tuning for Panoptic Segmentation . . . 57
4.2.3.1 Label Efficiency . . . 57
4.2.3.2 Generalization . . . 58
4.2.4 Fine-Tuning for Object Detection . . . 60
4.2.5 Ablation Studies . . . 60
4.2.5.1 Temporal Views . . . 62
4.2.5.2 Transformer Projection Head . . . 62
4.2.5.3 Different Number of Aggregated Scans . . . 63
4.3 Related Work . . . 63
4.4 Conclusion . . . 65

5 Unsupervised Instance Segmentation in 3D Point Clouds 67
5.1 Unsupervised Class-Agnostic 3D Instance Segmentation . . . 69
5.1.1 Our Approach . . . 70
5.1.2 Instance Proposals . . . 71
5.1.3 Self-Supervised Features . . . 72
5.1.4 Instance Cues . . . 72
5.1.5 Seeds Sampling . . . 73
5.1.6 GraphCut . . . 74

5.1.6.1 Terminal Edges . . . 74
5.1.6.2 Non-Terminal Edges . . . 75
5.1.7 Class-Agnostic Instance Segmentation Pipeline . . . 75
5.2 Open-World LiDAR Instance Segmentation Benchmark . . . 76
5.2.1 Problem Setting . . . 76
5.2.2 Benchmark Evaluation Metrics . . . 76
5.2.3 Open-World SemanticKITTI Dataset and Benchmark . . . 77
5.3 Experimental Evaluation . . . 78
5.3.1 Implementation Details and Experimental Setup . . . 79
5.3.2 Association Quality . . . 80
5.3.3 Instance Segmentation Quality . . . 81
5.3.4 Ablation Studies . . . 83
5.4 Related Work . . . 84
5.5 Conclusion . . . 86

6 Generative 3D Scene Completion 89
6.1 Scaling Diffusion Models to 3D LiDAR Scene Completion . . . 90
6.1.1 Our Approach . . . 91
6.1.2 Denoising Diffusion Probabilistic Models . . . 91
6.1.3 Diffusion Scene Completion . . . 93
6.1.4 Local Point Denoising . . . 94
6.1.5 Noise Prediction Regularization . . . 95
6.1.6 Refinement Network . . . 96
6.1.7 Noise Predictor Architecture . . . 96
6.2 Experimental Evaluation . . . 97
6.2.1 Implementation Details and Experimental Setup . . . 97
6.2.2 Scene Reconstruction . . . 100
6.2.3 Scene Occupancy . . . 101
6.2.4 Noise Regularization . . . 104
6.2.5 Condition Weights . . . 105
6.3 Related Work . . . 107
6.4 Conclusion . . . 109

7 Generating 3D Semantically Annotated Training Data 111
7.1 Generating Realistic 3D Semantic Training Data . . . 113
7.1.1 Our Approach . . . 113
7.1.2 Semantic Scene Variational Autoencoder . . . 114
7.1.2.1 Pruning Loss . . . 115
7.1.2.2 Semantic Loss . . . 116
7.1.2.3 Latent Loss . . . 116
7.1.2.4 VAE training . . . 117

7.1.3 Semantic Scene Latent Diffusion . . . 117
7.1.3.1 Diffusion Training . . . 117
7.2 Experimental Evaluation . . . 119
7.2.1 Implementation Details and Experimental Setup . . . 119
7.2.2 Generated Scene Realism . . . 122
7.2.3 Generated Labels as Training Data . . . 126
7.2.4 Synthetic Training Set Extension . . . 127
7.2.5 Conditional DDPM for Data Annotation . . . 130
7.2.6 Generating Large-Scale Scenarios . . . 132
7.2.7 Generated and Real Data Gaps . . . 133
7.3 Related Work . . . 135
7.4 Conclusion . . . 138

8 Conclusion **141**
8.1 Summary of the Key Contributions . . . 142
8.2 Future Work . . . 145

Acronyms

CD Chamfer distance

CNN convolutional neural network

DDPM denoising diffusion probabilistic model

IoU intersection-over-union

JSD Jensen-Shannon divergence

LiDAR light detection and ranging

mIoU mean intersection-over-union

MLP multilayer perceptron

MMD maximum mean discrepancy

MSE mean squared error

PQ panoptic quality

SDF signed distance field

VAE variational autoencoder

Chapter 1

Introduction

Robots are nowadays increasingly present in our lives and autonomous systems have been developed to take over dangerous, monotonous, or time-consuming tasks. To achieve these tasks, the robot is equipped with various sensors, such as cameras, radars, and 3D light detection and ranging (LiDAR) sensors. With the measurements collected by those sensors, the robot must perceive and understand its surroundings to plan how to interact with it, and finally execute the planned actions. It establishes a continuous loop between perceiving, planning, and acting. Ensuring robustness across all three capabilities enables the robot to operate autonomously in the real world.

Despite the promise of automating complex tasks in our everyday lives, robotic systems are currently limited to specific and often simple or well-defined tasks in controlled environments, such as cleaning our homes or managing inventory in warehouses. Those applications are achieved by restricting and simplifying the interaction between the robot and its surroundings, requiring only a limited form of autonomy. One of the core challenges in deploying mobile robots for more complex applications, such as autonomous driving or disaster response, is the limitation of state-of-the-art perception systems. The perception system must be reliable and robust in complex and constantly changing real-world scenarios to enable the deployment of robots out of controlled environments. Semantic information about the robots' surroundings is necessary in this case, for example, identifying the types of objects present in the scene, or the drivable surfaces for the robot to plan and act accordingly. Without enough semantic information, the robot cannot safely interact with the objects in open scenarios, limiting the automation of tasks in such unstructured environments.

One application that has been in the spotlight in recent years is autonomous driving. In this context, different agents such as pedestrians, cyclists, cars, and other vehicles are constantly interacting with each other. Furthermore, driving happens in the real world and the environment can only be controlled partially.

Figure 1.1: Examples of challenging scenarios in the autonomous driving context, collected at different seasons of the year, with objects "unusual" in this context. In the image, we can see kids running, adapted bikes, and a tractor. Such variability of scenarios demand the perception system to deal with complex situations as well as with "unusual" objects, potentially not labeled in the training dataset. Images from the PANIC dataset [162].

The driver has to identify the various agents and their behavior to plan and act accordingly to drive through the environment safely. In such a scenario, errors can lead to accidents and potentially impose risks to human lives, given the velocity of the vehicles. When considering non-autonomous vehicles, those accidents are usually caused by human errors due to fatigue or distractions while driving.

Autonomous systems, distinct from humans, are not prone to fatigue or distractions. Thus, replacing the human driver with autonomous systems has been the focus of recent research. Nowadays, several vehicle manufacturers employ driver assistance systems to identify potentially hazardous situations, notifying the driver or even taking actions to avoid potential accidents, such as braking the vehicle or keeping a distance from other cars. In such cases, the robotics applications are mainly assistive, without being completely autonomous due to the limitations of current perception systems to cope with the complexity of the scenario. Still, the development of completely autonomous vehicles often referred to as Level 5 autonomy has been the focus of much research, since it has the potential to increase safety, improve mobility and traffic flow in our cities, as well as directly impact transportation and logistics by automating those sectors.

An autonomous vehicle must deal with the complex interactions of various objects and agents that constantly occur in the environment. The formulation of such complex and dynamic agents' characteristics and interactions cannot be easily done heuristically. Therefore, learning-based methods such as convolutional neural networks are a popular approach to enable perception systems to perceive

and interpret the vehicle's surroundings via different semantic perception tasks. Such neural networks have to be trained on data collected in the target domain and annotated for the specific target task. After training, the network can be used to perform the target perception task, predicting semantic information from the data collected by the vehicle's sensors.

However, those perception models are still not reliable enough due to the complexity of providing sufficient labeled data to account for all possible situations that can occur in this unstructured driving scenario. For a model to provide a reliable prediction in such challenging situations, it's training dataset must be large in quantity and variability to account for the diversity of scenarios in the real world. In autonomous driving, this diversity is related to many factors, such as different seasons of the year, traffic scenarios, light conditions, and "unusual" objects that can occur, which are not common in this context. Fig. 1.1 illustrates some examples of challenging scenarios in urban environments. Such scenarios or unexpected objects not present in the training data may occur, making it difficult for the autonomous system to define a proper action towards them. Compared to image data, this requirement for data annotation becomes even more challenging when considering 3D LiDAR data due to the sparsity of the point clouds, changes in objects' representation with respect to the sensor viewpoint, and varying resolutions between different LiDAR sensors. This imposes a hard constraint on standard supervised trained neural networks for 3D data, as the data annotation scalability cannot keep up with the data demand for reliable perception systems. Therefore, an alternative approach to the development of perception systems can be to reduce the reliance on manual labeling, loosening the dependency of perception systems on data annotation.

In this thesis, we divide the challenge of developing reliable 3D perception systems into three central challenges related to the data annotation scalability. These are the amount of labeled data available, the coverage of the classes defined in the dataset, and the domain gap between data collected with different LiDAR sensors. In the following chapters, we tackle all three challenges as depicted in Fig. 1.2, pushing the state of the art forward. In Chapter 3, we propose a method to train a neural network to learn a descriptive latent representation from unlabeled 3D LiDAR data that can be fine-tuned to different perception tasks, improving the model performance without requiring more annotated data. Chapter 4 extends this method by leveraging sequential LiDAR data, learning a representation that is invariant to the LiDAR viewpoint, and reducing the perception model requirements for annotated data. In Chapter 5, we address the issue regarding the number of labeled classes by proposing a class-agnostic instance segmentation method, which can identify all objects in the scene independently of their class. In Chapter 6, we deal with the LiDAR sensors domain gap by using

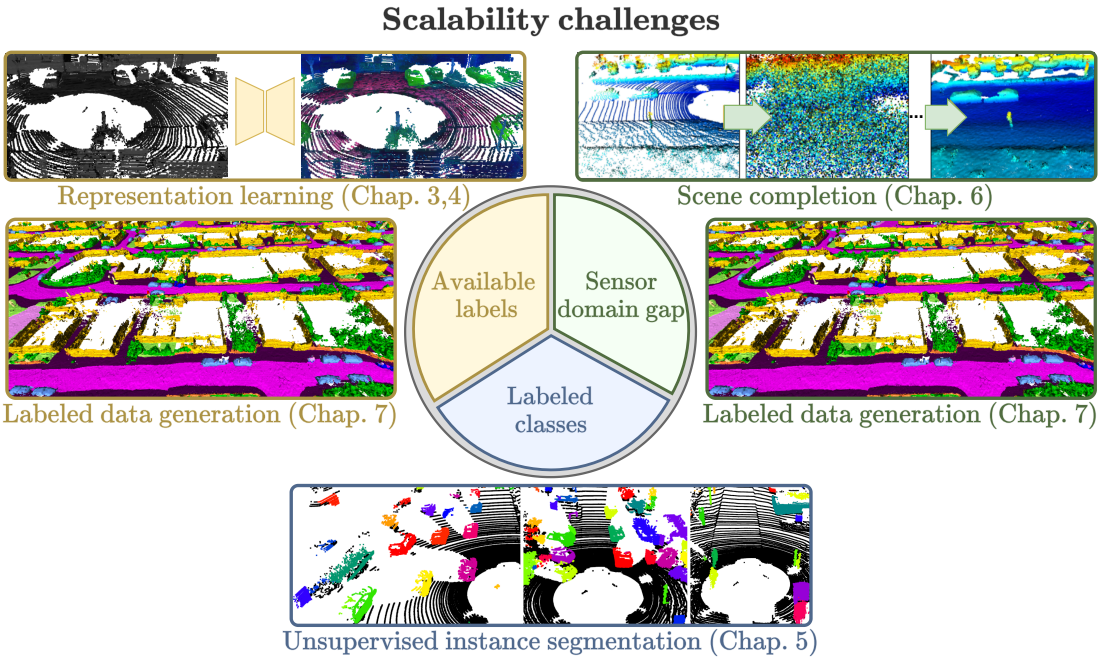


Figure 1.2: Diagram with the challenges to improve the performance of perception systems and with the thesis contributions for each of the three challenges. Chapters 3, 4, and 7 tackle the problem regarding the amount of labeled data available. Chapter 5 deals with the problem of labeling all possible classes of objects that can occur in the real-world, which are potentially infinite. Chapters 6 and 7 deal with the challenge of sensor-specific characteristics in the point clouds measured by different LiDARs.

generative models to generate a dense point cloud from a LiDAR scan, thereby achieving a dense and sensor-independent scene representation, which can serve as a proxy for sampling sensor-specific LiDAR point clouds. As the last technical contribution Chapter 7 presents our method for generating novel, dense, and semantically annotated 3D scenes. This enables sampling sensor-specific annotated LiDAR point clouds from the generated dense scenes, improving semantic segmentation performance by training the model with both real and synthetic annotated data. Finally, Chapter 8 summarizes our contributions and discusses future work that can be built upon the methods proposed in this thesis.

1.1 Main Contributions

This thesis investigates the challenge of enhancing perception systems without relying on extensive data annotation. We start by learning descriptive data representations with neural networks without requiring labels, and then leverage those representations to achieve a segmentation of objects without label supervision. Finally, we utilize recent generative models to generate additional annotated data based on existing datasets, improving perception systems without requiring fur-

ther data annotation. This section summarizes the key contributions of this thesis discussed in the individual chapters.

The *first contribution* aims at learning object-centric representations from unlabeled point clouds. The data annotation scalability is a limiting factor to the generalizability of perception systems' performance. In Chapter 3, we employ basic clustering and ground segmentation techniques to define coarse segments of objects in a point cloud. Then, we train a neural network to learn a representation that can discriminate between different objects in the scene. This pre-trained network serves as a foundational general representation that we can fine-tune to specific downstream tasks, improving the performance in those tasks without requiring further data annotation.

Our *second contribution* leverages sequences of LiDAR scans to learn a representation that is invariant to the object changes with respect to the sensor viewpoint, discussed in Chapter 4. We extract temporal coarse segments of objects by aggregating scans given their relative poses. Such segments are used to train the neural network to extract similar features from the same object seen from different viewpoints from the LiDAR via an implicit clustering scheme. This representation leads to an even larger improvement on perception tasks, effectively reducing the amount of labeled data required by the perception model. Our main contribution is a temporally consistent representation learning approach and an implicit clustering scheme, which decouples the development of perception systems from the scaling of data annotation.

Our *third contribution* described in Chapter 5 leverages the representation learned by the model trained without labels to embed object-level information in a point cloud, enabling unsupervised object segmentation. Due to the complexity of real-world scenarios, labeling all possible objects in the world within a finite number of classes is intractable. We use the network trained with our proposed representation learning method to compute features for each point in the LiDAR scan. Then, those features are used to build a graph representation of the point cloud, instantiating the objects in the scene by partitioning this graph. Our contribution relies on leveraging a representation learning method and a graph-based approach to achieve class-agnostic instance segmentation in LiDAR scans.

Given the complexity of identifying all objects in the scene independently of their classes, our *fourth contribution* is a benchmark for open-world LiDAR instance segmentation. We extend the SemanticKITTI dataset [5, 54] by annotating objects that were previously left unlabeled, providing a set of *unknown* instances. This benchmark provides a set of objects usually not labeled in autonomous driving datasets, providing a test bed for open-world instance segmentation methods to support further research in this field.

As the *fifth contribution*, we tackle the challenge of dealing with different

LiDAR sensors and the differences between their collected point clouds in Chapter 6. Besides the challenge of scaling data annotation, labeled data from one LiDAR sensor may not be directly applicable to a different LiDAR sensor due to sensor-specific characteristics. Chapter 6 discusses our method to achieve scene completion from single LiDAR point clouds. We employ state-of-the-art diffusion models to generate dense and complete point clouds conditioned on a LiDAR scan. It allows for generating the regions not seen by the LiDAR measurements, computing a dense and complete point cloud independent of the LiDAR sensor used during data collection. This dense scene representation enables the sampling of sensor-specific point clouds, bridging the gap between different LiDAR sensors. Our main contribution is a point-wise diffusion formulation, allowing the scalability of point diffusion models from object scale to scene scale.

The *sixth and last contribution* addresses both the scalability of data annotation and the differences between data collected with different LiDARs by generating annotated, dense scenes that we can use to train perception models in Chapter 7. Although self-supervised pre-training demonstrated to improve the performance on perception tasks in Chapters 3 and 4, training a neural network with more annotated data remains valuable. To tackle this, we use a variational autoencoder to learn a latent representation from dense and semantically annotated scenes by aggregating consecutive scans of available annotated datasets. Then, we employ a latent diffusion model to generate novel, dense, and annotated samples from the target dataset, which we can use to train sensor-specific models by sampling specific LiDAR configurations from the dense scenes. Our contribution is a method to generate close-to-real semantically annotated scenes, which we can use to train perception models and improving their performance, scaling the availability of annotated data without having to scale data annotation.

This thesis contributes to various areas of the robotics and computer vision communities. From learning descriptive representations from unlabeled 3D data and employing those representations to achieve specific perception tasks, to state-of-the-art generative methods for 3D scene-scale data. Besides the key claims and contributions in the individual chapters, the contributions in this thesis are not limited to the specific experiments and applications presented in its chapters. As discussed in Chapter 8, the contributions in this thesis can be explored and extended to other applications and domains in future work, setting a foundation for further research and advances in robotics and computer vision.

1.2 Publications

Most parts of this thesis have been published in the following peer-reviewed conference and journal articles:

- 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 (RA-L)*, 7(2):2116–2123, 2022. DOI: 10.1109/LRA.2022.3142440

- L. Nunes, X. Chen, R. Marcuzzi, A. Osep, L. Leal-Taixé, C. Stachniss, and J. Behley. Unsupervised Class-Agnostic Instance Segmentation of 3D LiDAR Data for Autonomous Vehicles. *IEEE Robotics and Automation Letters (RA-L)*, 7(4):8713–8720, 2022. DOI: 10.1109/LRA.2022.3187872

- L. Nunes, L. Wiesmann, R. Marcuzzi, X. Chen, J. Behley, and C. Stachniss. Temporal Consistent 3D LiDAR Representation Learning for Semantic Perception in Autonomous Driving. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023. DOI: 10.1109/CVPR52729.2023.00505

- L. Nunes, R. Marcuzzi, B. Mersch, J. Behley, and C. Stachniss. Scaling Diffusion Models to Real-World 3D LiDAR Scene Completion. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. DOI: 10.1109/CVPR52733.2024.01399

- L. Nunes, R. Marcuzzi, J. Behley, and C. Stachniss. Towards Generating Realistic 3D Semantic Training Data for Autonomous Driving. *arXiv preprint*, arXiv:2503.21449, 2025. DOI: 10.48550/arXiv.2503.21449 (resubmitted with minor revision)

1.3 Further Scientific Contributions

During my doctorate, I was also involved as a co-author in the following peer-reviewed conference and journal publications which are not part of the thesis:

- R. Marcuzzi, L. Nunes, L. Wiesmann, I. Vizzo, J. Behley, and C. Stachniss. Contrastive Instance Association for 4D Panoptic Segmentation using Sequences of 3D LiDAR Scans. *IEEE Robotics and Automation Letters (RA-L)*, 7(2):1550–1557, 2022. DOI: 10.1109/LRA.2022.3140439

- X. Chen, B. Mersch, L. Nunes, R. Marcuzzi, I. Vizzo, J. Behley, and C. Stachniss. Automatic Labeling to Generate Training Data for Online LiDAR-Based Moving Object Segmentation. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):6107–6114, 2022. DOI: 10.1109/LRA.2022.3166544

- B. Mersch, X. Chen, I. Vizzo, L. Nunes, J. Behley, and C. Stachniss. Receding Moving Object Segmentation in 3D LiDAR Data Using Sparse 4D

Convolutions. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):7503–7510, 2022. DOI: 10.1109/LRA.2022.3183245

- L. Wiesmann, L. Nunes, J. Behley, and C. Stachniss. KPPR: Exploiting Momentum Contrast for Point Cloud-Based Place Recognition. *IEEE Robotics and Automation Letters (RA-L)*, 8(2):592–599, 2023. DOI: 10.1109/LRA.2022.3228174

- R. Marcuzzi, L. Nunes, L. Wiesmann, J. Behley, and C. Stachniss. Mask-Based Panoptic LiDAR Segmentation for Autonomous Driving. *IEEE Robotics and Automation Letters (RA-L)*, 8(2):1141–1148, 2023. DOI: 10.1109/LRA.2023.3236568

- H. Lim, L. Nunes, B. Mersch, X. Chen, J. Behley, H. Myung, and C. Stachniss. ERASOR2: Instance-Aware Robust 3D Mapping of the Static World in Dynamic Scenes. In *Proc. of Robotics: Science and Systems (RSS)*, 2023. DOI: 10.15607/RSS.2023.XIX.067

- I. Vizzo, B. Mersch, L. Nunes, L. Wiesmann, T. Guadagnino, and C. Stachniss. Toward Reproducible Version-Controlled Perception Platforms: Embracing Simplicity in Autonomous Vehicle Dataset Acquisition. In *Proc. of the Intl. Conf. on Intelligent Transportation Systems Workshops*, 2023. DOI: 10.1109/ITSC57777.2023.10421988

- R. Marcuzzi, L. Nunes, L. Wiesmann, E. Marks, J. Behley, and C. Stachniss. Mask4D: End-to-End Mask-Based 4D Panoptic Segmentation for LiDAR Sequences. *IEEE Robotics and Automation Letters (RA-L)*, 8(11):7487–7494, 2023. DOI: 10.1109/LRA.2023.3320020

- M. Sodano, F. Magistri, L. Nunes, J. Behley, and C. Stachniss. Open-World Semantic Segmentation Including Class Similarity. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024. DOI: 10.1109/CVPR52733.2024.00307

- L. Wiesmann, T. Läbe, L. Nunes, J. Behley, and C. Stachniss. Joint Intrinsic and Extrinsic Calibration of Perception Systems Utilizing a Calibration Environment. *IEEE Robotics and Automation Letters (RA-L)*, 9(10):9103–9110, 2024. DOI: 10.1109/LRA.2024.3457385

- R. Marcuzzi, L. Nunes, E. Marks, L. Wiesmann, T. Läbe, J. Behley, and C. Stachniss. SfmOcc: Vision-Based 3D Semantic Occupancy Prediction in Urban Environments. *IEEE Robotics and Automation Letters (RA-L)*, 10(5):5074–5081, 2025. DOI: 10.1109/LRA.2025.3557227

- Y. Chong, L. Nunes, F. Magistri, X. Zhong, J. Behley, and C. Stachniss. Zero-Shot Semantic Segmentation for Robots in Agriculture. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2025. DOI: 10.1109/IROS60139.2025.11246227

- E. Marks, L. Nunes, F. Magistri, M. Sodano, R. Marcuzzi, L. Zimmermann, J. Behley, and C. Stachniss. Tree Skeletonization from 3D Point Clouds by Denoising Diffusion. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2025.

1.4 Open Source Contributions

To support further research and the reproducibility of results, I have released all the main contributions made in this thesis as open-source software:

- **SegContrast**: Representation learning for 3D LiDAR data
  `https://github.com/PRBonn/segcontrast`

- **TARL**: Temporally consistent representation learning for 3D LiDAR data
  `https://github.com/PRBonn/TARL`

- **3DUIS**: Class-agnostic unsupervised 3D instance segmentation
  `https://github.com/PRBonn/3DUIS`

- **LiDiff**: Point diffusion for 3D LiDAR scene completion
  `https://github.com/PRBonn/LiDiff`

- **3DiSS**: Diffusion model for generating 3D point cloud with semantic labels
  `https://github.com/PRBonn/3DiSS`

Chapter 2

Basic Techniques

In the introduction, we discussed the importance of learning-based approaches for enabling robust perception systems to be deployed in the real world. The following chapters discuss the proposed methods, which employ learning-based techniques to enhance the performance of perception systems while reducing the requirement for labeled data, thereby overcoming the challenge of data annotation scalability. The contributions of this thesis rely on deep network architectures trained with 3D point cloud data, optimizing them to learn useful representations that can be used by the perception system, either for discriminative or generative tasks. In this chapter, we discuss the basic techniques used throughout the subsequent chapters, which are essential for the understanding of the contributions of this thesis.

Essentially, a deep convolutional neural network (CNN) architecture consists of consecutive layers, with a set of parameters that can be optimized to achieve a specific task. This optimization is usually done based on a training dataset composed of paired input and target output. After several training iterations, this deep network model can be used to perform the task for which it was trained. Other samples not seen by the network during training are given as input, from which the network will provide a prediction for the target task. Such network models are employed to achieve various fine-grained perception tasks, such as semantic segmentation [174] or object detection [66]. To achieve this, the network parameters are optimized to map the input data to a feature space descriptive enough to accomplish the target task. Therefore, a network is essentially a feature extractor optimized for a specific perception task given the annotated data.

The amount of training data used during optimization is also related to the quality of the features extracted. If trained with few samples, the network parameters lose the capability of defining a useful feature space, instead memorizing the paired input and output to minimize the loss function used during optimization. In contrast, if trained with a vast amount of data, the network parameters would

define a more complex and descriptive model to compute the correct predictions for the samples in this large dataset [209]. It establishes a relation between data annotation scalability and the performance of perception systems.

The challenge of improving the performance of neural networks without scaling data annotation can be addressed in two ways: either by training the network without requiring annotated data, or by generating more annotated samples from the target data distribution. The former has been studied by defining relatively simple pretext tasks from which labels can be automatically generated, training the network with those automatically generated labels. The latter has been more recently studied first to learn the distribution of the target domain and then use generative models to generate samples from the learned data distribution.

In this chapter, we discuss the foundations of deep convolutional architectures for 3D data processing and the self-supervised learning strategies used throughout this thesis. Sec. 2.1 defines CNNs and their convolutional operations, followed by the definition of the sparse operations to process sparse 3D point cloud data. Sec. 2.2 discusses recent contrastive strategies for self-supervised learning to train a network without requiring labels. Finally, Sec. 2.3.1 discusses autoencoder networks, which are used to map a training data distribution into a latent and more compact representation, enabling the use of generative models to generate novel samples from this latent representation, which can be decoded into new samples from the training data distribution.

2.1 Sparse Convolutional Neural Networks

In computer vision, CNNs [86] are one of the standard architectures for machine learning for image interpretation. Such networks are trained to extract features from the input data for a specific perception task. In this part, we discuss how the convolutional blocks inside the CNN architectures process the input data and extract features from it. We first discuss the basics of the convolution operation for 2D image data and explain how 2D convolutions can be extended to operate on 3D data. Finally, we discuss the sparsity of 3D point clouds and how this sparsity is exploited to implement efficient sparse convolutions for 3D data processing.

2.1.1 2D Convolutional Neural Network

In image processing, convolutions refer to the operation of applying a filter to an image. Those filters are manually designed as kernels and applied locally around a pixel neighborhood. For example, to reduce noise from an intensity image $\mathsf{I} \in \mathbb{R}^{U \times V}$, with $(u, v) \in \mathbb{Z}^2$ as the pixel location, a Gaussian blur kernel with kernel size K is defined as a matrix $\mathsf{W} \in \mathbb{R}^{K \times K}$. The kernel W is applied

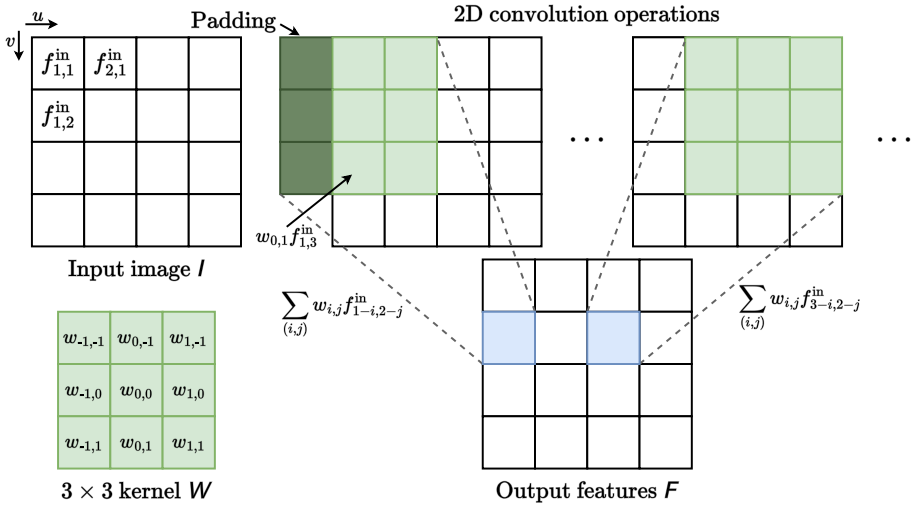


Figure 2.1: Example of a 2D convolution over an input image I. The kernel W is applied to all pixels $f^{\text{in}}_{u,v}$ in the image I, computing the output feature image F.

over the input image pixels $f^{\text{in}}_{u,v}$ to reduce the noise in the image by averaging the neighboring pixels of $f^{\text{in}}_{u,v}$, given the filter kernel weights $w_{i,j}$, computing the output pixel value $f^{\text{out}}_{u,v} \in \mathbb{R}$ as:

$$f^{\text{out}}_{u,v} = \sum_{(i,j)\in\mathcal{V}^2(W)} w_{i,j} f^{\text{in}}_{u-i,v-j}, \tag{2.1}$$

where $\mathcal{V}^2(W)$ denotes the set of offsets representing the neighborhood around the pixel (u, v). For a kernel size $K = 3$, i.e., $W \in \mathbb{R}^{3\times3}$, the offsets would be:

$$\mathcal{V}^2(W) = \{(-1,-1), (-1,0), (-1,1), (0,-1), (0,0), (0,1), (1,-1), (1,0), (1,1)\}. \tag{2.2}$$

The smoothed output image $F \in \mathbb{R}^{U\times V}$ is then computed by applying the kernel W to all pixels $f^{\text{in}}_{u,v}$ using Eq. (2.1). In the image border pixels, i.e., $u \in \{1, U\}$ and $v \in \{1, V\}$, the image is typically padded with zeros to enable the computation of output features over the border pixels. Fig. 2.1 shows the computation of the output feature image F given the kernel W.

This convolution operation can not only be applied to gray-scale but also to RGB images, where each pixel is represented with three channels instead of a single channel, as for the intensity images. In this case, in an image $I \in \mathbb{R}^{U\times V\times3}$, each pixel corresponds to a vector $\boldsymbol{f}^{\text{in}}_{u,v} \in \mathbb{R}^3$. Therefore, the kernel would be written as a tensor $W \in \mathbb{R}^{K\times K\times3}$, computing the output pixel value $f^{\text{out}}_{u,v} \in \mathbb{R}$ as a vector multiplication:

$$f^{\text{out}}_{u,v} = \sum_{(i,j)\in\mathcal{V}^2(W)} \boldsymbol{w}^{\top}_{i,j} \boldsymbol{f}^{\text{in}}_{u-i,v-j}. \tag{2.3}$$

In this example, the kernel applies a Gaussian blur filter over the input image. However, the kernel $\boldsymbol{W}$ can be designed to extract specific features from the input image $\boldsymbol{I}$, which can be used for different applications, such as edge detection, image blurring, or sharpening. We refer to the book by Förstner and Wrobel [52] for more examples.

In CNNs, the hand-defined convolution kernel weights are replaced by learnable parameters. This allows for optimizing the convolution kernels to extract features specific to the task for which the network is trained. The image convolution formulation can be generalized to an input $\boldsymbol{F}^{\text{in}}$ with an arbitrary channel size N^{in} as $\boldsymbol{F}^{\text{in}} \in \mathbb{R}^{U \times V \times N^{\text{in}}}$, leading to a kernel $\boldsymbol{W} \in \mathbb{R}^{K \times K \times N^{\text{in}}}$. Besides, CNNs often learn multiple kernels within a convolution to extract multi-dimensional feature maps $\boldsymbol{F}^{\text{out}} \in \mathbb{R}^{U \times V \times N^{\text{out}}}$ from the convolution input $\boldsymbol{F}^{\text{in}} \in \mathbb{R}^{U \times V \times N^{\text{in}}}$. Given the convolution input dimension N^{in} and the output dimension N^{out}, the learnable kernel is defined as $\boldsymbol{W}^{K^2 \times N^{\text{in}} \times N^{\text{out}}}$, with K being the kernel size. In this case, the kernel weights at a position (i, j) are a matrix $\boldsymbol{W}_{i,j} \in \mathbb{R}^{N^{\text{in}} \times N^{\text{out}}}$, which is multiplied by the input feature vector $\boldsymbol{f}^{\text{in}}_{u,v} \in \mathbb{R}^{N^{\text{in}}}$ to compute the output feature vector $\boldsymbol{f}^{\text{out}}_{u,v} \in \mathbb{R}^{N^{\text{out}}}$ as:

$$\boldsymbol{f}^{\text{out}}_{u,v} = \sum_{(i,j) \in \mathcal{V}^2(\boldsymbol{W})} \boldsymbol{W}^{\top}_{i,j} \boldsymbol{f}^{\text{in}}_{u-i,v-j}. \tag{2.4}$$

Apart from learnable kernels, CNNs often apply downsampling or upsampling in the convolutional layers, reducing or increasing the output feature $\boldsymbol{F}^{\text{out}}$ dimension size. Downsampling the input dimension and maintaining the kernel dimension increases the area covered by the kernels in the original input, increasing the kernel's receptive field. Downsampling can be achieved by applying the kernel $\boldsymbol{W}$ to the pixels with a stride. For example, with a stride of 2, the kernel would be applied to the input $\boldsymbol{F}^{\text{in}} \in \mathbb{R}^{U \times V \times N^{\text{in}}}$ starting from $\boldsymbol{f}^{\text{in}}_{1,1}$, then to $\boldsymbol{f}^{\text{in}}_{1,3}$, and so on, skipping one index in between, and computing the output $\boldsymbol{F}^{\text{out}} \in \mathbb{R}^{\frac{U}{2} \times \frac{V}{2} \times N^{\text{out}}}$ that is half the size of the input. Upsampling is typically used to recover the input's original size after downsampling layers. The upsampling is achieved using transposed convolutions. In this case, each convolution input feature vector $\boldsymbol{f}^{\text{in}}_{u,v} \in \mathbb{R}^{N^{\text{in}}}$ is multiplied by the individual kernel weights $\boldsymbol{W}_{i,j} \in \mathbb{R}^{N^{\text{in}} \times N^{\text{out}}}$, computing an output feature vector $\tilde{\boldsymbol{f}}^{\text{out}}_{i,j} \in \mathbb{R}^{N^{\text{out}}}$:

$$\tilde{\boldsymbol{f}}^{\text{out}}_{i,j} = \boldsymbol{W}^{\top}_{i,j} \boldsymbol{f}^{\text{in}}_{u,v}. \tag{2.5}$$

The individual output features $\tilde{\boldsymbol{f}}^{\text{out}}_{i,j}$ are then combined, resulting in an output matrix $\tilde{\boldsymbol{F}}^{\text{out}}_{u,v} \in \mathbb{R}^{K^2 \times N^{\text{out}}}$ with the same size as the kernel $\boldsymbol{W}$. The computed feature matrices $\tilde{\boldsymbol{F}}^{\text{out}}_{u,v}$ are then aggregated, given the convolution stride size and summing up the overlapping regions, defining the convolution output $\boldsymbol{F}^{\text{out}}$. For example,

in a convolution with a stride of 2, the input of size $\mathsf{F}^{\text{in}} \in \mathbb{R}^{U \times V \times N^{\text{in}}}$ is upsampled to an output feature $\mathsf{F}^{\text{out}} \in \mathbb{R}^{2U \times 2V \times N^{\text{out}}}$ with double the size of the input F^{in}.

Those convolution operations with learnable kernel parameters are combined with non-linear activation functions and normalization layers to ensure a zero-centered and normalized distribution over the computed features, defining convolutional blocks. Those convolutional blocks are consecutively stacked, defining the network. The CNN can then be trained to achieve specific vision tasks, and the convolutions kernel parameters are optimized to extract features specific to the target task. Those convolutional neural networks have become a standard technique for computer vision tasks, as features specific to the target task can be learned from the training data without hand-defining them, enabling significant progress in various perception tasks, such as image classification [38], semantic segmentation [5, 35], and object detection [54, 167].

2.1.2 3D Convolutional Neural Networks

The 2D convolutions detailed in Sec. 2.1.1 can be extended to process 3D data. In this case, the convolution has three-dimensional kernels $\mathsf{W} \in \mathbb{R}^{K^3 \times N^{\text{out}} \times N^{\text{in}}}$, with K as the kernel size, with offsets $\mathcal{V}^3(\mathsf{W})$. To employ the convolution operation detailed in Sec. 2.1.1 over a 3D point cloud $\mathcal{P} = \{\boldsymbol{p}_1, \dots, \boldsymbol{p}_P\}$ with P points $\boldsymbol{p}_p \in \mathbb{R}^3$ where $\boldsymbol{p} = (x, y, z)$, we need to represent it as a regular 3D grid, such that the 3D kernel can be applied to the point cloud, as in the 2D case. To do so, we voxelize the point cloud $\mathcal{P}$ given the grid resolution $\Delta s \in \mathbb{R}$, defining the corresponding discrete grid coordinates $\boldsymbol{u} \in \mathbb{Z}^3$:

$$\boldsymbol{u} = \left(\left\lfloor \frac{x}{\Delta s} \right\rfloor, \left\lfloor \frac{y}{\Delta s} \right\rfloor, \left\lfloor \frac{z}{\Delta s} \right\rfloor \right), \tag{2.6}$$

with $\lfloor \cdot \rfloor$ as the flooring operation. With the voxelization, different points $\boldsymbol{p}_p$ may refer to the same discrete coordinate $\boldsymbol{u}$. Therefore, each voxel in the grid stores all the points $\boldsymbol{p}_p$ falling into that grid cell, as well as features of the points in the cell. Such features can include an intensity value, the occupancy of the grid cell or even the original points $\boldsymbol{p}_p$ coordinates. After voxelization, the 3D voxel grid $\mathsf{F}^{\text{in}} \in \mathbb{R}^{U \times V \times R \times N^{\text{in}}}$ is defined as a tensor, where N^{in} represents the dimension of the features stored in the grid cells. With this regular 3D grid representation of the point cloud $\mathcal{P}$, we can apply the 3D kernel W over the 3D grid F^{in}, computing the corresponding output grid cells from $\mathsf{F}^{\text{out}} \in \mathbb{R}^{U \times V \times R \times N^{\text{out}}}$ as:

$$\boldsymbol{f}^{\text{out}}_{\boldsymbol{u}} = \sum_{\boldsymbol{i} \in \mathcal{V}^3(\mathsf{W})} \mathsf{W}_{\boldsymbol{i}} \boldsymbol{f}^{\text{in}}_{\boldsymbol{u}-\boldsymbol{i}}, \tag{2.7}$$

with $\boldsymbol{u} \in \mathbb{Z}^3$ as the 3D grid discrete coordinate and $\boldsymbol{i} \in \mathbb{Z}^3$ as the offset vectors from the set of kernel offsets $\mathcal{V}^3(\mathsf{W})$. As in the 2D convolution, padding,

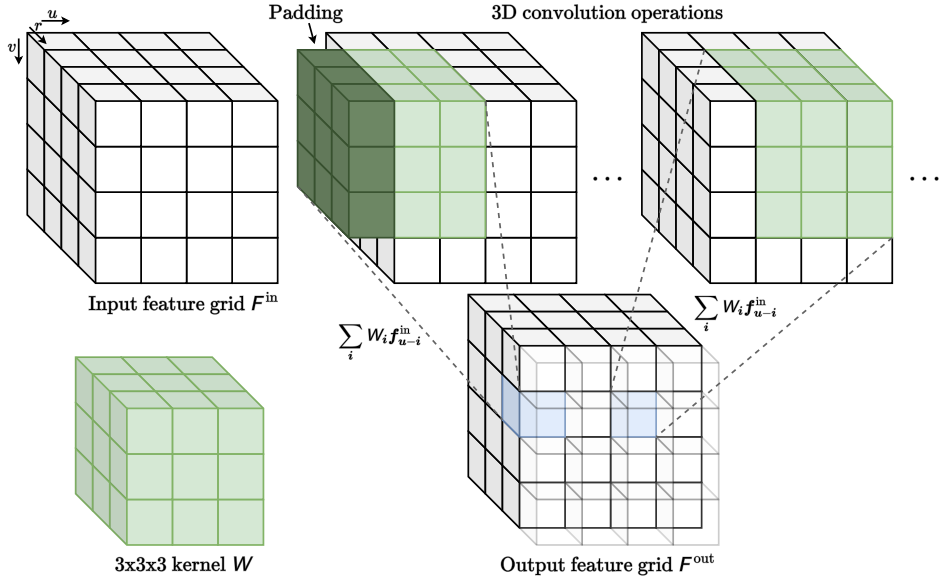


Figure 2.2: Example of a 3D convolution over the input feature grid F^{in}. Similar to the 2D convolution, the 3D kernel W is applied to all voxels in the input feature grid F^{in}, computing the output feature grid F^{out}.

downsampling and upsampling convolutions can also be applied. The learnable convolutions are followed by a non-linear activation and a normalization layer to build the convolutional blocks as in the 2D case. The 3D convolutional neural networks can be defined by consecutively stacking the 3D convolutional blocks to process 3D data. Fig. 2.2 shows an example of a $3 \times 3 \times 3$ kernel applied to a 3D grid, illustrating how the 3D convolution operates on the data. The kernels in this case operate over the 3D points neighborhood, extracting information from the voxel features.

2.1.3 Sparse 3D Convolutions

The 3D convolutions detailed in Sec. 2.1.2 enable the implementation of 3D CNNs to extract features from 3D data. However, such a dense grid representation, borrowed from image processing, is not scalable for scene-scale 3D data. Taking as an example a 3D LiDAR point cloud with a radius of 50 m collected in an urban environment, and voxelizing it with a voxel resolution of $0.1\,m$. After voxelizing this point cloud, the resulting dense 3D grid would contain $\frac{50^3}{0.1} = 1250000$ voxels, of which most are unoccupied. The 3D kernel would be applied to all voxels in this grid, including both occupied and unoccupied voxels, resulting in several unnecessary computations over the empty voxels. This inefficient computation hinders the scalability of 3D convolutions to scene-scale 3D data.

Sparse tensors were proposed to enable efficient 3D convolution operations [34,

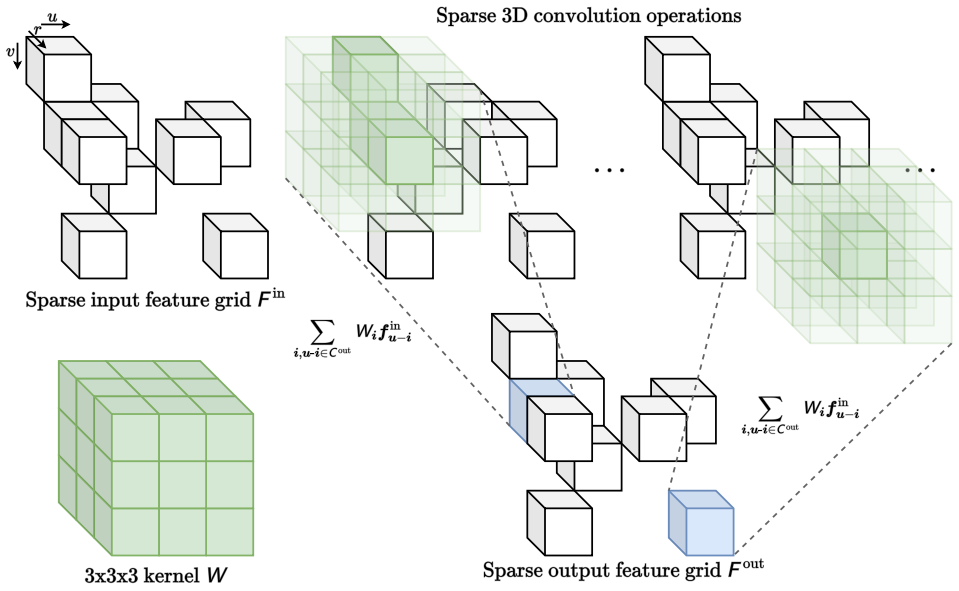


Figure 2.3: Example of a sparse 3D convolution over the sparse input feature grid F^{in}. The 3D kernel W is applied to the occupied voxels in F^{in}, applying the kernel operations only over the neighboring voxels that are occupied to compute the sparse output feature grid F^{out}.

57]. In this case, the dense 3D voxel grid representation is replaced by a sparse representation, storing only the occupied voxels from the grid. A sparse tensor is defined by the occupied voxels' coordinates $C \in \mathbb{R}^{P\times 3}$ and its features $F \in \mathbb{R}^{P\times N^{\text{in}}}$:

$$C = \begin{bmatrix} x_1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ x_N & y_N & z_N \end{bmatrix}, \quad F = \begin{bmatrix} \boldsymbol{f}_1^\top \\ \vdots \\ \boldsymbol{f}_N^\top \end{bmatrix}. \tag{2.8}$$

The voxel grid $\mathcal{G}$ from a point cloud is represented as the tuple of sparse tensors of occupied voxels coordinates and the corresponding features as $\mathcal{G} = (C, F)$. Ignoring unoccupied voxels with a more efficient and scalable 3D data representation.

This sparse representation enables an efficient 3D data representation while maintaining the neighboring information necessary for the convolution operation. The 3D convolution can be applied to the sparse voxel grid $\mathcal{G}^{\text{in}} = (C^{\text{in}}, F^{\text{in}})$, as in Sec. 2.1.2, given the kernel $W \in \mathbb{R}^{K^3\times N^{\text{out}}\times N^{\text{in}}}$. As in Eq. (2.7), the output sparse features F^{out} are computed over each cell coordinate C^{in}. However, the kernel computation is only done over the offset coordinates $\boldsymbol{i}$ present in C^{in} as:

$$\boldsymbol{f}^{\text{out}}_{\boldsymbol{u}} = \sum_{\boldsymbol{i}\in\mathcal{V}^3(W), \boldsymbol{u}-\boldsymbol{i}\in C^{\text{in}}} W_{\boldsymbol{i}} \boldsymbol{f}^{\text{in}}_{\boldsymbol{u}-\boldsymbol{i}} \quad \text{for } \boldsymbol{u} \in C^{\text{out}}. \tag{2.9}$$

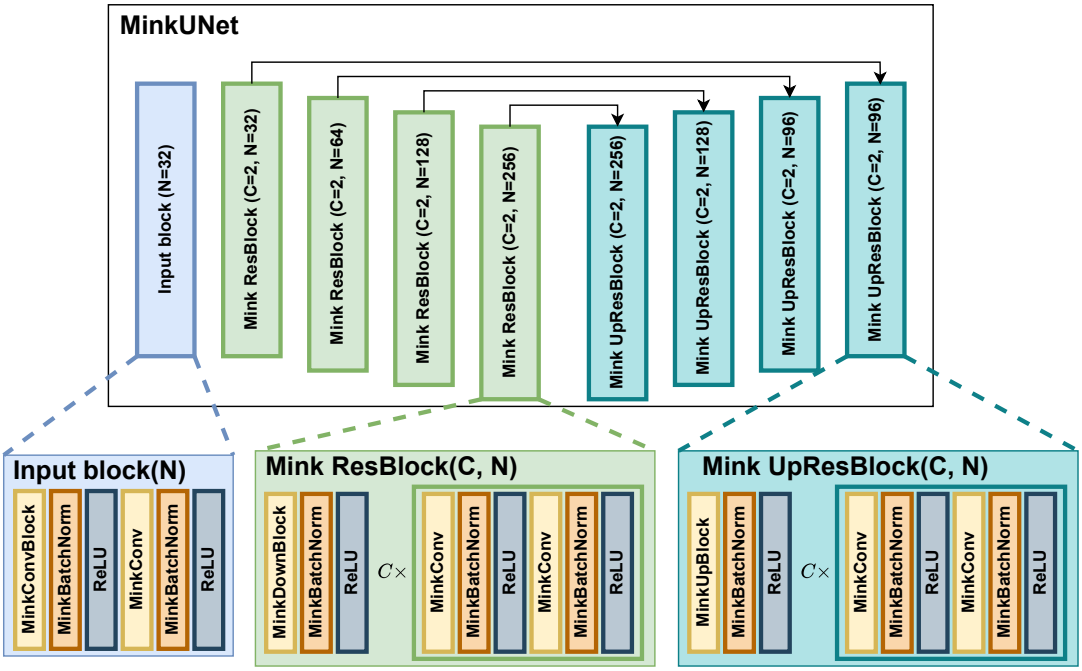


Figure 2.4: Diagram of the MinkUNet sparse 3D CNN architecture. The backbone used a ResUNet architecture [65] implemented with sparse operations implemented by the Minkowski Engine [34] library. The Mink ResBlock layers are downsampling layers with C residual blocks and N output features. The Mink UpResBlock are upsampling layers also with C residual blocks and N output features.

In this case, the convolution operation is only applied over occupied voxels, efficiently scaling the processing of scene-scale 3D data as depicted in Fig. 2.3. As with 3D dense convolutions, a 3D sparse CNN can be built using sparse convolutions, with upsampling and downsampling operations followed by non-linear activation and a normalization layers.

2.1.4 MinkUNet

3D sparse CNNs are used throughout the chapters of this thesis in the different proposed methods. More specifically, we use the MinkowskiEngine for sparse operations [34] and processing of 3D data. We use the MinkUNet architecture proposed by Choy et al. [34] as basis. This architecture uses sparse operations to implement concepts commonly employed in 2D CNNs, such as residual connections [65] and U-shaped networks with skip connections [149], which enable the compression and decompression of the input data dimensions while maintaining high-level information during the feature extraction. Fig. 2.4 depicts the network architecture used as basis in this thesis's chapters. The first input block maps the input point cloud to a feature space, maintaining the input resolution. Then, the features from the input block are processed by consecutive downsampling convo-

lutional blocks, reducing the dimension of the intermediary feature maps while increasing the receptive field of the network kernels. Finally, the features are passed over consecutive upsampling convolutional blocks, until arriving back to the same resolution as the input point cloud, computing the final output features for each point in the input point cloud. In the later chapters, some variations of this architecture may be adopted, depending on the target perception task, maintaining the overall structure.

2.2 Self-Supervised Pre-Training with Contrastive Learning

The standard practice for training a neural network is to initialize the convolution kernel weights randomly. Then, those random weights are adjusted during training to perform the target perception task. Although effective, starting the network with random weights requires many training iterations and a substantial amount of labeled data for the network parameters to be properly optimized for the target task.

Pre-training a network is a common alternative to random initialization of the kernel parameters, which often leads to improvement in the network performance using fewer iterations and less labeled data [21, 64]. During pre-training, the network is trained with other available datasets. At this stage, the goal is to train the network with as much data as possible, allowing the network parameters to extract meaningful features from the data. Then, the pre-trained network weights are fine-tuned for the target dataset by using them to initialize the kernel parameters at the beginning of the training.

Network pre-training is often done in one of two ways: supervised or self-supervised. The supervised pre-training trains the network with annotated data. Such supervised pre-training is a valuable strategy when the data used for pre-training has labels for the same perception task and the same domain as the target dataset. However, suppose the pre-training and target datasets are labeled for different tasks or come from different domains. In that case, the supervised pre-trained network may not yield a significant improvement in the target dataset, or even achieve worse performance than the random initialization strategy due to the domain gap between the two datasets.

Self-supervised pre-training provides a more powerful strategy. For self-supervised pre-training, the labels used to train the network are generated online based on a pretext task, such as image colorization [79], motion prediction from videos [41], or image reconstruction [63]. Those pretext tasks are more general and not task-specific. Since the labels are generated online, such self-supervised

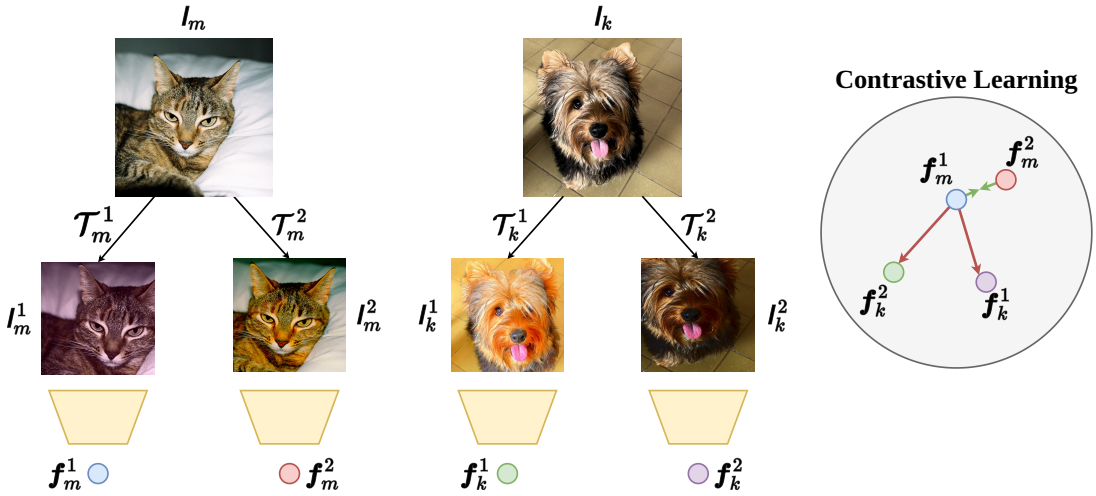


Figure 2.5: Visual example of the contrastive learning discriminative task. Given two images I_m and I_k we apply two sets of random image transformations over each image, generating the augmented pairs $\{I_m^1, I_m^2\}$ and $\{I_k^1, I_k^2\}$. Then, we compute the corresponding feature vectors $\{\boldsymbol{f}_m^1, \boldsymbol{f}_m^2\}$ and $\{\boldsymbol{f}_k^1, \boldsymbol{f}_k^2\}$ with a CNN, trained to discriminate between features from the transformed versions from the image I_m while pushing apart features from the images $I_{k \neq m}$ with Eq. (2.11).

methods are more scalable than supervised methods, often learning stronger representations during pre-training that lead to better performance after fine-tuning the network.

One way of self-supervised pre-training is contrastive learning. The downside of the self-supervised approaches listed before is that those methods target trivial pretext tasks based on simple data transformations. Given the discriminative aspect of perception tasks in general, contrastive learning [173] gained interest in the computer vision community by defining a pretext task focused on differentiating between similar and different samples. This pretext task also leverages data transformations, but instead of learning to reverse them, it learns how to distinguish between transformed versions of the same data sample within the learned feature space.

To formalize this task, we define a sample I_m from a dataset $\mathcal{I} = \{I_1, \ldots, I_M\}$ containing M samples. For each image $I_m \in \mathbb{R}^{U \times V \times 3}$, we sample several consecutive random image transformations, defining two sets of image transformations, $\mathcal{T}_m^1$ and $\mathcal{T}_m^2$. The transformations are applied to I_m to compute two distinct versions of it, $I_m^1 = \mathcal{T}_m^1(I_m)$ and $I_m^2 = \mathcal{T}_m^2(I_m)$, generating a total of $2M$ samples $\mathcal{I} = \{I_1^1, I_1^2, \ldots, I_M^1, I_M^2\}$. Then, an encoder network used to map the images in $\mathcal{I}$ into the corresponding set of feature vectors $\mathcal{F} = \{\boldsymbol{f}_1^1, \boldsymbol{f}_1^2, \ldots, \boldsymbol{f}_M^1, \boldsymbol{f}_M^2\}$, with $\boldsymbol{f}_m^{p \in \{1,2\}} \in \mathbb{R}^{N^{\text{out}}}$ where N^{out} is the network output feature vector dimension. The goal is to distinguish between the two samples I_m^1 and I_m^2 coming from the same image I_m from all the other $2M - 2$ samples $I_{k \neq m}^{p \in \{1,2\}}$, given the feature

vectors computed by the model. This discriminative task is formulated based on the cosine similarity. Given the augmented samples $p \in \{1, 2\}$ from image i, and $q \in \{1, 2\}$ from image j in the batch, the similarity $\delta_{i,j}^{p \to q}$ is computed as:

$$\delta_{i,j}^{p \to q} = \frac{(\boldsymbol{f}_i^p)^\top \boldsymbol{f}_j^q}{\tau}, \tag{2.10}$$

where τ is a normalization temperature parameter. With Eq. (2.10), we can compute the InfoNCE loss [173] between a pair of transformed samples I_m^1 and I_m^2 from an image I_m and all the other transformed images $I_k^{p \in \{1,2\}}$ from $\mathcal{I}$ with their corresponding feature vectors as:

$$\mathcal{L} = -\log \left(\frac{\exp\left(\delta_{m,m}^{1 \to 2}\right)}{\exp\left(\delta_{m,m}^{1 \to 2}\right) + \sum_{k,k \neq m}^{M} \exp\left(\delta_{m,k}^{1 \to p \in \{1,2\}}\right)} \right). \tag{2.11}$$

From Eq. (2.11), the contrastive learning objective aims to increase the similarity between the positive pairs, I_m^1 and I_m^2, which come from the same image I_m, while decreasing the similarity with the non-related $2M - 2$ transformed images $I_{k \neq m}^{p \in \{1,2\}}$. Fig. 2.5 shows an example of this discrimination between images.

To achieve this discrimination between images, the network must optimize the convolution kernels to extract features invariant to the various transformations applied to the image, allowing it to correlate the transformed samples coming from the same anchor image. It enables the network to learn strong data representations without requiring data annotation, and thus becomes a useful self-supervised pre-training strategy. This contrastive learning strategy was used as the basis for the methods proposed in Chapters 3 and 4, reformulated to learn strong object-centric features from 3D point cloud data.

2.3 Autoencoder Networks

Neural networks have been widely used in the computer vision community to achieve specific perception tasks. Those deep networks are optimized to extract features specific to the task for which they are trained. This learned feature space is capable of embedding sufficient semantic information for the model to achieve the target tasks. Given the ability of these models to optimize their parameters to learn a descriptive feature space, these deep networks were used to learn a compressed latent feature space that can embed all relevant information from the input data. Autoencoders use encoder-decoder network architectures to encode the training data into a compact feature space by leveraging an encoder and a decoder network. The encoder and decoder are trained together such that the encoder learns to encode the input data into a compact representation in the form of a feature vector. This encoded feature vector then becomes a smaller

representation of the input data, carrying all the relevant information to allow the decoder to reconstruct the image. The decoder is trained to reverse this mapping by learning how to decompose the encoded feature vector back to the original data. Those autoencoder networks were then used for various applications, such as data compression [32, 184], dimensionality reduction [36, 139].

2.3.1 Autoencoders

Autoencoders are often composed of two networks, an encoder ϕ and a decoder ψ. As an example, given the set of training images $\mathcal{I} = \{\boldsymbol{I}_1, \ldots, \boldsymbol{I}_M\}$ and a sample image $\boldsymbol{I}_m$, the encoder ϕ learns a feature space to compress the image $\boldsymbol{I}_m$ into a latent feature vector $\boldsymbol{z}_m = \phi(\boldsymbol{I}_m)$, where $\boldsymbol{z}_m \in \mathbb{R}^{N^z}$ with N^z as the latent dimension. Then, the decoder ψ receives as input the encoded image $\boldsymbol{z}_m$ and learns to reconstruct the image $\boldsymbol{I}'_m = \psi(\boldsymbol{z}_m)$. The goal is to reconstruct $\boldsymbol{I}'_m$ from the latent $\boldsymbol{z}_m$ as similar as possible to the original image such that $\boldsymbol{I}'_m \simeq \boldsymbol{I}_m$, optimizing the encoder ϕ and the decoder ψ with the mean squared error (MSE) loss:

$$\mathcal{L} = \frac{1}{M} \sum_{i=1}^{M} \left(\boldsymbol{I}_i - \boldsymbol{I}'_i\right)^2 . \tag{2.12}$$

From this loss function, both the encoder ϕ and the decoder ψ are optimized together to reconstruct an image $\boldsymbol{I}_m$ from the latent vector $\boldsymbol{z}_m$. Therefore, the encoder ϕ learns to embed the relevant information from the image $\boldsymbol{I}_m$ into $\boldsymbol{z}_m$ such that the decoder ψ can reconstruct an image $\boldsymbol{I}'_m \simeq \boldsymbol{I}_m$.

2.3.2 Variational Autoencoders

Despite being effective, the applications of autoencoder networks are limited. The model is optimized to achieve a deterministic mapping between an image $\boldsymbol{I}_m$ and its latent representation $\boldsymbol{z}_m$. This leads to an unbounded latent space learned by the encoder ϕ, which maps the training samples to a specific feature vector in the latent space. In this case, the reconstruction of an image $\boldsymbol{I}_m$ not present in the training images may not be properly encoded by the encoder ϕ and, therefore, poorly reconstructed by the decoder ψ. To overcome that, the feature space learned by the encoder ϕ can be regularized to represent the training samples as a data distribution rather than a deterministic mapping.

Given the training data distribution $p(\boldsymbol{I})$, a variational autoencoder (VAE) [82] is trained to encode a sample $\boldsymbol{I}_m \sim p(\boldsymbol{I})$ into a latent vector $\boldsymbol{z}_m$. The difference from the regular autoencoder is that, instead of optimizing for a one-to-one mapping between the image $\boldsymbol{I}_m$ and the feature vector $\boldsymbol{z}_m$, the VAE encodes the image into a continuous data distribution. To do so, instead of directly computing the

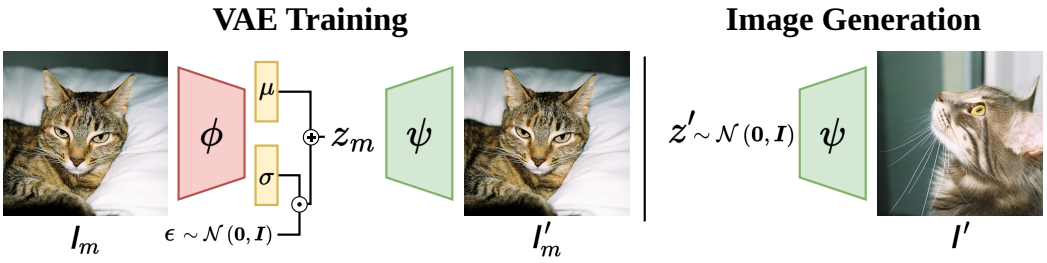


Figure 2.6: Given image I_m the VAE is trained to encode the image into a latent vector $\boldsymbol{z}_m$ given the predicted mean $\boldsymbol{\mu}_m$ and standard deviation $\boldsymbol{\sigma}_m$ given I_m, and reconstruct the original image from the latent $\boldsymbol{z}_m$ as I'_m. New images can be generated with the trained VAE by sampling random latent vectors $\boldsymbol{z}' \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and decoding it to a new image I'.

feature vector $\boldsymbol{z}_m$, the encoder ϕ predicts the mean $\boldsymbol{\mu}_\phi$ and the standard deviation $\boldsymbol{\sigma}_\phi$ of the learned distribution $q_\phi(\boldsymbol{z}|I)$ given the input image I_m. Then, the feature vector $\boldsymbol{z}_m$ is sampled with respect to the predicted mean $\boldsymbol{\mu}_\phi$ and standard deviation $\boldsymbol{\sigma}_\phi$ as:

$$\boldsymbol{z}_m \sim \mathcal{N}\left(\boldsymbol{\mu}_\phi, \boldsymbol{\sigma}_\phi\right). \tag{2.13}$$

In simpler terms, instead of encoding I_m into a deterministic latent feature $\boldsymbol{z}_m$, the VAE encodes the image I_m into a random latent $\boldsymbol{z}_m$ within the corresponding predicted mean $\boldsymbol{\mu}_\phi$ and standard deviation $\boldsymbol{\sigma}_\phi$, enforcing the learned data distribution $q_\phi(\boldsymbol{z}|I)$ to be continuous.

Although continuous, the encoder learned distribution $q_\phi(\boldsymbol{z}|I)$ can still be unbounded. This can lead to the mapping of individual samples from the training set to specific continuous regions in the latent space, arriving at the same problem as the standard autoencoder. To enforce the encoder ϕ to learn a bounded and descriptive data distribution, the VAE is optimized to define a distribution $q_\phi(\boldsymbol{z}|I)$ that matches a target prior distribution $q(\boldsymbol{z})$. For that, the Kullback-Leibler divergence, or short KL-Divergence, [125] is used to compute the distance between both distributions, training the network to minimize this distance:

$$\mathcal{L}_{\text{latent}} = \mathbb{KL}(q_\phi\left(\boldsymbol{z}_m|I_m\right) \,||\, q\left(\boldsymbol{z}\right)), \tag{2.14}$$

where the prior distribution $q(\boldsymbol{z})$ is often defined as a zero-centered isotropic normal distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. By parameterizing $\boldsymbol{z}_m$ with respect to the mean $\boldsymbol{\mu}_\phi$ and the standard deviation $\boldsymbol{\sigma}_\phi$, and approximating the encoder learned data distribution to a normal distribution, the VAE learns to map the training data distribution $p(I)$ into a continuous and well-behaved distribution $q_\phi(\boldsymbol{z}|I) \approx q(\boldsymbol{z})$, which enforces a continuous mapping between the training and latent distributions.

This variational formulation of the encoder enables the use of VAEs also as generative models [202]. Novel samples from the training data distribution $p(I)$

can be generated with the VAE by sampling a random latent vector $\boldsymbol{z}' \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Then, the decoder ψ can be used to decode the randomly sampled latent $\boldsymbol{z}'$ to generate a novel sample $\boldsymbol{I}' = \psi(\boldsymbol{z}')$. Given that the feature vector $\boldsymbol{z}'$ was sampled from the same distribution as the one the VAE encoder was optimized for, the decoded image $\boldsymbol{I}'$ would approximate to the training data distribution $\psi(\boldsymbol{z}') = \boldsymbol{I}' \sim p(\boldsymbol{I})$. Fig. 2.6 exemplifies the VAE training and the image generation from a random latent vector $\boldsymbol{z}' \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. This VAE formulation is used in Chapter 7 together with diffusion models to encode 3D annotated data into a latent representation, generating new labeled 3D scenes by decoding the latent features sampled from the learned latent distribution.

Chapter 3

Representation Learning from Unlabeled 3D Data

A key challenge in developing perception systems arises from the need for large amounts of annotated data to account for all possible situations and objects that can appear in the real world. For larger 3D point clouds, the scalability of data annotation is even more challenging due to the complexity of labeling these point clouds. At the same time, the performance of such learning-based systems tightly relates to the amount and the variability of available training samples. A common approach for overcoming this is to pre-train the neural network before optimizing it for the target task. Several works investigate various pre-training strategies, e.g., image reconstruction [63], image colorization [79], and contrastive learning [21, 64], see also Chapter 2.2.

Contrastive learning pre-training approaches have gained increasing attention in the computer vision field due to the strong representations learned with them. The first approaches used an encoder network optimized with the InfoNCE [173] loss to distinguish between positive samples, i.e., the two augmented samples from the same anchor image, and negative samples, i.e., the augmented samples from other images in the mini-batch [21]. The study in the field led to the development of more advanced methods, using a siamese network scheme with an online encoder that is optimized based on the loss gradients, and a momentum encoder updated as an exponential moving average of the online model [64]. The momentum network is then used to create a bank of negative samples, implemented as a first-in-first-out queue. At each training iteration, the feature vectors of the samples in the batch are computed using the momentum encoder and added to the feature bank, virtually increasing the mini-batch size and the number of negatives used to compute the loss. Given the dependency on negative samples, more recent approaches have proposed adding a few multilayer perceptron (MLP) layers to the momentum network, introducing asymmetry between the two net-

works, and eliminating the necessity for negative samples. Such methods were extended to not only optimize for learning a feature vector per image but also for learning a pixel-level representation [174], regarded as a pre-training strategy for more fine-grained downstream tasks, e.g., semantic segmentation, panoptic segmentation, and object detection.

Such contrastive learning strategies were also proposed as a pre-training strategy for 3D neural networks. The direct application of image-based methods to the 3D domain however is not straightforward. Especially for scene-scale data, learning strong representations is more challenging due to the limited features in the input 3D data compared to images. To encourage the network to learn more geometrically related features, PointContrast [196] proposes augmenting the point clouds with several similarity transformations and using the InfoNCE loss computed over the point-wise features. The network is optimized to learn strong local features that account for the point neighborhood, ignoring, however, the scene context. DepthContrast [209] proposed a similar approach using a 3D encoder CNN to compute a single feature vector to describe the entire 3D scene. In this case, the network learns to extract a global descriptor for the scene, but, opposite to PointContrast [196], it ignores the local information for the points. ContrastiveSceneContext [71] extends PointContrast [196] by dividing the scene into quadrants and predicting within each point the quadrant to which the point belongs. This quadrant prediction embeds more global-level features but still lacks object-level information.

In this chapter, we focus on using the contrastive learning strategy to train a neural network to learn object-level information from a 3D point cloud. We leverage an unsupervised ground segmentation method to identify object-related points and cluster these into individual instances of objects in the scene. Given the instances of objects in the 3D point clouds, we train an encoder-decoder neural network to extract point-wise features. The model is optimized to discriminate between the segments in the scene using the InfoNCE loss, training the network to embed object-level information without requiring any labels. We evaluate our approach by first pre-training a neural network with our proposed method using the SemanticKITTI dataset [5, 54] point clouds without the labels, and then fine-tuning this pre-trained model for different downstream tasks with the labels, comparing it with prior 3D contrastive pre-training methods. Given the object-level representation learned during pre-training, the network pre-trained with our method achieves improved performance after fine-tuning to the target perception task compared to prior works and to the network without pre-training. The experiments demonstrate that the representation learned by the network with our pre-training embeds relevant information that can be used during fine-tuning, improving the performance in the downstream task with fewer labels.

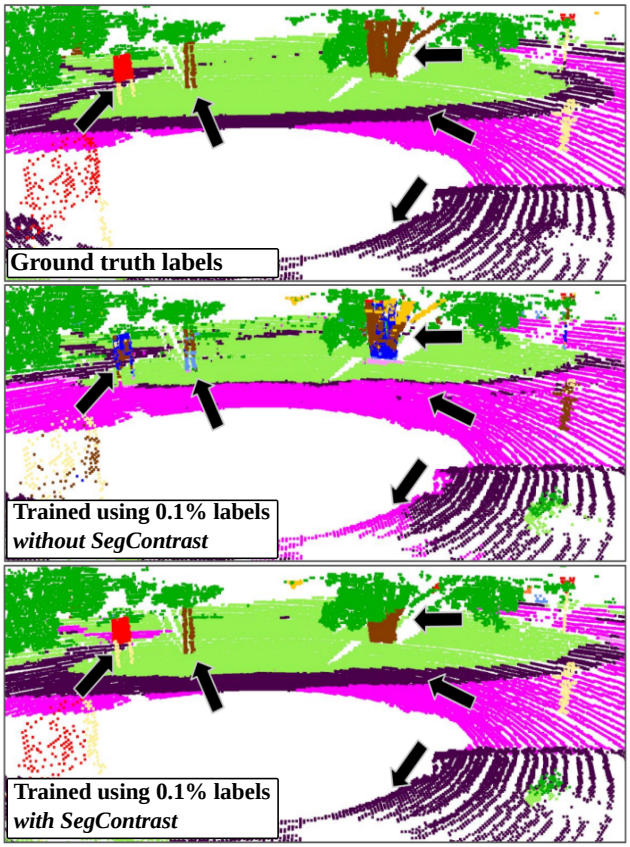


Figure 3.1: Results trained with only 0.1% of the labels, trained from scratch (middle), i.e., without any pre-training, and pre-training with our approach SegContrast (bottom). With SegContrast pre-training, the semantic segmentation can better delineate the structural information in the scene and better classify the more fine-grained objects, like trees, traffic signs and poles, highlighted in the black arrows.

3.1 3D LiDAR Data Representation Learning

The main contribution of this chapter is a self-supervised representation learning method for 3D LiDAR point clouds that is able to learn structural information. Our method extracts segments from the LiDAR data and uses contrastive learning to discriminate between similar and dissimilar structures. The learned feature representation is then used as a starting point for supervised fine-tuning, reducing the number of labeled training data needed. Our results suggest that our method can better learn the structural information (see Fig. 3.1) and a more descriptive feature representation during the self-supervised pre-training, outperforming previous point cloud-based contrastive methods in different evaluations. In summary, in this chapter we propose a 3D LiDAR representation learning method that, compared to prior state-of-the-art approaches, (i) is more efficient when using fewer labels, (ii) can better describe fine-grained structures, and (iii) is more transferable between different datasets.

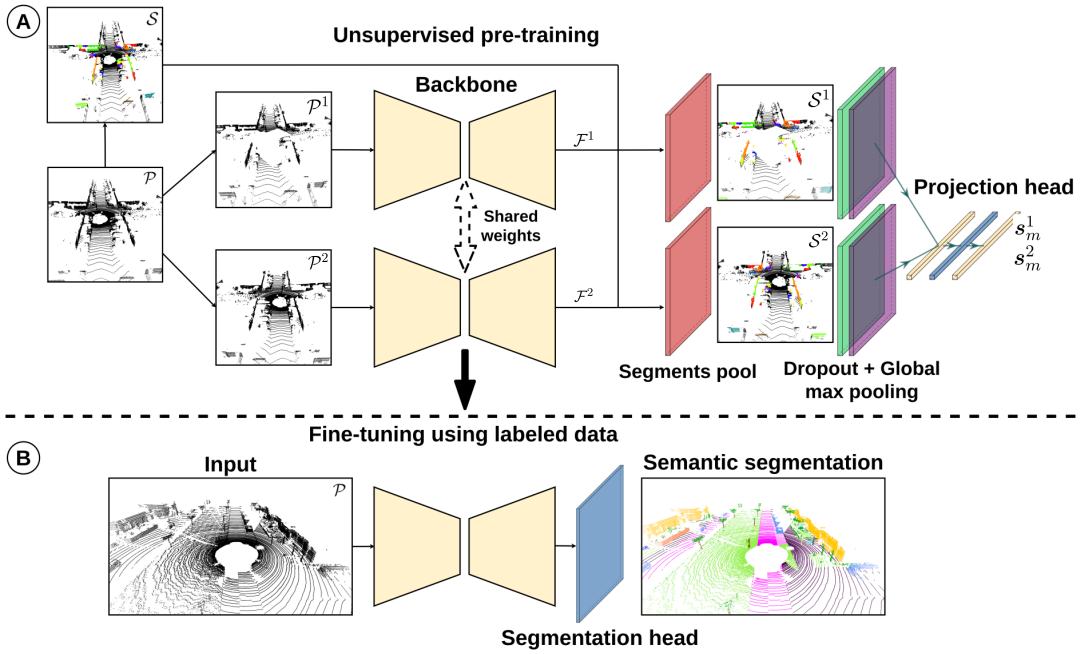


Figure 3.2: (A) From a point cloud $\mathcal{P}$, we extract class-agnostic segments $\mathcal{S}$ and then we generate the augmented views $\mathcal{P}^1$ and $\mathcal{P}^2$ by data augmentation. We compute the point-wise features $\mathcal{F}^1$ and $\mathcal{F}^2$ and determine the augmented segments $\mathcal{S}^1$ and $\mathcal{S}^2$ with its point-wise features using the point indexes of $\mathcal{S}$ extracted from $\mathcal{P}$. Then, we apply dropout followed by global max pooling over each segment, and we project the segment feature vectors using the projection head to get the final features $\boldsymbol{s}_m^1$ and $\boldsymbol{s}_m^2$ from the M segments and compute the contrastive loss. (B) After the pre-training, we fine-tune the pre-trained backbone for the downstream task, i.e., semantic segmentation.

3.1.1 Our Approach

Fig. 3.2 provides an overview of our approach. We rely on a class-agnostic point cloud segmentation to segment the structures in the scene. Unlike most prior work, our method uses a single backbone and does not require a point-wise mapping. We use the online and momentum encoders scheme and a feature bank [64] during training to increase the number of negatives samples and contrast structures segmented over different scans. Then, point-wise features are computed for the whole point cloud and the mapping of the segment is used to extract the segment-wise points and features. We apply dropout [166] and global max pooling for each segment and compute a feature vector using an MLP projection head [21]. After that, we calculate the contrastive loss over the segments and update the feature bank with these segments features. In the next sections, we provide more details on the individual parts of our method. The unsupervised segments' extraction is discussed in Sec. 3.1.2, the data augmentation applied to the segments is detailed in Sec. 3.1.3, Sec. 3.1.4 discusses the segment-wise contrastive loss, and finally Sec. 3.1.5 details the whole pipeline of our method.

3.1.2 Unsupervised Segment Extraction

Our method relies on segments of different structures in the point cloud. In outdoor LiDAR data, the different objects in a scene are mainly connected by the ground and usually better separated compared to indoor scenes. Given this characteristic, it is comparably easy to extract segments without labels in two steps by removing first the ground, then clustering the remaining points [7, 56, 67].

Given a point cloud $\mathcal{P} = \{\boldsymbol{p}_1, \dots, \boldsymbol{p}_R\}$ with points $\boldsymbol{p}_r \in \mathbb{R}^3$, we can fit the ground plane and partition the point cloud $\mathcal{P}$ into ground $\mathcal{G}$ and non-ground points $\tilde{\mathcal{P}}$, such that $\mathcal{P} = \tilde{\mathcal{P}} \cup \mathcal{G}$ and $\tilde{\mathcal{P}} \cap \mathcal{G} = \emptyset$, similarly to prior approaches [7, 67]. Then, using a clustering algorithm, we can divide $\tilde{\mathcal{P}}$ into M segments $\mathcal{S}_m$, such that $\tilde{\mathcal{P}} = \bigcup_{m=1}^{M} \mathcal{S}_m$ and $\bigcap_{m=1}^{M} \mathcal{S}_m = \emptyset$, i.e., the segments are mutually exclusive. Each segment $\mathcal{S}_m$ in this partition represents a different structure from the original point cloud.

More specifically, we use RANSAC [50] to fit the ground plane and define the ground $\mathcal{G}$ and non-ground points $\tilde{\mathcal{P}}$. Given the fitted plane and a distance threshold ν, the inliers (ground) and outliers (non-ground) points are partitioned. We use DBSCAN [46] to cluster the non-ground points $\tilde{\mathcal{P}}$ and determine the segments $\mathcal{S}_m$.

A common problem of such class-agnostic segmentation is the aspect of over- and under-segmentation [7]. To overcome this and extract representative segments, we define a minimum number of points ξ in a cluster to be considered a segment. Moreover, since we will have a different number of segments for every point cloud segmentation, we select the ζ segments with the highest number of points to avoid memory overflow during training. The remaining segments are also added to the set of ground points $\mathcal{G}$.

Despite its simplicity, this segmentation method can divide the scene into distinct structures, as illustrated in the Fig. 3.3. To save computations while training, we cache the segments after the first pass over the point clouds.

3.1.3 Segment Augmentation

With each point assigned to a segment, we apply data augmentations to generate the segments augmented pairs $\mathcal{S}^1$ and $\mathcal{S}^2$. We extract random views, $\mathcal{P}^1$ and $\mathcal{P}^2$, by cropping a random cuboid region from the anchor point cloud $\mathcal{P}$ [209]. Then, we apply random augmentations individually over the point clouds $\mathcal{P}^1$ and $\mathcal{P}^2$. We use random rotation around the z axis, random scale, random flip, random cuboid dropout [209], point jittering, and rotation perturbations around x, y, and z axes to augment the views. Two independent sets of augmentations $\mathcal{T}^1$ and $\mathcal{T}^2$ are defined and applied respectively to each augmented view $\mathcal{P}^1$ and $\mathcal{P}^2$.

By extracting a pair of views and applying those augmentations on the point

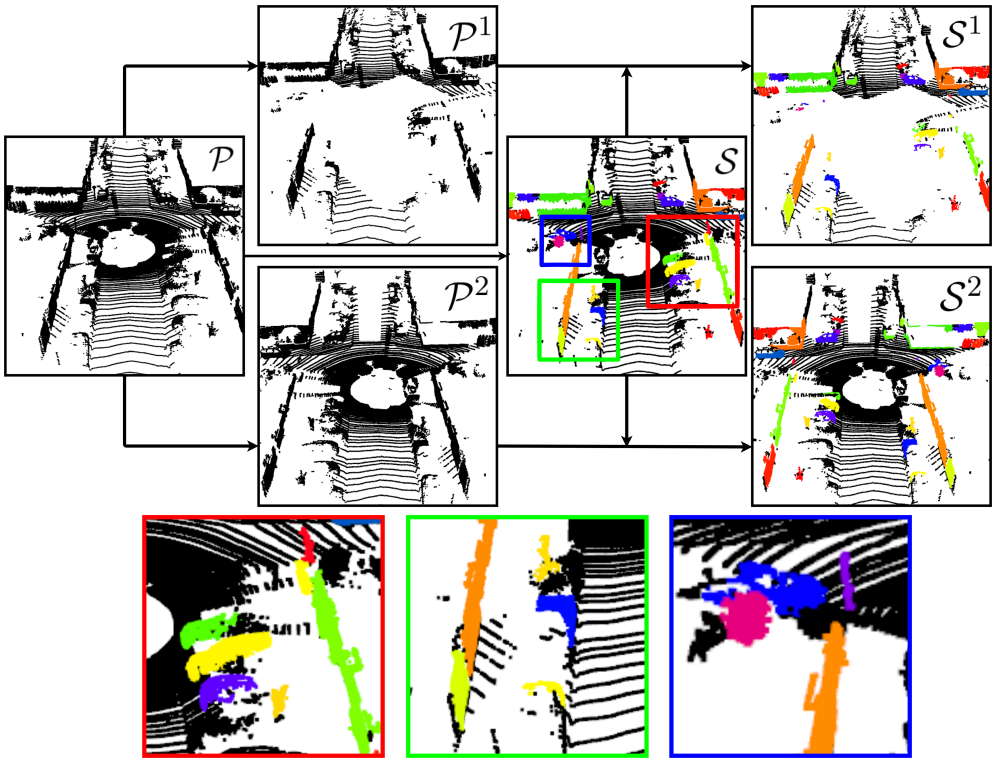


Figure 3.3: From $\mathcal{P}$, we extract the segments $\mathcal{S}$ and generate the augmented views $\mathcal{P}^1$ and $\mathcal{P}^2$. With the augmented views and the segments point indexes, we can extract the set of augmented segments $\mathcal{S}^1$ and $\mathcal{S}^2$. We highlight some of the structures for a better visualization of the segments (solid colored squares).

clouds, we indirectly apply augmentations over the extracted segments, see Fig. 3.3 for an illustration. Since we maintain the point segment assignments via the point indexes during the augmentation, we can easily extract the augmented segments $\mathcal{S}^1$ and $\mathcal{S}^2$ from the augmented views and compute the contrastive loss.

3.1.4 Segment Contrastive Loss

The contrastive loss function is designed to discriminate between the positive and negative pairs. Following He et al. [64], we use the InfoNCE loss [173], together with the online and momentum encoders and the feature bank. In this case, instead of computing the loss with the $2M$ augmented samples in a mini-batch of M as described in Sec. 2.2, a feature bank $\mathcal{K} = \{\boldsymbol{k}_1, \dots, \boldsymbol{k}_K\}$ is defined, to store the last $K \gg M$ samples during training. Given the pair of positive sample l^1_m and l^2_m, the feature vector $\boldsymbol{f}^1_m$ is computed from l^1_m with the online encoder, while the feature vector $\boldsymbol{f}^2_m$ is computed from l^2_m computed with the momentum encoder. At each iteration, the samples computed with the momentum encoder are stored in the feature bank $\mathcal{K}$, implemented as a first-in-first-out queue. The temperature-scaled cosine similarity between the feature vectors $\boldsymbol{f}^p_i$ and $\boldsymbol{f}^q_j$ is

calculated as described in Sec. 2.2:

$$\delta_{i,j}^{p \to q} = \frac{(\boldsymbol{f}_i^p)^\top \boldsymbol{f}_j^q}{\tau}. \tag{3.1}$$

The similarity $\delta_{m,m}^{1\to2}$ between the positive pairs $\boldsymbol{f}_m^1$ and $\boldsymbol{f}_m^2$ is then used to compute the contrastive loss against the K feature vectors stored in the feature bank $\mathcal{K}$ as:

$$\mathcal{L}_m^{1\to2} = -\log \frac{\exp\left(\delta_{m,m}^{1\to2}\right)}{\exp\left(\delta_{m,m}^{1\to2}\right) + \sum_k^K \exp\left((\boldsymbol{f}_m^1)^\top \boldsymbol{k}_k/\tau\right)}. \tag{3.2}$$

In our case, we have two augmented views, $\mathcal{P}_b^1$ and $\mathcal{P}_b^2$, from the anchor point cloud $\mathcal{P}_b$ from the B point clouds in the mini-batch. For simplicity, we will omit the subscript b refering from now on to a sample point cloud $\mathcal{P}_b$ as $\mathcal{P}$. We compute the point-wise features $\mathcal{F}^1$ and $\mathcal{F}^2$ from both augmented views $\mathcal{P}^1$ and $\mathcal{P}^2$ and extract the augmented segments $\mathcal{S}^1$ and $\mathcal{S}^2$ from them. Then, we pass the segments through the projection head to compute the segment-wise feature vectors $\boldsymbol{s}_m^1$ and $\boldsymbol{s}_m^2$ from the M segments in the point cloud $\mathcal{P}$. Therefore, we define our contrastive loss as a segment discrimination computing the similarity in Eq. (3.1) between the segments feature vectors $\boldsymbol{s}_m^1$ and $\boldsymbol{s}_m^2$, computing the contrastive loss as:

$$\mathcal{L}^{1\to2} = \sum_{m=1}^{M} \mathcal{L}_m^{1\to2}, \tag{3.3}$$

where, as in Eq. (3.2), τ is a normalization temperature parameter and $\boldsymbol{k}_k$ are the features from the feature bank $\mathcal{K}$ as before. Note that the number of segments M may vary for different point clouds but is the same between the positive pairs $\mathcal{S}^1$ and $\mathcal{S}^2$.

At the end of each iteration, the feature bank is updated with the segment features from the current batch, maintaining only the last K segments seen by the network.

3.1.5 Pre-Training Strategy

Given one input point cloud $\mathcal{P}$ and its augmented pair $\mathcal{P}^1$ and $\mathcal{P}^2$, we use the backbone to compute the point-wise features $\mathcal{F}^1$ and $\mathcal{F}^2$. We use the entire point cloud during the backbone forward pass to learn the relationship between the segments and the scene. Then, we extract the augmented segments $\mathcal{S}^1$ and $\mathcal{S}^2$ from the point cloud with its point-wise features. Next, we apply dropout and global max-pooling over each segment to compute a feature vector. This feature vectors are then passed through the projection head to get $\boldsymbol{s}_m^1$ and $\boldsymbol{s}_m^2$ and calculate the contrastive loss.

3.2 Experimental Evaluation

In this section, we present our experiments to show the capabilities of our method. We compare our method to the state of the art and show that our approach is more efficient when using fewer labels, can better describe fine-grained structures, and is more transferable between different datasets.

3.2.1 Implementation Details and Experimental Setup

Our experimental setup follows the usual evaluations on contrastive learning methods. First, we pre-train the backbone using our contrastive method. Then, we fine-tune it on different setups for a more in-depth ablation.

Datasets. We used the SemanticKITTI [5, 54] and SemanticPOSS [135] datasets for the self-supervised pre-training and fine-tuning, both collected in an outdoor urban environment. During pre-training we use the scans without the annotations, and later fine-tune the pre-trained models with the scans labels to compare the performance of the different pre-training strategies.

Model architecture. We compare our approach to PointContrast [196] and DepthContrast [209], using their official implementations for pre-training and data pre-processing. We use MinkUNet [34] as the backbone for all the pre-training strategies. The MinkUNet employs sparse convolutions for 3D processing, enabling the processing of large point clouds. As the projection head, we use two linear layers as proposed by Chen et al. [21], which is also used in DepthContrast [209], and we set $p = 0.4$ for the dropout layer.

Pre-training. For pre-training, we use the stochastic gradient descent optimizer with a momentum of 0.9 and set the learning rate to 0.12 and the weight decay to 0.0004. We use a cosine annealing learning rate schedule [101] with a minimum learning rate of 0.00012, following the pre-training scheme used by Zhang et al. [209]. For the class-agnostic segmentation, see Sec. 3.1.2, we set $\nu = 0.25\,\text{cm}$, $\xi = 20$ and $\zeta = 50$, which are the RANSAC distance threshold, the minimum number of points per segment and the maximum number of segments extracted per point cloud, respectively. For all the methods, we randomly sample 20,000 points from the entire point cloud after data augmentation to limit the number of points per sample, and we pre-train the backbone for 200 epochs. Given the size of SemanticKITTI, we use $K = 8{,}152$ for the feature bank of the DepthContrast [209] method. For our approach, we set it to $K = 65{,}536$, with the temperature parameter $\tau = 0.1$. It is important to highlight that we have a bigger feature bank since we save the M segments features extracted from the point cloud instead of the complete point cloud. For our method and DepthContrast [209], we use batch size 8, and for PointContrast [196], we use batch size 16 to maintain the batch size 4 per GPU as in the original paper. Given the Point-

Contrast [196] method characteristics, it is known that batch size change may affect the approach performance. However, we used the same cluster with four NVIDIA GTX1080TI 12 GB GPUs for all the methods to make the comparisons as fair as possible.

Fine-tuning. For the semantic segmentation experiments, we use stochastic gradient descent with momentum of 0.9, learning rate of 0.24, and weight decay of 0.0004. We also use a cosine annealing scheduler with a minimum learning rate equal to 0.00024, following the semantic segmentation setup used by Tang et al. [168]. We set the batch size to 2 and randomly sample 80,000 points per point cloud during training. For the object detection experiment, we use PointRCNN [160] as the base detector, and the same 5% label set used by Zhang et al. [209] for a fair comparison. For all the experiments, we use one NVIDIA RTX2080 Super 8 GB GPU. The experimental results are collected in the validation sequences from both datasets, i.e., sequence 8 for SemanticKITTI and sequences 4 and 5 for SemanticPOSS. In both datasets, the validation sequences were not used for pre-training or fine-tuning.

3.2.2 Fine-Tuning for Semantic Segmentation

In this evaluation, we want to measure the semantic information learned by the network during pre-training. We fine-tune the pre-trained network to semantic segmentation on SemanticKITTI and SemanticPOSS after pre-training on SemanticKITTI without the labels.

3.2.2.1 Label Efficiency

The first experiment evaluates the robustness of the features learned from the different representation learning methods by fine-tuning it to the semantic segmentation task using the SemanticKITTI dataset. Since it is a larger dataset, we can divide it into different label percentage regimes and increasingly compare it to support our first claim. We define five different label percentage regimes, i.e., using 0.1%, 1%, 10%, 50%, and 100% of the labeled training data, where we select a fixed subset of scans from the dataset given the regime percentage. Every subset is chosen from the entire dataset, such that all the classes are present. When training with fewer labels, the total number of training iterations will decrease accordingly. Therefore, we increase the number of epochs as we reduce the number of training scans to achieve convergence at every label regime (see Tab. 3.1).

Fig. 3.4 gives a qualitative comparison between the different contrastive methods and the network without pre-training when training with 0.1% of the labels. From the top views, we observe that the network trained from scratch could not

Table 3.1: Number of training epochs used for different label regimes on SemanticKITTI.

Label regime	0.1%	1%	10%	50%	100%
Number of epochs	300	120	40	20	15

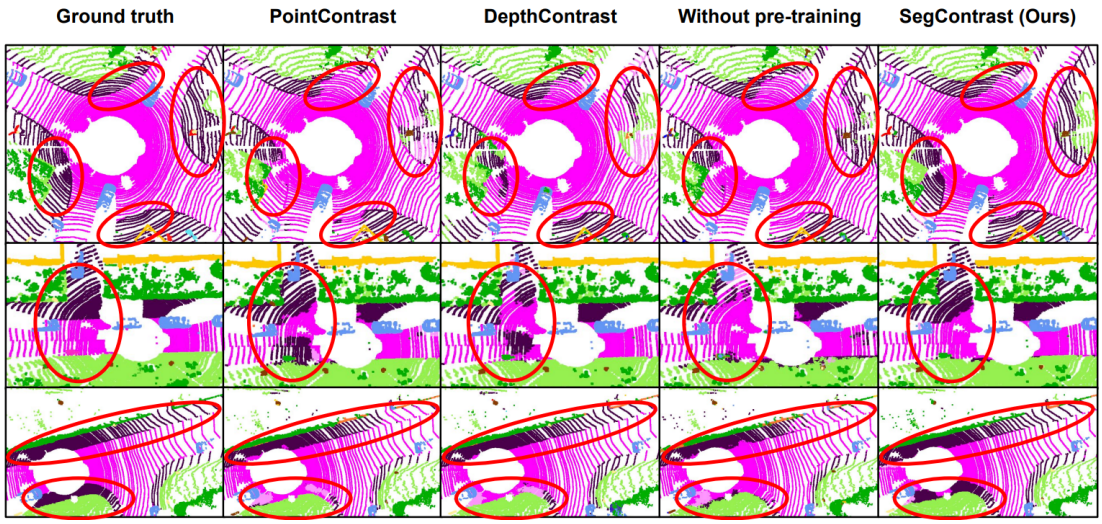


Figure 3.4: Qualitative results on three different validation scans (rows). We show results of the networks fine-tuned with only 0.1% of the labels that are pre-trained with different contrastive methods or trained from scratch (without pre-training). We compare the results from PointContrast [196], DepthContrast [209] and our method, SegContrast. With our pre-training, SegContrast, the fine-tuned network can better distinguish the different structures, i.e., sidewalk and road, as shown by the highlighted areas (solid red circles).

learn much structural information, leading to a noisy division between different structures. This same noisy division is seen in the other contrastive methods. Our approach better learns the structural information of the point clouds, which leads to better boundaries between different structures. This improvement can also be seen in more fine-grained classes, e.g., pole and traffic signs, shown in Fig. 3.1.

Tab. 3.2 shows the results on the different label regimes. All methods perform better than training from scratch, i.e., without pre-training, and the gap between the results diminishes as the number of scans used for training increases. Our method is better at lower label regimes. As the amount of labels increases, DepthContrast [209] achieves a comparable performance. At the full label training, all the methods converge to a similar result as training from scratch. This is expected since the data used for pre-training and fine-tuning are the same. The evaluation shows that our approach performs better when using fewer labels compared to the other contrastive learning methods, being able to better describe

Table 3.2: Fine-tuning at different label regimes on SemanticKITTI for semantic segmentation using mean intersection-over-union (mIoU).

	mIoU [%] ↑				
Method	0.1%	1%	10%	50%	100%
Scratch	25.59	41.70	53.87	58.34	59.63
PointContrast [196]	28.52	43.40	53.79	57.30	59.77
DepthContrast [209]	33.51	46.41	**56.29**	**58.54**	59.88
SegContrast (Ours)	**34.78**	**47.41**	55.21	58.33	**60.53**

the different structures in the point cloud.

3.2.2.2 Linear Evaluation

A typical experiment used for image-based contrastive methods is the so-called linear evaluation. This evaluation freezes the pre-trained backbone and trains only a linear layer on top of it to compare how well the feature representation can describe the different classes, even without fine-tuning the whole network. This evaluation is done to analyze how well the pre-trained network can describe the different objects in the scene by only fine-tuning a classifier on top of the pre-trained network features. Our experiment follows the same setup, the pre-trained backbone weights are frozen, and we train only a linear segmentation head. The result of this experiment supports our claim that our method can learn a feature representation that better describes fine-grained structures.

Tab. 3.3 displays the results of the linear evaluation over the different self-supervised contrastive methods and over the randomly initialized network without pre-training for a lower bound on the performance. DepthContrast [209] shows the worst performance in this evaluation, which indicates that the method cannot learn a feature representation as descriptive as the other methods. PointContrast [196] shows a better performance, since the method uses a point-wise contrastive loss. Furthermore, when looking at the underrepresented classes, e.g., parking, trunk, fence, pole or traffic sign, our method outperforms the other methods. PointContrast [196] achieves a higher mIoU by learning the more represented classes, e.g., road, building. In contrast, our method seems better suited to represent the fine-grained classes.

3.2.2.3 Generalization

In the third experiment, we evaluate the generalization of our learned feature representation. The results support our claim that our method is more transferable

Table 3.3: Linear evaluation mIoU and per-class intersection-over-union (IoU) pre-trained and evaluated on SemanticKITTI.

	IoU [%] ↑											
Method	road	sidewalk	parking	building	car	vegetation	trunk	terrain	fence	pole	traffic-sign	mIoU
Without pre-training	30.61	2.72	0.0	5.56	0.02	42.39	0.0	0.05	0.24	0.0	0.0	4.29
PointContrast [196]	**70.61**	**37.00**	0.0	**83.37**	**88.93**	**75.10**	33.26	**53.93**	8.66	5.55	0.0	**24.02**
DepthContrast [209]	40.68	5.35	0.0	54.81	59.43	65.30	24.55	17.56	13.50	20.47	0.0	15.91
SegContrast (Ours)	59.02	30.83	**0.5**	71.68	80.23	73.13	**34.56**	37.51	**15.68**	**40.59**	**0.19**	23.36

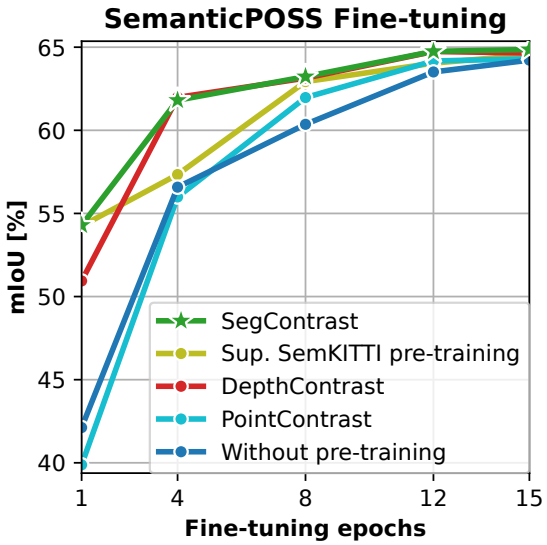


Figure 3.5: Comparison between contrastive pre-trained networks fine-tuned to the SemanticPOSS dataset with the network trained from scratch. At the beginning of training, our method shows a comparable performance to supervised pre-training on SemanticKITTI, evidencing that our learned feature representation is as robust as the supervised pre-training.

between different datasets. We use SemanticKITTI for unsupervised pre-training of the backbone with different contrastive methods or use the commonly used supervised pre-training. Then, we fine-tune the differently pre-trained networks on the SemanticPOSS dataset and compare their performance. SemanticPOSS is smaller than SemanticKITTI, which gives a setup aligned to standard image-based setting, where a larger dataset, e.g., ImageNet [38], is used for pre-training and then fine-tuning is performed on smaller datasets.

Fig. 3.5 displays the network performance in different training epochs during fine-tuning. Here, we see a comparable performance of the unsupervised and supervised pre-training. As shown, after only one training epoch, our method shows a considerably better performance than the other contrastive methods and achieves the same performance as the supervised pre-training. Even though our approach does not use any labels, our method learns a feature representation comparable with the supervised pre-training on SemanticKITTI. This result suggests that our method can learn a more general feature representation than previous methods and it seems to be more suitable for fine-tuning on a different dataset.

Tab. 3.4 displays the results of the different contrastive pre-training methods and the supervised pre-training on SemanticKITTI. The pre-training methods improve the network performance at the lower label regimes, but previous methods cannot surpass the supervised pre-training. However, our method outperforms both self-supervised and supervised pre-training at all label regimes. This exper-

Table 3.4: Pre-training with SemanticKITTI and fine-tuning on SemanticPOSS with different label regimes showing mIoU.

	mIoU [%] ↑				
Method	0.1%	1%	10%	50%	100%
Without pre-training	33.09	43.14	57.27	63.34	64.22
Supervised pre-training	42.88	54.40	60.22	64.26	64.54
PointContrast [196]	31.78	49.06	56.49	62.93	64.30
DepthContrast [209]	41.94	52.66	59.27	64.09	64.65
SegContrast (Ours)	**43.69**	**55.21**	**60.33**	**64.58**	**64.86**

Table 3.5: Linear evaluation on SemanticPOSS with pre-training on SemanticKITTI using mIoU and accuracy.

Method	mIoU [%] ↑	Accuracy [%] ↑
Without pre-training	6.32	42.46
PointContrast [196]	24.79	72.55
DepthContrast [209]	16.41	63.09
SegContrast (Ours)	**31.51**	**73.51**

iment indicates the robustness of the learned representation when fine-tuning to a different LiDAR data, exceeding even the supervised pre-training.

In Tab. 3.5, we show the linear evaluation over the SemanticPOSS with the network pre-trained on SemanticKITTI. DepthContrast [209] shows the lowest performance in this evaluation, showing that the feature representation is less descriptive when transferring to a different dataset compared to the other methods. Our method surpasses the other approaches, outperforming them by a large margin. These results suggest that our approach can learn a point cloud representation transferable across different datasets and LiDAR sensors.

Finally, we evaluate the self-supervised pre-training on SemanticKITTI and SemanticPOSS, fine-tuning it on SemanticPOSS. In Tab. 3.6, we show both the linear evaluation and the fine-tuning. As we can see, the performance of the pre-training on SemanticKITTI is better on both experiments. We can see that we obtain a performance gain using a large dataset for self-supervised pre-training. This also highlights the generalization of the feature representation achieved by our method. The pre-training on SemanticKITTI performed better than on SemanticPOSS, even though both datasets were collected with a different LiDAR sensors and different sensor mounting positions.

Table 3.6: Fine-tuning perfomance in mIoU on SemanticPOSS with pre-training on SemanticKITTI and SemanticPOSS using our method.

	mIoU [%] ↑	
Dataset	Linear evaluation	Fine-tune
Without pre-training	6.32	64.22
SemanticPOSS	29.68	64.31
SemanticKITTI	**31.51**	**64.86**

3.2.3 Fine-Tuning for Object Detection

For a more complete evaluation, in this experiment we also compare the methods fine-tuning to object detection. Tab. 3.7 presents our results on the object detection task on the KITTI dataset [54]. In this experiment, we use the same 5% labels set used by Zhang et al. [209]. With 100% of labels, the comparison between the network without pre-training and the pre-trained models shows no significant difference. Since we use KITTI for pre-training and fine-tuning, this is expected since no new data is seen during pre-training. With 5% of labels, it is possible to show the gain of the pre-trained network. Except for the pedestrian class, all the methods outperform the network without pre-training by a large margin. In the car and cyclist classes, our method surpasses previous methods in almost all the difficulties. These results indicate that our approach can learn a robust feature representation, suitable for fine-tuning to different downstream tasks and outperforming previous methods when using fewer labels.

3.3 Related Work

Pre-training a neural network is often a better alternative to random network weights initialization. Different pre-training strategies have been studied in the computer vision field as a way to train neural networks to learn strong representations in a self-supervised manner, reducing the need for labels. In this section, we discuss related self-supervised representation learning studies in the image and 3D point cloud domains, aiming to achieve both global and local representations.

Self-supervised representation learning methods define a pretext task to train a network without any labels. Prior works [13, 59, 60, 74, 109, 193] defined reconstruction pretext tasks to learn useful representations, such as solving jigsaw puzzles [59, 74, 109] or reconstructing masked data [13, 60, 193]. To learn more descriptive representations, contrastive learning was defined as a discriminative task optimized directly at the feature level [21, 22, 23, 58, 64, 173, 205]. Those methods take advantage of data augmentation to generate two distinct versions

Table 3.7: Fine-tuning with 5% and 100% of labels on KITTI for object detection showing mAP.

	Car			Pedestrian			Cyclist		
	mAP [%] ↑								
	100% labels								
Method	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
Without pre-training	91.46	81.77	**79.92**	68.29	**63.76**	58.07	90.65	73.35	70.05
PointContrast [196]	91.44	**81.83**	79.85	**69.81**	63.56	**58.13**	90.48	73.87	69.81
DepthContrast [209]	91.79	81.48	79.90	66.39	60.44	55.41	**92.45**	74.69	**71.18**
SegContrast (ours)	**91.97**	80.17	79.79	67.46	62.71	56.42	90.71	**75.38**	70.79
	5% labels								
Method	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
Without pre-training	79.33	63.83	58.92	69.34	61.72	54.49	84.48	61.82	57.80
PointContrast [196]	80.83	66.52	61.70	69.75	**62.77**	**55.71**	**88.82**	67.53	63.17
DepthContrast [209]	82.32	67.61	62.59	**69.81**	62.37	55.13	87.52	66.91	62.50
SegContrast (ours)	**82.62**	**68.27**	**64.41**	69.10	61.59	54.49	88.70	**68.48**	**64.35**

of an anchor sample, creating a positive pair. Then, the network is trained to learn a feature representation that maximizes the similarity between this positive pair and minimizes the similarity with other so-called negative samples. The representation learned without labels during the contrastive training can then be fine-tuned to specific downstream tasks, often achieving better performance in those tasks. With this contrastive pre-training strategy, recent works have been able to improve the performance of various image-based classification methods using fewer or even no labels [21, 22, 23, 58, 64, 173, 205] compared to a network with randomly initialized weights trained with task-specific labels.

More recent contrastive learning works focus on fine-grained tasks, such as semantic segmentation [174, 179] or object detection [66]. In this case, the discriminative pretext task remains the same. However, the network is also optimized for discriminating specific regions in the image. The image is divided into segments using a class-agnostic segmentation approach, and the contrastive loss is applied over the features extracted from the image segments [66, 174]. Other methods proposed using augmented positive pairs to optimize the network at the pixel level [179, 192]. Such works improve the network performance for more fine-grained tasks using the contrastive loss for pre-training [66, 174, 192] or as an auxiliary supervised loss [137, 179].

3D self-supervised representation learning has been studied, given the compelling results of contrastive learning methods on 2D image data. Early works focus on learning representations from single-object 3D point clouds [48, 87, 108, 155]. Such works directly apply the contrastive learning strategy from the image domain to 3D point clouds of objects. Other works aim at more complex scene-level 3D perception tasks [71, 196, 209]. These studies focus on learning representations from 3D point clouds, targeting point-level perception tasks, such as semantic segmentation [168, 218] and object detection [24, 142]. As a direct application of the image domain contrastive strategy, Zhang et al. [209] propose a global contrastive loss for 3D point cloud data. As in the image domain, a pair of augmented views is generated for each point cloud, i.e., the positive pair. Then, the other augmented scans are used as negative samples, and the contrastive loss is computed over the extracted features from these scans. This scan-wise discrimination trains the network to learn a global representation from the 3D point cloud of the scene. Such a global descriptor ignores the local information around the points in the scene. This lack of point-level representation impacts the application of the pre-trained network to fine-grained 3D perception tasks.

Differently, Xie et al. [196] define the contrastive objective as a point-level discrimination task. Given a sequence of LiDAR scans, the relative poses between scans are used to determine corresponding points between consecutive point clouds. The contrastive loss is defined as a point-wise objective, with the corre-

sponding points as positive pairs and the non-corresponding points as negatives. In contrast to Zhang et al. [209], the network in this case is optimized with point-wise features. Still, this point-wise contrastive loss ignores higher-level information about the scene despite learning a point-level representation. Hou et al. [71] add more context to the point-wise contrastive pre-training by dividing the scene into different spatial partitions. Then, together with the contrastive loss, the network is optimized to predict the corresponding partition for each point. In this case, spatial information is embedded with the point-wise features together with the local information learned via the contrastive objective.

Despite promising results, those works either focus on global context, i.e., scan-level features, or local context, i.e., point-level features. Given the complexity of outdoor 3D point clouds, neither global nor local representations account for the distinct objects and structures present in the scene. In this chapter, we detail our proposed contrastive representation learning method for 3D LiDAR data used in autonomous driving. We extract class-agnostic segments from the point cloud and propose a contrastive loss to be applied over the extracted segments. Distinct from prior work, our representation learning method learns more contextualized information by discriminating between segmented structures in the point cloud, learning a more robust and descriptive embedding space.

3.4 Conclusion

In this chapter, we presented a novel representation learning approach for LiDAR point clouds in outdoor environments. Our approach exploits the characteristics of outdoor LiDAR data to extract class-agnostic segments and applies the contrastive loss over these segments. Our pre-training strategy enables the training of a neural network to learn fine-grained object-level information without labels, thereby decoupling the scaling of training data from the data annotation. We evaluated our strategy on different datasets and provided comparisons with other state-of-the-art feature representation learning approaches. The experiments suggest that our approach can learn a more robust feature representation than previous works, outperforming them on different downstream tasks. Furthermore, our self-supervised feature representation is more transferable when fine-tuning on a different target dataset, outperforming even the supervised pre-training.

We assessed our method with different evaluations. First, our approach demonstrated better performance than prior methods when fine-tuned to the various perception tasks, particularly in the limited label regime. Additionally, we evaluated the learned representation using the so-called linear evaluation, which fine-tunes only a semantic segmentation MLP on top of the self-supervised learned representations. In this evaluation, our approach demonstrated superior

performance on underrepresented classes compared to prior global- and local-level methods. Finally, we evaluated the transferability of the learned representations by fine-tuning the pre-trained network on a different dataset than the one used for pre-training, which was collected with the same LiDAR sensor. In that case, our method achieved better performance, even when compared to supervised pre-training, where the network is directly pre-trained for the same perception task.

In the next chapter, we discuss how to extend this segment discrimination pretext task to also consider temporal information. The data augmentations applied to the point clouds described in this section may not portray the view changes of an object with respect to its position changes over time during the data collection process. This gap between virtual and real data transformations can hinder the representation learned by the network. By detecting corresponding segments within a sequence of LiDAR point clouds, we enable the network to learn stronger representations invariant to the objects' transformations inherent in the LiDAR sensor's sparse data.

Chapter 4

Temporal Representations from Unlabeled 3D Data

Segment-level contrastive learning as the method discussed in the previous chapter enables the training of neural networks to learn object-level representations that we can fine-tune to different perception tasks, reducing the requirement for data annotation. Still, given the sparsity of the LiDAR point clouds, the way an object is depicted in the point cloud can change drastically depending on the position of the object with respect to the sensor. This can impact the representation learned by the network, as no direct information about those view changes is given to the model during training. The data augmentation process used by the contrastive training tries to address this by generating different versions of the same object via similarity transformations applied to the point clouds. However, the augmentations applied during training might be insufficient to replicate the changes that occur in the real world. When considering individual LiDAR scans, such an object's aspect change can hinder the network training. However, we can train the network to learn stronger representations that rely not only on virtual data augmentations but also on real augmentations by matching those distinct views of an object inherent to sequences of LiDAR scans.

Sequential data is leveraged in the image domain to learn more robust representations by optimizing the network to extract features invariant to the object changes within consecutive images. To achieve this, videos are used to train a network to match representations within consecutive frames, rather than considering only individual images [8, 77, 134, 143]. Similar to the standard image contrastive learning approach, those methods were trained to distinguish between negative and positive pairs. However, positive pairs are sampled from consecutive frames, trying to embed visual and geometric information from the changes within the video frames. Some approaches have been proposed to extend the

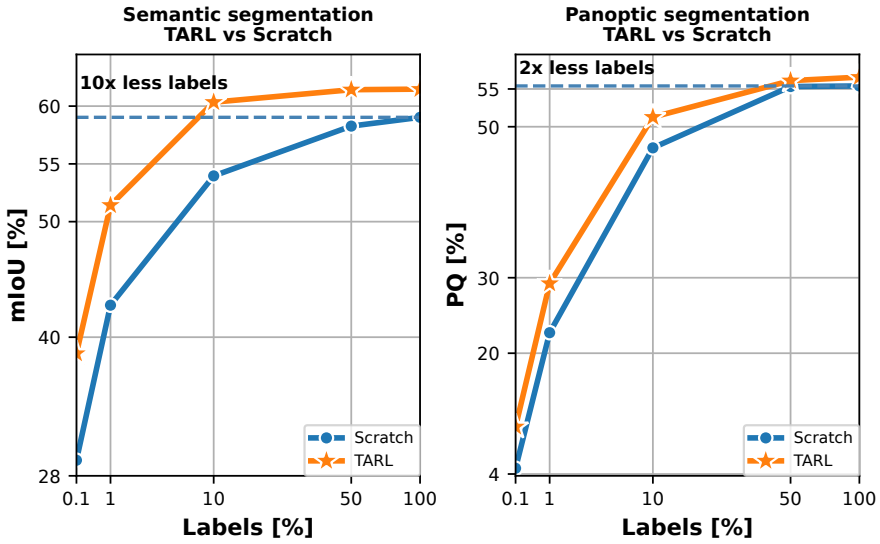


Figure 4.1: Our pre-training (TARL) can reduce the amount of necessary labels during fine-tuning on SemanticKITTI [5, 54]. Our method requires only 10% of labels to surpass the network trained from scratch with the full dataset for semantic segmentation. For panoptic segmentation, our method requires only half of the labels.

frames' discrimination to more complex pretext tasks. For example, classifying whether two random clips from the same video overlap or not [77], separating between stationary and non-stationary features [8], or defining positive and negative pairs regarding motion cues extracted from the video frames [40, 92]. Those video-based contrastive learning methods often lead to stronger representations learned by the network, as they embed both visual and temporal aspects.

As for images, the temporal correspondences of 3D LiDAR frames can be exploited to learn representations invariant to view changes. While collecting data, a vehicle equipped with a LiDAR sensor moves around the environment, collecting consecutive 3D point clouds. If we consider global-context contrastive learning methods [213], the information about the order in which the point clouds were collected is sufficient to also embed temporal information into the network training process, as in video contrastive learning methods. However, for the segment-level contrastive learning described in the previous chapter, we also need segment-wise correspondences between frames to define the positive and negative segment pairs. Then, segment-wise discrimination can be employed to train the network to learn representations that incorporate geometric information about the objects in the scene and their view changes over time, achieving stronger geometric representations invariant to changes in the objects' characteristics.

In this chapter, we extend the segment-wise discrimination method described in the previous chapter to the temporal domain. We exploit the sequence of point clouds collected with the LiDAR sensor to aggregate consecutive scans based on

their relative poses, inspired by video contrastive learning methods. Then, the unsupervised segments extraction defined in the previous chapter is applied to define segments of objects over time. Temporal views of the same object over the individual scans are defined by maintaining the mapping between the points in the aggregated point cloud and the individual LiDAR scans. The segment-wise contrastive learning is then applied over the temporal views of segments in the scene, learning temporally consistent representations that are invariant to the object aspect changes inherent in LiDAR data. As in the previous chapter, we evaluate our proposed temporal representation learning method by fine-tuning the trained model to different downstream tasks. Our proposed method demonstrates improved performance compared to the temporally agnostic method described in the previous section and to other self-supervised representation learning methods, requiring only a fraction of the labels as illustrated in Fig. 4.1.

4.1 Temporal Consistent 3D LiDAR Representation Learning

The main contribution of this chapter is a temporally-consistent representation learning method for 3D LiDAR data designed for autonomous driving data. We exploit the vehicle motion to extract different views of the same objects across time. Then, we train a network to maximize the similarity between point-wise features from the same objects in the point clouds collected at different points in time, embedding the objects' dynamics. This temporally-consistent features optimization leads to a stronger representation learned by the model, improving performance on different perception tasks while requiring less annotated data. In summary, the key contributions of this chapter are (i) a self-supervised pre-training method for 3D LiDAR data able to learn temporally-consistent representations, (ii) which achieves better performance than previous state-of-the-art methods on different downstream tasks, (iii) requires only 10% of labels to surpass the network trained from scratch for semantic segmentation using the complete SemanticKITTI training set (Fig. 4.1), and (iv) provides representations more suited for transfer learning than supervised pre-training, achieving better performance when fine-tuning to a different dataset.

4.1.1 Our Approach

In this chapter, we propose a new self-supervised representation learning method able to learn temporally consistent representations for LiDAR data obtained in the context of autonomous driving. Our approach requires only unlabeled point clouds and their corresponding relative poses. Pose information is usually readily

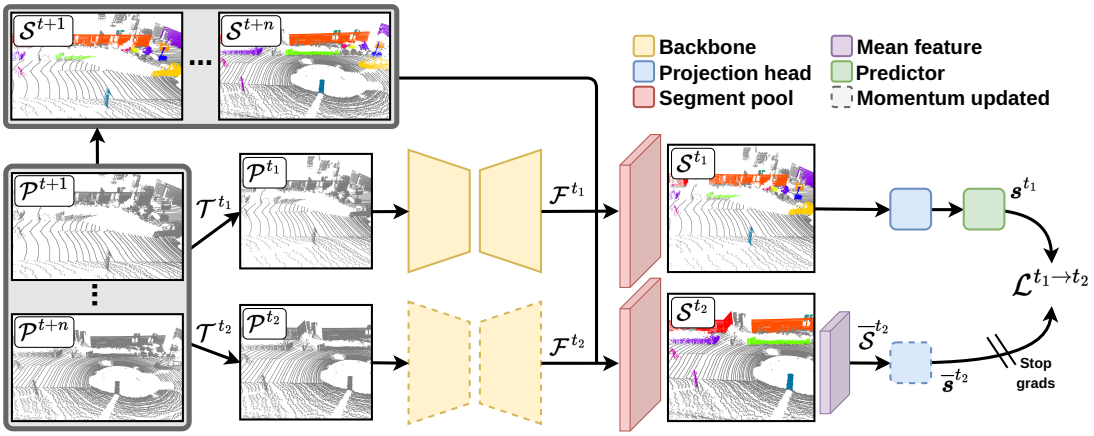


Figure 4.2: Overview of our approach. We aggregate n point clouds and extract segments $\mathcal{S}$ from it by removing the ground and clustering the remaining points. Then we sample two point clouds $\mathcal{P}^{t_1}$ and $\mathcal{P}^{t_2}$, and apply random augmentations $\mathcal{T}^{t_1}$ and $\mathcal{T}^{t_2}$ to them. Next, we compute point-wise features for each scan and list its segments $\mathcal{S}^{t_1}$ and $\mathcal{S}^{t_2}$ with their corresponding point features. For $\mathcal{S}^{t_2}$ we compute the mean feature vector for each segment and compute the segments target embeddings $\overline{\boldsymbol{s}}^{t_2}$ with a projection head. For $\mathcal{S}^{t_1}$ we use a point-wise projector followed by a predictor to identify for each segment point features $\boldsymbol{s}^{t_1}_{m,p}$ the corresponding target embedding $\overline{\boldsymbol{s}}^{t_2}_{m}$ by minimizing the loss $\mathcal{L}^{t_1 \to t_2}$.

available by means of GPS/IMU, odometry approaches [37, 178], or SLAM systems [6, 26], thus, this is not a limitation in practice. As in the previous chapter, we use a siamese network scheme with an online and a momentum network, illustrated in Fig. 4.2. The pipeline consists of extracting objects as coarse segments viewed at different times over an interval of scans. Then, point-wise features are computed with the backbone and a transformer encoder projector. Finally, we perform an implicit clustering to put points from different views of the same object together. In the following subsections, we explain the individual steps of our method.

4.1.2 Temporal Objects Views

Instead of only using data augmentation to generate artificial views of one object, we exploit the vehicle motion to extract real-world object segments viewed from different perspectives. Given the vehicle motion and the properties of LiDAR sensors, one object can have different appearances depending on its position relative to the sensor. Such changing appearance is usually a problem when fine-tuning the model to data collected with different sensors [2, 75, 88, 203]. However, we exploit these properties in our favor to extract natural augmented views of one object. Previous works [29, 71, 76, 128, 196, 213] relied only on data augmentation to generate pairs of one scan, such as rotation, translation and scaling. Instead,

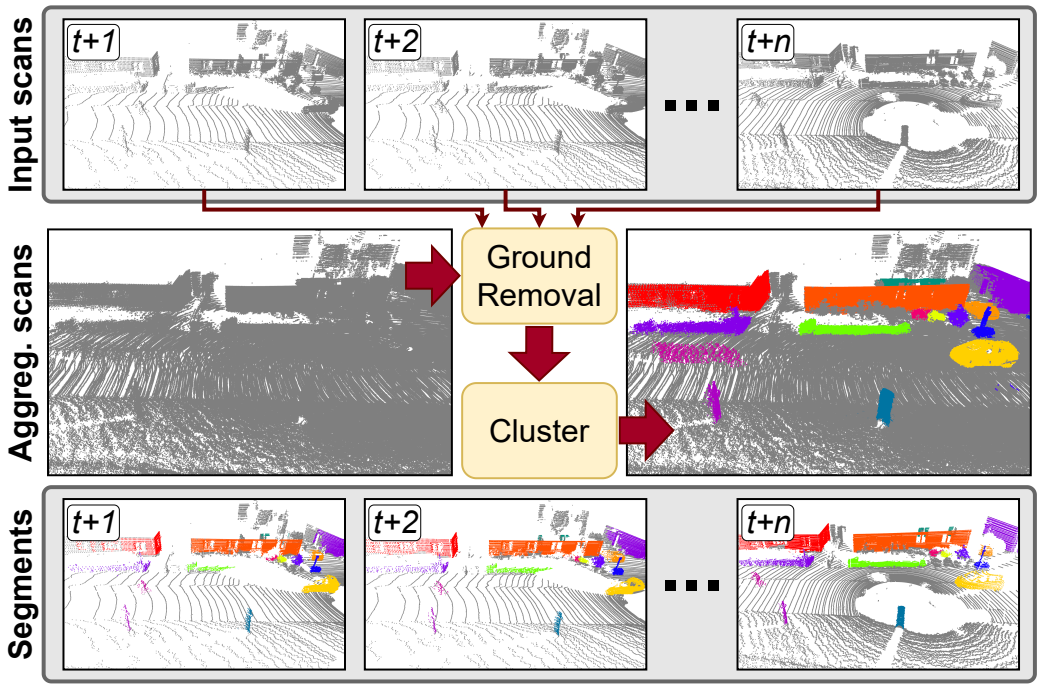


Figure 4.3: Temporal segments generation. We aggregate n point clouds from different times and extract coarse segments from the aggregated point cloud in an unsupervised manner. We keep the point indices mapping to extract views at different times from one object as augmented pairs.

we extract object views collected from different perspectives by the LiDAR sensor while the vehicle navigates through the environment. We then apply augmentations over the associated scans captured at a different time and train the network to learn a common representation for the object different views.

We define a point cloud at time t as $\mathcal{P}^t = \{\boldsymbol{p}_1^t, ..., \boldsymbol{p}_R^t\}$ with points $\boldsymbol{p}_r^t \in \mathbb{R}^3$. To extract coarse object segments in an individual scan, we first separate $\mathcal{P}^t$ into ground $\mathcal{G}^t$ and non-ground points $\tilde{\mathcal{P}}^t$ in an unsupervised manner using the method proposed by Lim et al. [96], where $\mathcal{P}^t = \tilde{\mathcal{P}}^t \cup \mathcal{G}^t$ and $\tilde{\mathcal{P}}^t \cap \mathcal{G}^t = \emptyset$. Afterwards, we cluster the non-ground points $\tilde{\mathcal{P}}^t$ into M segments $\mathcal{S}^t = \{\mathcal{S}_1^t, ..., \mathcal{S}_M^t\}$ using HDBSCAN [15], where $\tilde{\mathcal{P}}^t = \bigcup_{m=1}^{M} \mathcal{S}_m^t$.

To map different segment views at different times, we define an interval of n scans from which the views of the object will be extracted. Those scans are transformed to a common global coordinate frame to be then aggregated. The global scan poses can be easily acquired with GPS/IMU, SLAM systems [6, 26], or LiDAR odometry [37, 178]. The aggregated point cloud $\mathcal{P}$ is given by $\mathcal{P} = \{\mathcal{P}^{t+1}, \mathcal{P}^{t+2}, ..., \mathcal{P}^{t+n}\}$. Similarly, we aggregate the individual ground segmentation labels $\mathcal{G} = \{\mathcal{G}^{t+1}, \mathcal{G}^{t+2}, ..., \mathcal{G}^{t+n}\}$ and get the aggregated non-ground points $\tilde{\mathcal{P}} = \{\tilde{\mathcal{P}}^{t+1}, \tilde{\mathcal{P}}^{t+2}, ..., \tilde{\mathcal{P}}^{t+n}\}$. As in the individual scan case, we cluster $\tilde{\mathcal{P}}$ to get the M segments $\mathcal{S} = \{\mathcal{S}_1, ..., \mathcal{S}_M\}$. By keeping the point index mapping from

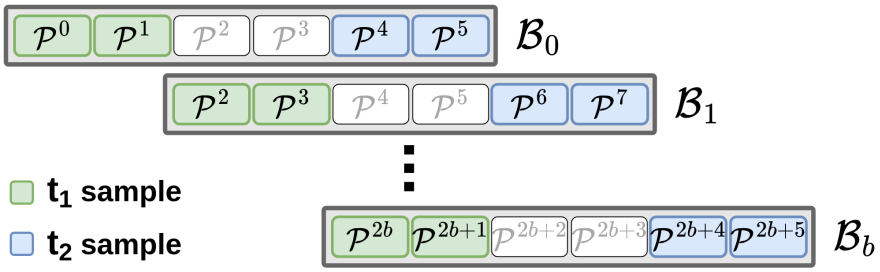


Figure 4.4: Temporal batch example, for an interval of $n = 6$ point clouds $\mathcal{P}$. We divide a sequence of N scans into batches $\mathcal{B}_b$ where we sample t_1 from the beginning and t_2 from the end of the interval. Then, the next batch starts in the middle of the previous interval.

the aggregated point cloud to the individual n scans, we identify the n segments of the same object viewed at different times as $\mathcal{S}_m = \{\mathcal{S}_m^{t+1}, ..., \mathcal{S}_m^{t+n}\}$. We then list the temporal views from each of the M segments in the aggregated point clouds as $\mathcal{S}_{1:M} = \{\mathcal{S}_1^{t+1}, ..., \mathcal{S}_1^{t+n}, ..., \mathcal{S}_M^{t+1}, ..., \mathcal{S}_M^{t+n}\}$.

To properly segment views of both static and moving objects, we assume a LiDAR at the commonly used frequency of 10 Hz, where the scan overlap is usually enough to cluster together also moving objects, as depicted in Fig. 4.3. With this procedure, our augmented pairs are views of the same object at different times from static and moving objects. By maximizing the similarity between the object views at different times, the network needs to learn a more general representation consistent across time and embed the object dynamics. In Sec. 4.2.5, we provide an experimental comparison between the pre-training using the temporal object views as augmented pairs and with augmented pairs generated only with data augmentation to validate the use of temporal views.

4.1.3 Temporal Batch

During pre-training, we need to sample a subset of n consecutive scans from the training sequence to be aggregated and extract the temporal object views as explained in Sec. 4.1.2. This sample will be used during pre-training to learn a temporal consistent representation for the segmented objects. For a sequence of N point clouds, we divide it into batches $\mathcal{B}$ with intervals of n scans. During the pre-training forward pass, we sample two scans at different times, t_1 and t_2. Since the goal of this temporal sampling is to have different views of the same object, we enforce this by sampling t_1 from the beginning and t_2 from the end of this n scans interval such that $t_1 < \frac{n}{3}$ and $t_2 \geq 2\frac{n}{3}$. In this case, the scans between $\frac{n}{3} \leq t < 2\frac{n}{3}$ would never be sampled. To also process those scans, the next batch starts at this unseen interval. Therefore, for a batch sample $\mathcal{B}_b$ the n scans from the sequence of N point clouds are $\mathcal{B}_b = \{\mathcal{P}^t | b\frac{n}{3} \leq t < b\frac{n}{3} + n\}$.

In Fig. 4.4, we show how the batches would look like in an example with scans interval $n = 6$. Even though we do not sample t from $\frac{n}{3} \leq t < 2\frac{n}{3}$, we still need to aggregate and cluster these scans to correctly segment dynamic objects and match their correspondences between the scans. With this sampling scheme, we guarantee that the objects views will be distant in time and still all the data will be seen during pre-training.

4.1.4 Implicit Clustering

With our coarse object segments $\mathcal{S}$, we have the prior that points from the same segment should be semantically similar. Given that we rely on a set of object segments, one straightforward way to distinguish the learned embeddings is to cluster together point features from the same object. Differently from the previous chapter, we want to maximize the similarity of the point-wise feature vectors rather than only maximizing the similarity between segment-wise feature vectors. Recent methods propose an online clustering scheme to separate samples around a fixed number of prototype learnable cluster centers [17]. In our case, we aim at clustering point features from the same object close together but in a more straightforward way with an implicit clustering.

Given the temporal sampled pair $\mathcal{P}^{t_1}$ and $\mathcal{P}^{t_2}$ from a batch sample $\mathcal{B}_b$, we compute point-wise features $\mathcal{F}^{t_1}$ and $\mathcal{F}^{t_2}$ with the online and momentum encoders respectively. As the target embedding, we list from $\mathcal{F}^{t_2}$ the set of M segments $\mathcal{S}^{t_2}$. We compute for each segment a mean representation from its point-wise features, and project this embedding with a self-attention transformer encoder as a projection head to get the M target mean feature vectors $\overline{\boldsymbol{s}}^{t_2} \in \mathbb{R}^{M \times N^{\text{out}}}$ where N^{out} corresponds to the feature dimension.

The segments $\mathcal{S}^{t_1}$ are listed from the point-wise features $\mathcal{F}^{t_1}$. For $\mathcal{S}^{t_1}$, we do not compute mean embeddings but keep the features at point level, using the transformer encoder to compute point-wise intra-class correspondences. To deal with the attention mechanism large memory requirement, we set a maximum number of P points per segment, which are randomly sampled. After sampling P points for each segment, we input the segments $\mathcal{S}^{t_1}$ point-wise features to the transformer projection head, followed by another self-attention transformer encoder as a predictor. We then get for each segment the P point-wise feature vectors $\boldsymbol{s}^{t_1} \in \mathbb{R}^{M \times P \times N^{\text{out}}}$.

With the segment target mean representations $\overline{\boldsymbol{s}}^{t_2}$ and the predicted point-wise feature vectors $\boldsymbol{s}^{t_1}$ for each segment, we compute the loss to minimize the differences between the point features and the corresponding segment mean representation. For each point p from a segment m, we compute the temperature-scaled cosine similarity $\delta_{m,p,k}^{t_1 \to t_2}$ between the normalized point-wise embedding $\boldsymbol{s}_{m,p}^{t_1} \in \mathbb{R}^{N^{\text{out}}}$

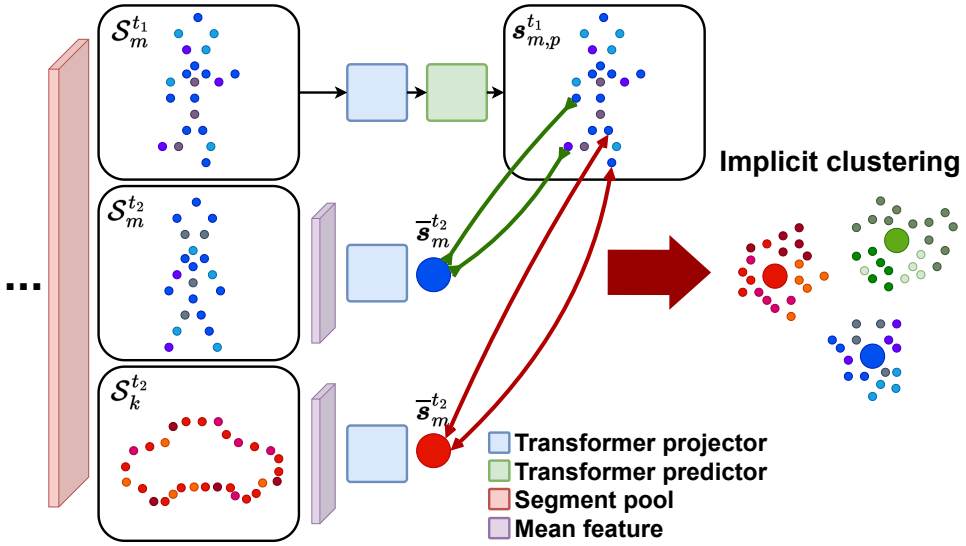


Figure 4.5: Diagram of the implicit clustering scheme. We pool the segments $\mathcal{S}^{t_1}$ and $\mathcal{S}^{t_2}$ from the backbone features. For t_2, we compute a mean representation for each segment and project it with the transformer projection head. For t_1, we project the point-wise features and then use the transformer predictor to predict the corresponding mean representation from t_2, clustering the point-wise features close to its mean target representation.

and the mean representation $\overline{\boldsymbol{s}}_k^{t_2} \in \mathbb{R}^{N^{\text{out}}}$ from a segment k as:

$$\delta_{m,p,k}^{t_1 \to t_2} = \frac{(\boldsymbol{s}_{m,p}^{t_1})^\top \overline{\boldsymbol{s}}_k^{t_2}}{\tau}. \tag{4.1}$$

Next, we use the cross-entropy loss to maximize the similarity between each point p from a segment m and the corresponding target segment mean representation as follows:

$$\mathcal{L}_m^{t_1 \to t_2} = -\sum_{p=1}^{P} \log \left(\frac{\exp(\delta_{m,p,m}^{t_1 \to t_2})}{\sum_k^M \exp(\delta_{m,p,k}^{t_1 \to t_2})} \right). \tag{4.2}$$

We compute this loss for all the M segments and sum it to get the loss for predicting segments from t_2 with the points of the segments from t_1:

$$\mathcal{L}^{t_1 \to t_2} = \sum_{m=1}^{M} \mathcal{L}_m^{t_1 \to t_2}. \tag{4.3}$$

With this formulation, our task is to predict for each point in the segment at time t_1 the corresponding segment at t_2. Since the target for all the P point embeddings $\boldsymbol{s}_{m,p}^{t_1}$ from a segment m is the same mean segment representation $\overline{\boldsymbol{s}}_m^{t_2}$, the loss will push all the points from a segment to converge to a mean representation while separating from other segments, implicitly clustering together points from the same object as shown in Fig. 4.5.

Lastly, we repeat the forward pass swapping t_1 and t_2 to have a symmetric representation, learning both the correspondences from $t_1 \rightarrow t_2$ and from $t_2 \rightarrow t_1$ leading to:

$$\mathcal{L}_{TARL} = \mathcal{L}^{t_1 \rightarrow t_2} + \mathcal{L}^{t_2 \rightarrow t_1}. \tag{4.4}$$

In the next section we give an intuition behind our choice of projection head, and how it can help the model to learn a more fine-grained representation.

4.1.5 Transformer Projection

Self-supervised representation learning methods [21, 27, 64] typically add a non-linear projection head to the backbone to project the embeddings to the target feature space. The idea is that the backbone should learn general features, and this projection head will overfit to the pre-training pretext task and later be discarded during fine-tuning. Our projection scheme follows more recent methods [23, 58], which show that an asymmetric network can improve the learned representation. Besides, with the prior that points from one segment should be similar, we want to focus on the point-wise relationships. Therefore, we replace the non-linear projection head with a self-attention transformer encoder [175].

In our siamese network scheme, the online updated network computes point-wise features. In this case, the *queries*, *keys*, and *values* for our transformer projector will be point-wise features. We aim to learn intra-class features with the attention mechanism and identify correspondences between the points from one segment. In contrast, the momentum updated network computes a mean feature vector for each segment. In this case, the attention mechanism should look for similarities and differences within the different objects segments in the scene. By using the self-attention over points and segments we aim at learning, at the same time, point-wise correspondences while also learning the differences between segments.

With the transformer projector, the pre-training can guide the learned features towards a more fine-grained representation, where intra-class point-wise features are matched with the individual segment features. Sec. 4.2.5 provides ablations comparing the transformer with the commonly used non-linear projection head.

4.1.6 Pre-Training Strategy

Sec. 4.1.2 to 4.1.5 explain the individual steps of our approach. This section summarizes the entire pre-training, putting together the modules explained in the previous subsections.

Our pipeline shown in Fig. 4.2 fetches a batch sample $\mathcal{B}_b$ of n sequential scans and applies a transformation to the scans $\mathcal{P}^t$ with the corresponding poses to have all points in a common coordinate frame. We use an unsupervised ground

segmentation approach to remove the ground $\mathcal{G}$. Next, we cluster the remaining points $\tilde{\mathcal{P}}$, defining coarse segments $\mathcal{S}$ of the objects in the scene. We sample two random scans at different times $\mathcal{P}^{t_1}$ and $\mathcal{P}^{t_2}$ from $\mathcal{B}_b$, in which we apply random augmentations $\mathcal{T}^{t_1}$ and $\mathcal{T}^{t_2}$. We compute point-wise features $\mathcal{F}^{t_1}$ and $\mathcal{F}^{t_2}$ from the sampled point clouds with the backbone, and list the segments $\mathcal{S}^{t_1}$ and $\mathcal{S}^{t_2}$ with its corresponding point-wise features. We calculate a mean embedding $\overline{\mathcal{S}}^{t_2}$ for each segment m in $\mathcal{P}^{t_2}$ and compute the target representation $\overline{\boldsymbol{s}}^{t_2}$ for each segment with a transformer projection head. For the segments $\mathcal{S}^{t_1}$, we project the point-wise features with another transformer projector followed by a transformer encoder as a predictor to get the point-wise features $\boldsymbol{s}^{t_1}$. Lastly, we compute the loss to predict the corresponding segment representation at t_2 for each point in t_1. We repeat the process changing t_1 and t_2 to symmetrically match points from scan t_2 to the corresponding segment at t_1.

4.2 Experimental Evaluation

In this section, we compare our method proposed in this chapter with the state of the art and the temporally-agnostic approach presented in Chapter 3. We show that our method achieves better performance than previous methods, and provides representations more suited for transfer learning than supervised pre-training, achieving better performance when fine-tuning to a different dataset.

4.2.1 Implementation Details and Experimental Setup

Our experimental setup follows the evaluation from Chapter 3. We first train a backbone network using the different pre-training strategies, and then fine-tune it for different downstream tasks to compare the performance of the different methods.

Datasets. We use the SemanticKITTI dataset [5, 54] for our pre-training, an autonomous driving LiDAR data benchmark with point-wise annotations. During pre-training, we use only the raw point clouds from the training sequences and the given poses to extract the temporal objects views. For fine-tuning, we use SemanticKITTI with the full scans annotations and the scribbles annotations [133]. Besides SemanticKITTI, we also use nuScenes [14, 51] for fine-tuning, to evaluate our approach in different autonomous driving datasets collected with different LiDARs and sensor setups, reporting the results on the validation sets.

Model architecture. For our evaluation, we use a MinkUNet [34] as the backbone, which voxelizes the point clouds and uses sparse convolutions to extract point-wise features. The backbone receives the point cloud coordinates and intensity as input, and the output dimension feature has dimension $N^{\text{out}} = 96$.

We use a multi-head self-attention encoder for the projection head with one layer and eight attention heads. For the predictor, we use the same multi-head self-attention encoder architecture.

Pre-training. For pre-training, we use the AdamW optimizer [102] with a learning rate of $2{\cdot}10^{-4}$ and decay of 10^{-4}, training for 200 epochs with batch size 8 using a single NVIDIA RTX A6000. We use $n = 12$ for the scans to be aggregated. We provide experiments with different numbers of scans in Sec. 4.2.5 to support this choice. The voxel resolution is set to $0.05\,m$ for the input point clouds, from which we sample a maximum of 40,000 points per point cloud. During the segment pooling, we limit it to a total of $M = 50$ segments with a maximum of $P = 300$ points per segment to avoid memory overflow. We use $\tau = 0.1$ to compute δ in Eq. (4.1) and momentum of 0.999 to update the momentum network according to the online network weights. For the baselines, we use their official repositories with their default parameters, also training them for 200 epochs and sampling 40,000 points per scan.

Fine-tuning. To evaluate our method and compare it with the baselines, we fine-tune the models to three downstream tasks: semantic segmentation, panoptic segmentation, and object detection. For semantic segmentation, we use the same training pipeline used in the previous chapter with the AdamW optimizer [102] and a learning rate of 0.001. For panoptic segmentation, we use the baseline evaluated by Hong et al. [70], where a 3D backbone network is used together with a semantic and an instance heads, followed by a clustering post-processing to identify the instances. We use the pre-trained MinkUNet [34] model as the 3D backbone. We use a learning rate of 0.2 with batch size 8, training for 50 epochs. For object detection, we use the same toolbox and hyperparameters used in DepthContrast [213] with the PartA2 detector [152]. For all the tasks, we evaluate the method with the same model trained from scratch (without pre-training) and with the model after pre-training with the different self-supervised methods.

4.2.2 Fine-Tuning for Semantic Segmentation

In this evaluation, we want to measure the semantic information learned by the network during pre-training. We fine-tune it to semantic segmentation on SemanticKITTI and nuScenes after pre-training on SemanticKITTI.

4.2.2.1 Label Efficiency

In this experiment, we use the SemanticKITTI subsets of scans used in previous chapter, with 0.1%, 1%, 10%, 50%, and 100% of the labeled scans. We also evaluate on the scribbles annotations [133], which labels just a subset of points

Table 4.1: Results given as mIoU of the models pre-trained on SemanticKITTI fine-tuned to semantic segmentation on SemanticKITTI with different percentage of labels and scribbles annotation.

	mIoU [%] ↑					
Method	Scribbles	0.1%	1%	10%	50%	100%
Scratch	54.96	29.35	42.77	53.96	58.27	59.03
PointContrast [196]	54.52	32.63	44.62	58.68	59.98	61.45
DepthContrast [213]	55.90	31.66	48.05	57.11	60.99	61.14
SegContrast	56.70	32.75	44.83	56.31	60.45	61.02
TARL (Ours)	**57.25**	**38.59**	**51.42**	**60.34**	**61.42**	**61.47**

for each scan in the SemanticKITTI training sequences. We report the mean intersection-over-union (mIoU) of the models on the validation set, while fine-tuned on the aforementioned subset of labels from the training set.

Tab. 4.1 shows the results from our approach compared to the baselines. As seen, overall the pre-training methods boost the performance compared to the network without pre-training on both scribbles annotations and the subsets of labeled scans. However, our approach surpasses previous state-of-the-art methods in all the different subsets, with a bigger gap when fewer labeled scans are used to fine-tune the model. We can also notice that, after pre-training with our method, only 10% of labeled scans were necessary for the model to surpass the performance of the network trained from scratch with 100% of the labeled scans, as shown in Fig. 4.1. This result suggests that our method can effectively reduce the amount of labeled data necessary for the semantic segmentation task by around ten times.

4.2.2.2 Linear Evaluation

In this experiment, the pre-trained backbone is frozen, training only a linear head on top of it to evaluate how descriptive the self-supervised learned representation is without fine-tuning it to the target task. We perform this evaluation on SemanticKITTI and nuScenes datasets. In Tab. 4.2, the randomly initialized network (frozen backbone) achieves low performance, suggesting that the features extracted by the backbone play the main role in achieving semantic segmentation. All the pre-training methods improve the performance compared to the network without pre-training. However, our approach has a clear performance gap compared to the baselines, especially when training to the same dataset used for pre-training. These results suggest that the representation learned by our method can embed more semantic information, even without using labels, achieving higher

Table 4.2: Linear evaluation results given as mIoU on SemanticKITTI and nuScenes datasets for semantic segmentation.

	mIoU [%] ↑	
Method	KITTI	nuScenes
Frozen backbone	4.10	5.62
PointContrast [196]	25.53	19.70
DepthContrast [213]	13.87	9.65
SegContrast	27.42	23.75
TARL (Ours)	**35.90**	**26.08**

performance than previous state-of-the-art methods on both datasets.

4.2.2.3 Generalization

In this experiment, we evaluate the generalization of the learned features. We use the network pre-trained on SemanticKITTI and fine-tune it on nuScenes. We use nuScenes full training set and the mini training subset to fine-tune the network, and evaluate on the full validation set. In this evaluation, we also compare the results of the fully supervised semantic segmentation pre-training on SemanticKITTI fine-tuned on nuScenes. As shown in Tab. 4.3, even though all methods can improve the performance of the model trained on nuScenes, our approach can surpass previous self-supervised methods by around 8% when training with fewer labels. Besides, compared with supervised pre-training of the network on SemanticKITTI, our approach achieves better performance on both full and mini training sets. These results suggest that our method can replace supervised pre-training for LiDAR data, since it achieved better performance when fine-tuning to a different dataset, collected with a different sensor setup.

4.2.3 Fine-Tuning for Panoptic Segmentation

In this evaluation, we want to assess the use of the learned features also for different downstream tasks. We fine-tune the pre-trained models for panoptic segmentation to also evaluate the instance-level features of our learned representation.

4.2.3.1 Label Efficiency

Same as for semantic segmentation, we fine-tuned the model to SemanticKITTI using the same percentage subsets, reporting the mIoU and the panoptic quality

Table 4.3: Results given as mIoU of the self-supervised and supervised pre-trained models on SemanticKITTI fine-tuned to semantic segmentation on nuScenes mini and full training sets.

	mIoU [%] ↑	
Method	Mini	Full
Scratch	26.94	66.03
Supervised pre-training	38.39	67.35
PointContrast [196]	31.92	67.31
DepthContrast [213]	27.81	64.70
SegContrast	31.27	67.70
TARL (Ours)	**39.36**	**68.26**

(PQ) on the validation sets. Tab. 4.4 shows that our method is consistently better than previous self-supervised pre-training approaches. As the number of training samples increases, the differences diminish. However, our method has a clear performance gap compared to the baselines when trained with fewer labels. Similar to the semantic segmentation experiment, after pre-training with our method, only half of labeled scans were necessary for the model to surpass the performance of the network trained from scratch with 100% of the labeled scans, as seen in Fig. 4.1. These results show that our method can derive a feature representation suitable not only for semantic segmentation, but also for other downstream tasks.

4.2.3.2 Generalization

To evaluate the generalization of our learned features on panoptic segmentation, we use the network pre-trained on SemantiKITTI to fine-tune it on nuScenes full and mini training sets, evaluating it on the full validation set. Tab. 4.5 shows that the representation learned by our method is more suited to be transferred to a different dataset. As in semantic segmentation, our method achieved better performance than previous self-supervised approaches and supervised pre-training on SemanticKITTI. These results agree with the semantic segmentation evaluation, suggesting that our method is better suited for transfer learning than the supervised pre-training.

With these experiments, we validate that our learned representations is able to extract not only semantic knowledge from the data but also instance-level information, surpassing previous methods in IoU and PQ metrics.

Table 4.4: Results reporting PQ and mIoU when fine-tuning the pre-trained models to panoptic segmentation with different percentage of labels on SemanticKITTI.

	0.1%		1%		10%		50%		100%	
Method	PQ [%] ↑	IoU [%] ↑	PQ [%] ↑	IoU [%] ↑	PQ [%] ↑	IoU [%] ↑	PQ [%] ↑	IoU [%] ↑	PQ [%] ↑	IoU [%] ↑
Scratch	4.76	11.13	22.72	30.84	47.20	53.53	55.32	61.94	55.40	59.75
PointContrast [196]	5.86	11.51	27.37	32.49	47.57	54.63	54.21	59.48	55.85	61.49
DepthContrast [213]	7.65	13.56	27.31	32.30	46.85	51.27	54.55	59.60	56.15	60.81
SegContrast	7.58	14.46	26.14	32.85	47.02	53.47	55.38	60.04	**56.73**	61.96
TARL (Ours)	**10.26**	**17.01**	**29.24**	**34.71**	**51.27**	**57.59**	**56.10**	**62.36**	56.57	**62.05**

Table 4.5: Results reporting PQ and mIoU of the self-supervised and supervised pre-trained models on SemanticKITTI fine-tuned to panoptic segmentation on nuScenes mini and full training sets.

	Mini		Full	
Method	PQ [%] ↑	IoU [%] ↑	PQ [%] ↑	IoU [%] ↑
Scratch	23.78	23.96	52.98	58.17
Supervised pre-training	24.77	23.60	53.19	58.05
PointContrast [196]	26.58	25.46	51.06	56.39
DepthContrast [213]	28.66	27.30	51.51	57.06
SegContrast	28.84	26.79	52.31	57.24
TARL (Ours)	**32.22**	**30.73**	**53.26**	**59.14**

4.2.4 Fine-Tuning for Object Detection

As a third downstream task, we compare the methods fine-tuned for object detection to evaluate how general is the representation learned by our method compared with previous state-of-the-art methods. In this evaluation, we fine-tune the model with the whole training set of the KITTI dataset [54] and report the mean average precision (mAP) for car, pedestrian, and cyclist classes on the three difficulty levels.

In Tab. 4.6, we compare the baselines' performance with our approach. For the car class, the pre-training methods achieve a marginal improvement compared to the network trained from scratch. For pedestrian and cyclist, the improvements brought by the pre-training are more substantial. On those classes, our method achieves the best performance on moderate and hard difficulties, showing that our approach can also boost the performance when fine-tuned on object detection.

Together with the evaluations of the other downstream tasks, these results suggest that our method is able to learn a general representation. Our method achieved better performance than previous approaches on all three downstream tasks, showing that our approach learns a robust representation, not biased to a specific task but with generalizable representations suited for different tasks and datasets.

4.2.5 Ablation Studies

In this section, we present further ablation studies regarding the pre-training and its different modules. In these experiments, we evaluate the network by fine-tuning the pre-trained network on semantic segmentation. We compare the

Table 4.6: Results reported as mAP when fine-tuning the pre-trained models to object detection on KITTI full training set for car, pedestrian and cyclist classes on the easy, moderate and hard difficult levels.

	Car			Pedestrian			Cyclist		
	mAP [%] ↑								
Method	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
Scratch	91.75	82.13	80.16	67.65	63.13	57.73	91.74	74.94	70.34
PointContrast [196]	91.55	81.70	79.88	67.33	62.32	56.55	92.33	75.20	70.66
DepthContrast [213]	91.98	82.04	80.03	**69.13**	63.97	57.71	**92.68**	73.67	70.10
SegContrast	92.08	82.10	**80.18**	68.72	63.77	57.36	91.26	74.70	70.28
TARL (Ours)	**92.09**	**82.15**	80.10	68.50	**64.00**	**58.74**	92.61	**75.44**	**70.97**

Table 4.7: Ablation study comparing the pre-training performance using only data augmentation and our proposed temporal views extraction together with the data augmentation showing mIoU.

	mIoU [%] ↑			
	0.1%	1%	10%	Linear eval
Data augmentation only	38.40	50.08	59.98	28.50
Temporal views + Data augmentation	**38.59**	**51.42**	**60.34**	**35.90**

Table 4.8: Ablation study comparing the pre-training performance using a non-linear projection head and a transformer encoder as a projection head showing mIoU.

	mIoU [%] ↑			
	0.1%	1%	10%	Linear eval
Non-linear projection head	38.19	50.22	56.54	32.27
Transformer projection head	**38.59**	**51.42**	**60.34**	**35.90**

results in the lowest label regimes, i.e., 0.1%, 1%, and 10%, since those are regimes with more significant performance differences between the methods. Besides, we compare the linear evaluation results to measure the amount of semantic information embedded in the different ablations.

4.2.5.1 Temporal Views

In this experiment, we compare the performance of our method pre-trained with temporal views of an object as natural augmented pairs instead of using only data augmented to generate artificial augmented pairs from the same scan. Tab. 4.7 shows the results when generating augmented pairs using only artificial data augmentation and when extracting temporal views of one object and then applying data augmentation to generate the augmented pairs. As seen, using the temporal views as natural augmentations of an object can boost the network performance when fine-tuning with different amounts of labels. Also, the linear evaluation suggests that by learning from different views of the same object, the representation can extract more semantic knowledge during pre-training.

4.2.5.2 Transformer Projection Head

In this ablation study, we compare the pre-training performance using the common non-linear projection head with the transformer projection head. In this experiment, we use a non-linear projection head composed of one linear layer, a

Table 4.9: Ablation study comparing the pre-training performance using a different number of n scans interval to extract temporal views of the objects showing mIoU.

	mIoU [%] ↑			
Number of scans	0.1%	1%	10%	Linear eval
30	36.36	44.14	56.41	32.27
18	38.19	45.76	58.39	32.50
15	38.15	50.44	57.84	33.52
12	38.59	51.42	**60.34**	35.90
9	38.30	**51.82**	57.42	35.97
6	**38.75**	51.37	59.57	**36.88**

batch normalization and a ReLU layer, and another linear layer. Tab. 4.8 shows that the network could achieve better performance with the transformer projection head when fine-tuning with different label percentages. Also, similarly to the results reported in Sec. 4.2.5.1, the linear evaluation suggests that the network pre-trained using the transformer projection head could learn a representation that embeds more semantic information, achieving better mIoU.

4.2.5.3 Different Number of Aggregated Scans

In this ablation study, we pre-train the network with different n values for the number of scans in the interval to extract the object views. We report the results in Tab. 4.9 to compare our chosen $n = 12$ with different sizes of the scans intervals. From $n = 6$ to $n = 12$, the results have just marginal differences. However, when fine-tuning with 10% of labels, we can see that the pre-training with $n = 12$ achieves the best performance, also performing considerably better than the network trained from scratch with 100% of labels. Furthermore, when using bigger n the network performance decreases. Therefore, we decided to stick with the $n = 12$.

4.3 Related Work

Image-based self-supervised representation learning methods learn strong representations by optimizing a network to distinguish between augmented versions of the same image. Such methods were extended to process video data, leveraging consecutive frames of a video. In Chapter 3, we discussed our method for learning representations from a 3D point cloud with a segment-wise discrimination. In this chapter, we extended this approach to also account for temporal data in a sequence of 3D LiDAR scans, guiding the network to learn more meaningful

representations. This section discusses video-based and further 3D representation learning methods related to the approach presented in this chapter.

Video self-supervised representation learning has been studied given the strong representations learned within image-based self-supervised methods [21, 22, 23, 58, 64, 173, 205], leveraging video data, which learn temporally consistent representations [8, 40, 77, 92, 134, 143, 172, 183]. Some works use the contrastive learning formulation, typically used with single images, to process sequential image data [8, 77, 134, 143]. In this case, positive and negative pairs are sampled within frames close or far away in time. The processing of video data optimizes the network to embed temporal information, learning scene changes within the video, and deriving stronger representations from it. Some works train the network in conjunction with more complex pretext tasks, such as discriminating whether two random video clips overlap, as in the work by Jenni et al. [77], or separating stationary and non-stationary features in the video, as in the approach by Behrmann et al. [8]. Apart from contrastive-based methods, masked autoencoders are leveraged to learn representations from temporal data [172, 183]. Such methods mask out patches of pixels in the videos, training the network to reconstruct the missing part, learning meaningful information from the data in this process. Inspired by those video representation learning methods, we extend the approach discussed in Chapter 3 to account for temporal data. Especially for LiDAR data, consecutive scans collected in a dynamic environment lead to drastic changes in the object appearance due to the sparsity and viewpoint changes within the LiDAR trajectory. Therefore, optimizing the network to match representations between different views of the same object yields more descriptive features compared to the temporal-agnostic method described in the previous chapter.

4D self-supervised representation learning followed video-based methods from the image domain to also train a network with unlabeled sequences of point clouds [29, 49, 73, 182]. Earlier works focused on training the network with sequences of 3D point clouds with recorded human actions [49, 182]. Given a sequence of 3D point clouds recording the movements of a person, Wang et al. [182] optimize the network to predict the correct point cloud sequence. Differently, Fan et al. [49] train the model to maximize the similarity of features over fixed regions within the point cloud throughout the sequence of point clouds. Huang et al. [73] focus on object-level features by generating synthetic sequences of 3D point clouds by applying several consecutive geometric transformations to the objects' point clouds, and training the network to maximize the feature similarities across all the synthetic point clouds in the sequence. Similarly, Chen et al. [28] generates synthetic sequences of point clouds, but at the scene level. Objects are synthetically placed in a large and dense scene point cloud, and an artificial object trajectory is generated, moving the object through

the scene. Then, the network is trained to compute similar point-wise features for corresponding points across the consecutive scene point clouds and the synthetic moving objects. Similarly, in this chapter, we exploit sequences of point clouds to learn a spatio-temporal representation from unlabeled data. However, our method leverages real-world LiDAR data and the dynamic objects within it, learning a temporal representation from real data.

Transformer encoders were initially proposed in the natural language processing community, achieving state-of-the-art performance in the field [175]. Such architectures were then quickly adopted by the computer vision community [43, 61, 194, 200, 204]. Transformer encoders use the attention mechanism to learn the relationship between features without requiring convolutions. With the attention mechanism, transformer encoders can learn the global context from the image, rather than just the local neighboring context, as in convolutional models [43]. With the growing interest in transformers, self-supervised representation learning methods have been used to train transformer encoders, evaluating the capacity of these models to learn meaningful representations. Multiple studies [18, 153, 191] highlighted the capabilities of transformer encoders to learn strong representations in a self-supervised manner, showing that the attention mechanism can learn class-wise correspondences even without fine-tuning with labeled data [18]. Given these properties, our approach employs a transformer encoder to extract segment-wise features from the point cloud, learning a descriptive and discriminative representation from spatio-temporal point cloud data.

Despite the performance improvement brought by 3D and more recently 4D self-supervised representation learning methods, the performance of these approaches is still limited due to learning only over non-temporal data or relying on synthetic 4D point clouds. Similar to previous works [28, 73], our approach focuses on learning representations from moving objects in the 3D point clouds. However, unlike prior methods, we extract natural augmented views of the objects in the scans collected at different points in time while the vehicle drives in the environment, without relying on synthetic data. Then, the network is trained to define a common representation for those temporal views of one object, learning its correspondences with a transformer encoder. Leveraging corresponding segments of objects seen at different points in time enables our method to learn a more robust semantic representation with temporally consistent features from real-world dynamic objects, invariant to the sensor viewpoint changes.

4.4 Conclusion

In this chapter, we exploit the vehicle motion to extract different views of objects across time and learn a temporally consistent representation. We use a trans-

former encoder as a projection head, implicitly clustering features of points from the same object together while discriminating between different objects' features. Our approach enables the training of neural networks to learn a representation invariant to the sensor viewpoint, deriving a descriptive feature space embedding object-level information without requiring annotated data. Our experiments demonstrate that our method outperforms previous state-of-the-art approaches on various downstream tasks. Besides, our approach reduces the amount of necessary labeled data to only 10% when fine-tuning for semantic segmentation on SemanticKITTI. These results show that our proposed approach learns a general LiDAR point cloud representation, embedding semantic information by matching objects representations viewed at different times. In addition, our method achieves better performance than supervised pre-training when fine-tuning to a dataset collected with different LiDAR sensors, suggesting that our approach could replace supervised pre-training in the LiDAR data domain.

As in Chapter 3, we evaluated our approach by fine-tuning the pre-trained model to different perception tasks and different datasets. Our method outperforms previous methods on all three evaluated downstream tasks, i.e., semantic segmentation, panoptic segmentation, and object detection. Additionally, our temporal representation learning method achieves the best performance when fine-tuned on a dataset collected with a different LiDAR sensor, even when compared to the supervised pre-training. The experimental evaluation highlights the advantage of the method proposed in this chapter compared to prior works. Our method demonstrates to be an effective pre-training strategy, even able to replace supervised pre-training, learning descriptive data representations without labels.

The segment-level representations learned by the methods described in this and previous chapter reduces the amount of annotated data needed to train perception models. Still, another challenge lies in identifying objects not labeled in the training set. Supervised perception tasks usually define a closed set of classes to classify objects present in the 3D point cloud, following a closed-world assumption. However, determining all possible classes that may appear in the real world is an intractable task. Therefore, in the next chapter, we investigate how we can use the segment-level representations learned by the network in a self-supervised manner to identify all objects in a 3D point cloud in a class-agnostic way. By using the pre-trained network to extract point descriptors, we propose a graph-based method to separate object points from background points, identifying all object instances in the scene independent of their classes.

Chapter 5

Unsupervised Instance Segmentation in 3D Point Clouds

Apart from the amount of labeled data, a second challenge in scaling data annotation regards the typically predefined and fixed set of labeled classes. During dataset annotation for different segmentation tasks, such as instance, semantic, and panoptic segmentation, a closed set of classes is defined for the data to be categorized into. In the case of instance segmentation, the task is to identify each object in the scene and assign a semantic class to it. Defining a fixed closed set of classes is the common practice for the data annotation and standardizes the labels in the dataset. It also simplifies the task for the perception system, which can then assign the most likely class to an instance within the closed set of classes. This closed-world formulation of the classification task, however, may not be sufficient to ensure safe interactions between the autonomous vehicle and its surroundings in outdoor, unstructured environments.

In the real world, it is impossible to predefine a closed set of classes to classify all possible objects that may appear in the scene during operation. The set of classes that can occur may be unlimited, which inevitably leads to some classes not being defined in the closed set of labels. This limitation comes from the so-called closed-world assumption commonly done during data annotation, i.e., the fact that all classes have been defined beforehand and have been seen during training. This closed-world assumption, which does not hold in the real world, leads to *unknown* objects being seen by the perception system at test time, which have not been seen during training time. Fig. 5.1 depicts one such open-world scenario example, where pedestrians and cars are *known* classes labeled in the SemanticKITTI dataset, but the baby stroller is not labeled, therefore, it is considered an *unknown* class. In such cases, the behavior of the perception system for those unknown classes may not be predictable, leading to possibly unsafe in-

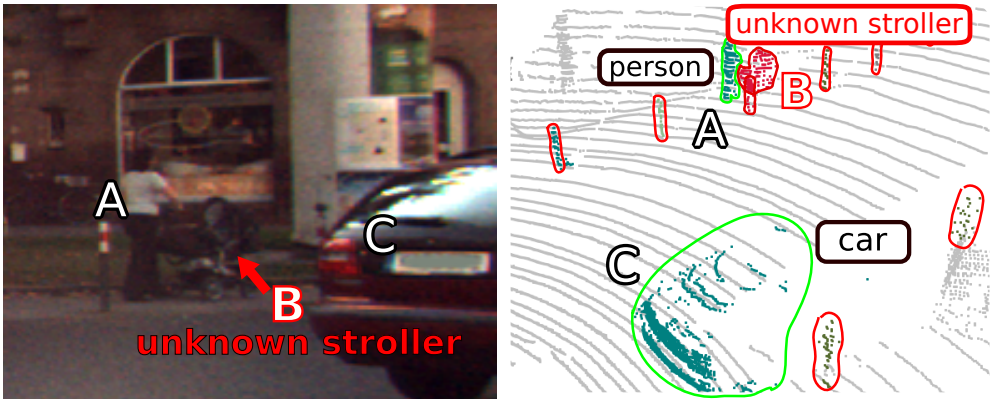


Figure 5.1: We show the extended instance annotations of the SemanticKITTI dataset [4, 54], where segments with green outline correspond to known classes, i.e., person A and car C, and red outlines correspond to things that are not part of the training data. While A and C are known classes, B corresponds to a baby stroller – an unknown class, which obviously should be detected even though not labeled in the training set.

teractions between the autonomous vehicle and its surroundings. A more general formulation is to define this task considering an open-world scenario. In this case, we divide the classification task into two categories, known and unknown objects. The known objects are those from a closed set of classes defined during data annotation. The unknown objects are those that do not fall into any of the labeled classes. This formulation enables the identification of objects in the scene even if their class is not known, providing valuable information for safe decision-making.

This open-world formulation has been studied in prior works [7, 11, 25, 44, 72, 188]. Earlier approaches employ non-learning-based methods, first performing geometric ground segmentation, followed by dividing the remaining points into individual instances by assessing the estimated points' normals [11, 124] or by clustering the non-ground points [25, 180], achieving instance segmentation in a class-agnostic manner. More recently, learning-based approaches define the identification of unknown objects as a foreground and background separation task [72, 188]. Those methods divide the objects in the scene into "stuff" and "things" classes. In the literature [84], "stuff" classes are defined as uncountable classes, e.g., road, sidewalk, etc. On the contrary, "things" classes are defined as countable objects with clearly defined boundaries. Given this distinction, a semantic segmentation network trained on known classes is used to filter out points from the known "stuff" classes. Then, a geometric clustering method is applied to the remaining points belonging to the "things" classes, e.g., car, pedestrian, cyclist, and the unknown objects in the scene, separating them into individual instances, achieving open-world instance segmentation. While non-learning methods rely purely on geometric cues without any semantic information to separate

the instances of objects in the scene, learning-based approaches strongly depend on the semantic segmentation prediction. This dependency may lead to the incorrect classification of unknown objects since they were not seen during training time. To address this issue, a self-supervised trained network can be employed to extract features from the scene, leveraging geometric cues from the point cloud and object-level information computed by the network to segment both known and unknown objects in the scene independently of their classes.

In Chapters 3 and 4, we propose two novel methods to learn meaningful representations from unlabeled data, improving performance on perception tasks by fine-tuning the learned representation to target downstream tasks, alleviating the requirement for large amounts of annotated data. In this fifth chapter, we focus on tackling the challenge of labeling all possible objects that may appear in a real-world unstructured environment. We aim to merge geometric approaches with object-level information extracted from a self-supervised pre-trained network to identify all objects in the scene in a class-agnostic manner. Given a 3D LiDAR scan, we use a network pre-trained with our method described in Chapter 3 to compute features for each point in the point cloud. Then, we identify ground points using an unsupervised ground segmentation approach [95] and cluster the non-ground points to define initial instance proposals, similar to prior geometric-based methods. With the point-wise features, we build a graph around each instance proposal, weighting the edges according to the similarities between the point-wise features. Finally, we separate the foreground instance from the background points by finding the min-cut in the graph, leveraging both geometric and object-level information from the point-wise features computed by the self-supervised pre-trained network. Besides, we propose an open-world instance segmentation benchmark to compare our method with prior works and support further research in the open-world domain. In our experiments, we compare our method with non-learning and learning based methods, showing that our method can segment instances without any labels, achieving competitive performance when compared to both supervised and unsupervised methods.

5.1 Unsupervised Class-Agnostic 3D Instance Segmentation

The main contribution of this chapter is a new unsupervised class-agnostic instance segmentation method. We approach the open-world instance segmentation task as a class-agnostic problem, defining the task as the identification of all objects in the scene, both known and unknown, independent of their classes. To achieve this, we use a network trained with our proposed self-supervised represen-

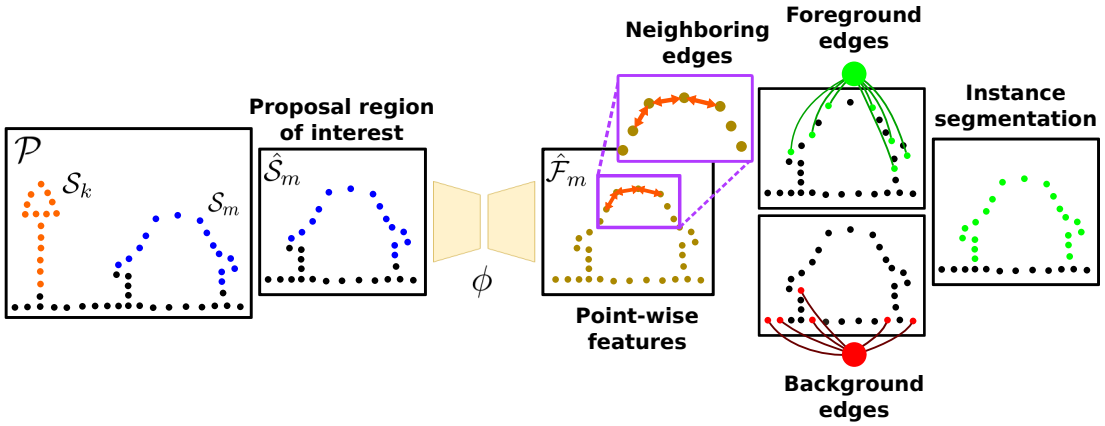


Figure 5.2: Given a point cloud $\mathcal{P}$ and set of instance proposals $\mathcal{I}$, we iterate over each proposal $\mathcal{S}_m$ defining a region of interest $\hat{\mathcal{S}}_m$ around the proposal. We extract point-wise features $\hat{\mathcal{F}}_m$ for this proposal region using a network ϕ pre-trained with the method proposed in Chapter 3. Then, we build a graph weighting the neighborhood edges with the features' similarity, and the foreground and background edges given the sampled seeds. Finally, we apply a min-cut over the graph to segment the instance from the background.

tation learning method to extract point-wise features for the point cloud and build a graph that maps the similarities between neighboring points. Then, we find the min-cut in this graph separating foreground objects from background points, achieving instance segmentation without requiring any labels. Also, to evaluate the class-agnostic instance segmentation for known and unknown classes, we propose a dataset that is an extension to the SemanticKITTI benchmark [4, 54]. We provide instance labels for unknown objects together with the instance labels already present in the original dataset. In summary, in this chapter we propose a class-agnostic instance segmentation method that (i) can segment instances without labels and without semantic background removal and (ii) achieves competitive performance compared to supervised methods.

5.1.1 Our Approach

An overview of our approach is shown in Fig. 5.2. Our method exploits GraphCut [12] and self-supervised learned features to segment instances from the point cloud. We represent the point cloud as a graph, and use the self-supervised network to compute the points' saliency map [161, 206] to sample foreground and background seeds and perform the instance segmentation. As a LiDAR point cloud may contain many instances, it can be hard to define foreground and background. Therefore, we divide ground and non-ground points and cluster the non-ground points to define the initial instances' proposal. Then, we iterate over each proposal and select a cubic region of interest around it with the same size

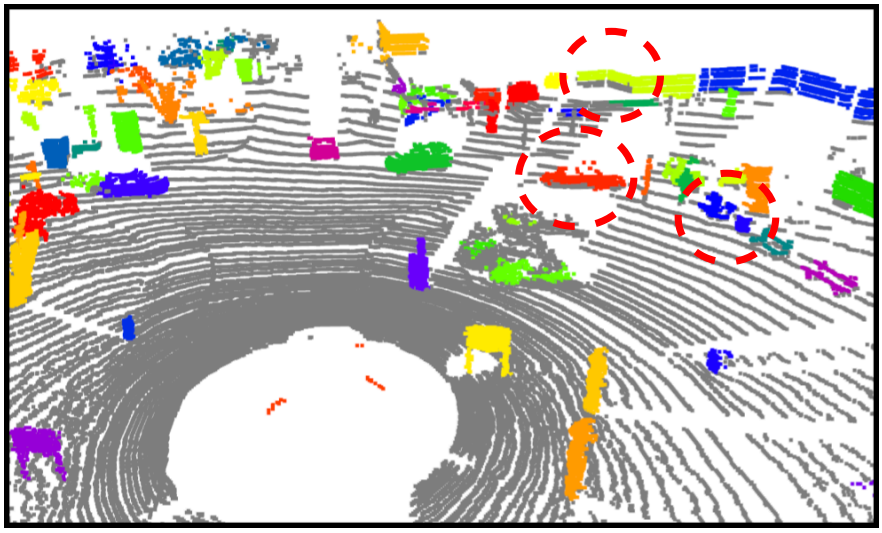


Figure 5.3: Instance proposals from ground removal and clustering with HDBSCAN [15]. Different instance proposals are shown with randomly assigned colors. Red circles highlights proposals that are either assigning different objects to the same segment, or proposals which do not cover the entire object. Best viewed in color.

as the proposal plus a pre-defined margin. For a region of interest around the instance proposal, we define the points from the current instance as foreground and all the other points (from the ground or other instances) as background. Then, we create a graph from this region of interest and divide the graph into foreground and background.

5.1.2 Instance Proposals

To segment the instances, our method relies on initial guesses, so-called proposals, which are refined to a more accurate segmentation. To define the instance proposals, we follow the same process used in previous chapters. Given a point cloud $\mathcal{P} = \{\boldsymbol{p}_1, \dots, \boldsymbol{p}_R\}$ with $\boldsymbol{p}_r \in \mathbb{R}^3$, we use a ground segmentation method [95] to partition the scene into ground $\mathcal{G}$ and non-ground points $\hat{\mathcal{P}}$. Then, we cluster the non-ground points $\hat{\mathcal{P}}$ with HDBSCAN clustering [15] to divide the scan into M segments which will be the instance proposals $\mathcal{I} = \{\mathcal{S}_1, \dots, \mathcal{S}_M\}$ where $\mathcal{S}_m \subset \hat{\mathcal{P}}$ and $\mathcal{S}_m \cap \mathcal{S}_k = \emptyset, m \neq k$.

As displayed in Fig. 5.3, these segments are a starting point but not perfect, often showing under- or over-segmentation. These segments are just the initial guess for our method. Our approach refines these proposals by mapping them in a graph of feature similarities for points inside the proposal. Then, we separate the object from the background achieving a refined instance segmentation.

5.1.3 Self-Supervised Features

Our approach takes advantage of self-supervised representation learning to extract point-wise features. We use the MinkUNet [34] as a feature extractor pre-trained with SegContrast, the approach we introduced in Chapter 3. SegContrast relies on segments extracted in an unsupervised manner by removing the ground and clustering the remaining points. Then, the contrastive loss is applied segment-wise, learning a feature space via segments discrimination.

For each instance proposal $\mathcal{S}_m \in \mathcal{I}$, we define a region of interest $\hat{\mathcal{S}}_m \subset \mathcal{P}$ and use the SegContrast pre-trained model ϕ to extract point-wise features $\hat{\mathcal{F}}_m$. We use those features to later compute the point similarity instead of using the directly measured information, such as coordinates or intensity. Since SegContrast trains the network in an unsupervised manner to produce embeddings able to discriminate segments, such a model is likely to produce relevant features for our instance segmentation task. Therefore, we have descriptive features to identify the differences between the instance segment and the background.

5.1.4 Instance Cues

With the LiDAR scan and its computed point-wise features, we aim to achieve instance segmentation by separating foreground and background points within the region of interest around an instance proposal. To achieve this separation with GraphCut, we need to identify the points most likely to belong to the foreground instance. Recently, self-supervised trained transformer networks have demonstrated a good indication of objects' boundaries drawn from their attention layers, even without labels [18]. Similar visualizations can be drawn from CNNs using saliency maps [161, 206]. Saliency maps are a visual representation of the regions in the input data that have the most influence on the computation of the network output features. Those maps are computed as the gradients over one value from the network output features with respect to the input data. Similar to transformers, we observed that the CNNs trained with SegContrast can also arrive at a similar "instance attention" in the saliency maps. These saliency values can be interpreted as which points are more likely to be foreground or background in the graph, which we exploit to sample the respective seeds for the GraphCut.

Given a model pre-trained with SegContrast ϕ, a point cloud $\mathcal{P}$, and an instance proposal $\mathcal{S}_m \subset \mathcal{P}$, we partition a region of interest $\hat{\mathcal{S}}_m \subset \mathcal{P}$ from the point cloud around the proposal $\mathcal{S}_m$ where $\mathcal{S}_m \subset \hat{\mathcal{S}}_m$. Then, we extract point-wise features $\hat{\mathcal{F}}_m = \{\boldsymbol{f}^i_m = \phi(\boldsymbol{p}^i_m) \mid \boldsymbol{p}^i_m \in \hat{\mathcal{S}}_m\}$ for all the points inside the region of interest $\hat{\mathcal{S}}_m$, and calculate the mean value $\overline{f}_m$ of all the feature vectors

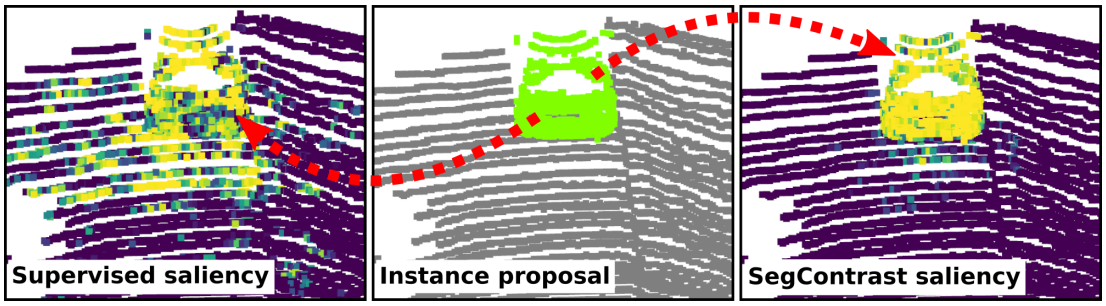


Figure 5.4: Saliency map over the proposal points features, comparing the semantic segmentation supervised network and the network self-supervised trained with SegContrast. Best viewed in color.

$\mathcal{F} = \{\boldsymbol{f}_m^p = \phi(\boldsymbol{p}_m^p) \mid \boldsymbol{p}_m^p \in \mathcal{S}_m\}$ from the P points inside the instance proposal $\mathcal{S}_m$:

$$\overline{f}_m = \frac{\sum_p^P \overline{\boldsymbol{f}}_m^p}{P}, \quad \text{where} \quad \overline{\boldsymbol{f}}_m = \frac{\sum_i^{N^{\text{out}}} \boldsymbol{f}_m^i}{N^{\text{out}}}, \tag{5.1}$$

with N^{out} as the dimension of feature vector computed by the pre-trained model ϕ.

We compute the gradients with respect to $\overline{f}_m$ calculated in Eq. (5.1) to get the saliency map around the input proposal region $\hat{\mathcal{S}}_m$. In Fig. 5.4, we compare the saliency maps generated from the network trained in an unsupervised manner with SegContrast, and trained with labels for semantic segmentation, as used for background removal by Hu et al. [72]. From this comparison, we notice that the saliency from SegContrast highlights the instance points, while the network trained supervised for semantic segmentation highlights many regions around the scan, e.g., road and sidewalk. Such analysis suggests that together with the network point-wise features, we can use the saliency values to select the seeds to perform GraphCut, since it indicates the points most related to the instance.

5.1.5 Seeds Sampling

To select the foreground and background seeds, we use the instance proposals $\mathcal{S}_m$ and their saliency values. The instance proposals correspond to different objects, which means that their shape and size are variable. Sampling a fixed number of seed points may not work well for all the proposals. Thus, we sample a number of seeds according to the proposal size to perform the GraphCut. Given the proposal $\mathcal{S}_m$ and the cube region of interest around it $\hat{\mathcal{S}}_m$, we count the number of proposal points n_f and the number of non-proposal points n_b. Then, we select $\tau_f = \frac{n_f}{\gamma_f}$ points as foreground seeds and $\tau_b = \frac{n_b}{\gamma_b}$ points as background seeds, where γ_f and γ_b are pre-defined parameters.

Fig. 5.5 displays the seed selection process. As described in Sec. 5.1.4, we compute the saliency map around the proposal region of interest to evidence the points most related to the instance. We select the τ_f points with the highest

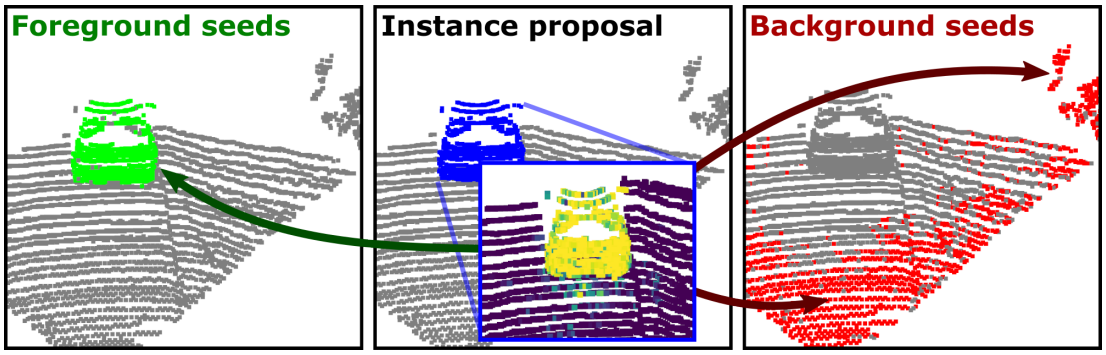


Figure 5.5: From the proposal points (in blue) we compute the saliency and sample foreground (in green) and background seeds (in red).

saliency as foreground, and the τ_b points with the lowest saliency as background. We also sample the k nearest neighbors for each foreground seed to reinforce the foreground likelihood in the seeds' neighborhood. Lastly, to avoid possible outliers from the saliency map, we remove any foreground seeds from outside the proposal $\mathcal{S}_m$ and any background seeds from inside it. By doing so, we can avoid a wrong seed being improperly assigned and harming the GraphCut performance.

5.1.6 GraphCut

To segment the instances from the point cloud, we use GraphCut [12], a classical method used previously for image segmentation, and apply it for point clouds. The method consists of a graph representation, mapping the relationship between each node and its neighbors and two terminal nodes, foreground and background. Then, a min-cut over the graph is applied, cutting the edges with the weakest relationship. We use the SegContrast point-wise features to compute the neighboring points similarity and define the non-terminal edges. And we calculate the saliency maps to sample the seeds and determine the edges between the points and the terminal nodes.

In our formulation, the graph contains a set of nodes $\mathcal{Z} = \{z_1, \ldots, z_{N+2}\}$, with $|\mathcal{Z}| = N+2$ where each node is a point in the proposal region $\hat{\mathcal{S}}_m$, corresponding to N points and 2 virtual terminal nodes, the foreground and background.

5.1.6.1 Terminal Edges

Every point has an edge with both virtual terminal nodes, and we weight these edges based on the probability of a node belonging to these nodes. Each edge probability is initialized with a small ϵ close to zero. Then, the edge probability between a selected seed and the corresponding terminal node is updated to 1.0.

Finally, the edge between a node i and a terminal node t is weighted as:

$$w_{i,t} = -\lambda \log(p_{i,t}), \tag{5.2}$$

where λ is a pre-defined parameter and $p_{i,t}$ is the defined probability for the node i to belong to the terminal node t.

5.1.6.2 Non-Terminal Edges

For non-terminal nodes, edges are defined between each point and its k nearest neighbors, according to the coordinates of the points. To weight the edges, we use our point-wise features and calculate the dissimilarity between one point feature and its neighbors. Since we rely on the min-cut of the graph to segment the object instances, the dissimilarity between the points should be as significant as possible. Thus, we use the L1 norm as a dissimilarity function between the point features $\boldsymbol{f}^i_m$ and $\boldsymbol{f}^j_m$ from a region of interest $\hat{\mathcal{S}}_m$ as:

$$d_{i,j} = \left\| \boldsymbol{f}^i_m - \boldsymbol{f}^j_m \right\|_1 . \tag{5.3}$$

Then, the edge weight between two points will be their affinity computed given the dissimilarity $d_{i,j}$,

$$w_{i,j} = \omega \exp\left(-\frac{1}{2\sigma} d_{i,j}\right), \tag{5.4}$$

where σ scale the affinity computation with respect to the dissimilarity, and ω weights the affinity value. With the graph built as described, and edges computed between each point and the terminal nodes, we then perform the min-cut on the graph, separating the instance from the background. We do this process iteratively over each proposal to achieve the instance segmentation for the whole point cloud, separating each object in the point cloud.

5.1.7 Class-Agnostic Instance Segmentation Pipeline

The pipeline for our class-agnostic instance segmentation method is presented in Fig. 5.2. Given a LiDAR scan $\mathcal{P}$, we define a set of instance proposals $\mathcal{I}$. For each proposal $\mathcal{S}_m \in \mathcal{I}$, we define a region of interest $\hat{\mathcal{S}}_m$ around it, computing its point-wise features $\hat{\mathcal{F}}_m$ with a network ϕ pre-trained with our proposed self-supervised representation learning method. Then, we build a graph from the instance proposal region of interest $\hat{\mathcal{S}}_m$, weighting the edges based on the neighboring points' features similarities. We sample foreground and background seeds from the nodes in the graph, defining the terminal connections. Finally, we partition this graph with GraphCut, separating the foreground instances from the background points.

In the next section, we provide details on our proposed open-world instance segmentation benchmark and the metrics used to assess the performance of our approach and previous state-of-the-art methods. We formally define the open-world instance segmentation problem, and present the proposed evaluation metrics as well as the description of our open-world instance segmentation dataset.

5.2 Open-World LiDAR Instance Segmentation Benchmark

This section outlines our open-world benchmark, which extends the test set and validation set of SemanticKITTI [4] with *unknown* class instance labels.

5.2.1 Problem Setting

Existing instance segmentation models assume all object classes present during inference to be manually labeled and present at training time. We refer to these object classes as *known* classes. In this chapter, we also want to focus on instance segmentation for those objects that may only appear during the inference, which we denote as *unknown* classes.

More formally, the set of all possible object classes $\mathcal{X}$ is potentially unlimited, and many instances occur rarely. In practice, we cannot record, label, and evaluate performance on all possible object classes, as these appear in the long tail of the object class distribution. It is thus practically only feasible to label a fixed subset of these classes.

To this end, we perform the following division. We use the labels provided from SemanticKITTI for the set of known classes (i.e., $\mathcal{K} = \{k_1, \ldots, k_N\} \subset \mathcal{X}$) for the whole dataset consisting of train, validation, and test set. This set contains the frequently occurring object classes, such as *car*, *person*, *truck*, and similar. We label an additional, disjoint set of object instances only in the validation and test set, i.e., unknown object instances $\mathcal{U} \subset \mathcal{X}$ with $\mathcal{U} \cap \mathcal{K} = \emptyset$. As we tackle instance segmentation, we label only "thing" classes, which can be clearly instantiated. Thus, our test set provides a proxy for evaluating performance for unknown objects that only appear rarely. We note that examples of these instances may occur in the training and validation set, but are not labeled as instances.

5.2.2 Benchmark Evaluation Metrics

For open-world instance segmentation, we evaluate how well we can decompose a LiDAR point cloud into a unique set of object instances. To quantify the per-

formance, one possibility would be to adopt the recall-based variant of panoptic quality, unknown quality [188], that replaces the recognition quality term (F1-score) with a recall-based measure. However, this metric treats the "stuff" regions of the point cloud as a single instance, which is not desirable [141]. For example, the vegetation class could be decomposed into several instances (tree trunks, small bushes) – these are not labeled by human annotators but can be considered valid instances, depending on the task, e.g., for segment-based LiDAR odometry or SLAM [45].

We instead adopt the recently introduced LiDAR Segmentation and Tracking Quality (LSTQ) metric [1]. It consists of two terms, a classification term S_{cls} and a segmentation term S_{assoc}. However, since we propose a class-agnostic task, we remove the S_{cls} term, relying only on the S_{assoc} term to evaluate how good are the instance segments.

The association term S_{assoc} measures how well we assign points to their instances independent of the semantics:

$$S_{assoc} = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \frac{1}{|t|} \sum_{\substack{s \in \mathcal{S} \\ s \cap t \neq 0}} TPA(s,t) IoU(s,t), \tag{5.5}$$

where the IoU term for each ground truth object $t \in \mathcal{T}$ and a prediction $s \in \mathcal{S}$ pair is computed based on sets of true positive associations (TPA), false negative associations (FNA), and false positive associations (FPA). These sets are evaluated in a class-agnostic manner, but differ from the work of Aygün et al. [1], as the sets are calculated per-scan instead of the whole sequence. Intuitively, the TPA set quantifies how many points were correctly assigned to their corresponding instance, and TPA and FPA sets signal two different types of point-to-instance association errors. More precisely, the TPA set contains all mutually overlapping points. The FPA set denotes all points in s that do not overlap with t, and finally, FNA contains all points from t that are not contained in s (see also Aygün et al. [1]).

The association term is class-agnostic and only informs us how well we assign points to labeled object instances. This allows us to evaluate instance segmentation independent of semantics, making this metric uniquely suitable for open-world LiDAR instance segmentation evaluation.

5.2.3 Open-World SemanticKITTI Dataset and Benchmark

A natural basis for a dataset suitable for benchmarking segmenting objects that appear in the long tail of the object class distribution would be a dataset that provides instance labels for the most common objects, such as traffic partici-

Table 5.1: Instance and bounding box counts per class for Open-World SemanticKITTI test set. Known instance class annotations are also annotated in the train set, and unknown are only available on the test set and validation set.

Object class	Number of instances	Number of bounding boxes
car	5,034	318,718
bicycle	549	50,440
person	373	25,287
other-vehicle	138	8,655
motorcycle	82	8,167
bicyclist	80	4,296
truck	50	2,596
motorcyclist	9	490
unknown	**3,587**	**292,871**

pants, and semantic labels for stuff classes. SemanticKITTI [5, 54] or nuScenes-lidarseg [14] provide such labels, opposed to recent object detection datasets which provide only 3D bounding boxes [14, 19, 167]. We opt for the SemanticKITTI dataset [5, 54] that extends the KITTI odometry benchmark [54] with dense point-wise semantic and instance labels for each LiDAR scan. It contains 23,000 labeled scans in train and 20,000 labeled scans in test set, providing semantic and object instance labels for 6,315, in total 418,649 bounding boxes, object instances belonging to several known object classes.

We built the open-world SemanticKITTI benchmark, which is suitable for assessing the performance in the open-world setting. For this thesis, we extend the hidden test set of SemanticKITTI with 3,587 instances, in total 292,871 additional object instance labels for object classes that do not necessarily belong to a semantic class from the original object classes – unknown objects. See Tab. 5.1 for statistics on the distribution of classes. We additionally also label instances of unknown classes in the validation set, but provide only a server-side evaluation to keep the instances unknown at training time.

5.3 Experimental Evaluation

The main focus of this chapter is a novel class-agnostic instance segmentation approach. Unlike previous methods, our work does not need instance or semantic labels to remove the background. We present our experiments to show the capabilities of our method to segment instances without semantic background removal, which achieves competitive performance compared to state-of-the-art supervised methods.

Table 5.2: Evaluation results with S_{assoc} metric on open-world instance segmentation benchmark test set for known, unknown and all the instances. Ours† is our method with ground removed using the proposals ground segmentation.

		$S_{assoc} \uparrow$		
Method	Supervised	known	unknown	all
Euclidean		0.651	0.426	0.611
Quick shift		0.212	0.406	0.246
HDBSCAN		0.660	0.601	0.650
Range image		0.270	0.427	0.297
4D-PLS	✓	**0.795**	0.004	0.657
Hu et al.	✓	0.697	0.587	0.678
Ours†		0.677	**0.605**	0.664
Ours		0.720	0.599	**0.699**

5.3.1 Implementation Details and Experimental Setup

In our evaluation, we compare our novel class-agnostic instance segmentation method with previous unsupervised and supervised methods. All the methods are evaluated on the hidden test set from our proposed open-world instance segmentation benchmark to assess the capabilities of the different approaches on identifying both known and unknown objects in the scene.

Hyperparameters. In all our experiments, we use the same parameters, defined empirically. For the instance proposal, we use a pre-defined margin of 1 meter around the proposal size to define the region of interest. For the seed sampling, we use $\gamma_f = 2$, and $\gamma_b = 2$. For GraphCut, we use $\sigma = 1.0$, $\omega = 10.0$ and $\lambda = 0.1$, and we select the $k = 8$ nearest neighbors to define the non-terminal edges, and build the graph. As the self-supervised pre-trained model, we use the same pre-training described in Chapter 3, trained for 200 epochs.

Baselines. We compare our approach in two setups: using only the segmentation from GraphCut, named as *Ours*, and with a post-processing step, where we filter points from the ground segmentation used to generate the proposals, named as *Ours*†. We compare our results with different unsupervised clustering methods and supervised learning-based methods. For the clustering methods, we use the same instance proposals process used for our approach, i.e., remove the ground, then cluster the remaining points. We evaluate HDBSCAN [15] and Euclidean clustering [151]. Also, we compare with a technique that operates on the birds-eye-view representation of the point cloud, clustering with quick shift algorithm [123], and a fast range image-based proposal generation method [11]. Additionally, we evaluate two supervised learning-based approaches. The method

Table 5.3: Evaluation results with S_{assoc} on open-world instance segmentation benchmark test set for each known class.

		S_{assoc} ↑							
Method	Sup.	car	bicycle	motorcycle	truck	other-vehicle	person	bicyclist	motorcyclist
HDBSCAN		0.695	0.178	0.497	0.690	0.695	0.674	0.772	0.757
4D-PLS	✓	**0.876**	0.183	0.436	0.553	0.495	0.504	0.687	0.649
Hu et al.	✓	0.755	0.200	0.539	0.512	0.509	0.503	0.693	**0.793**
Ours†		0.714	0.193	0.508	0.693	0.698	0.678	0.769	0.747
Ours		0.762	**0.203**	**0.552**	**0.722**	**0.737**	**0.684**	**0.788**	0.773

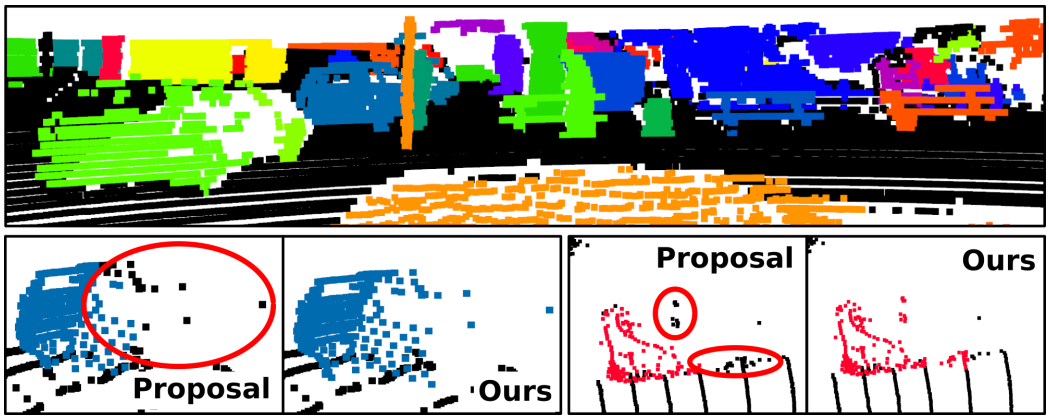


Figure 5.6: Without using any background removal our method can define the boundaries between the objects and the ground. On the lower part we compare our final result with the proposal generated with HDBSCAN. Our method can refine the proposals and correctly segment points previously ignored (highlighted in the red circles).

proposed by Hu et al. [72] remove the background with a semantic segmentation network, and cluster the remaining points given a learned *objectness* value. And a fully data-driven method 4D-PLS [1] for closed-world instance segmentation.

5.3.2 Association Quality

This experiment evaluates the association between the predicted and ground truth instances on the test set. The results show that our method achieves state-of-the-art performance, even though not using labels.

Tab. 5.2 shows the evaluation of the different methods with the S_{assoc} metric, for the known and unknown instances, and the class-agnostic case, i.e., both known and unknown. Comparing the different clustering methods, HDBSCAN

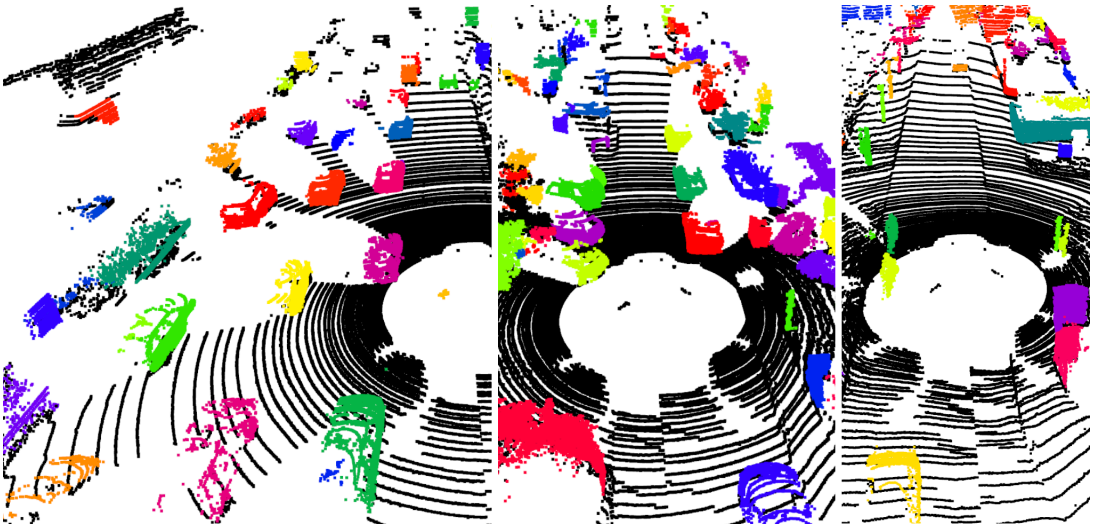


Figure 5.7: Qualitative results of our class-agnostic instance segmentation method in different scenarios.

presents the best performance, which supports our choice of using it for the instance proposal. The 4D-PLS method achieves the best performance for the known instances, while our method has the best performance for the unsupervised methods. Our approach improves by a large margin the HDBSCAN instance proposal and even surpasses the method proposed by Hu et al. [72]. For the unknown instances, the performance of supervised methods degrades. Our method achieves the best performance for unknown when removing the ground points. When looking at the class-agnostic case, our method with ground removal is better than unsupervised and supervised methods.

Tab. 5.3 shows the S_{assoc} for each known class. The performance of the supervised methods presents the best performance on the most represented class, i.e., car. For other classes, their performance degrades since those classes have fewer samples in the training set. Our method surpasses HDBSCAN instance proposals and the supervised methods on most classes. By using unsupervised learned features, our approach is less overfitted to frequently occurring classes, being more suited to class-agnostic instance segmentation. Fig. 5.6 shows our results on one scan and compares our segmentation with the HDBSCAN instance proposals. Fig. 5.7 depicts further results of our method in different scenarios.

5.3.3 Instance Segmentation Quality

This experiment evaluates the instance segmentation quality using intersection-over-union (IoU) and recall between predicted instances and the ground truth. We compute the IoU and recall filtered by different IoU thresholds and calculate the average of the different thresholds to evaluate the overall performance of the

Table 5.4: Evaluation results with IoU and Recall with different IoU thresholds and the performance averaged over all thresholds {0.5, 0.6, 0.7, 0.8, 0.9} on test set.

		IoU_{90} [%] ↑		$Recall_{90}$ [%] ↑		IoU_{70} [%] ↑		$Recall_{70}$ [%] ↑		IoU_{50} [%] ↑		$Recall_{50}$ [%] ↑			
Method	Sup.	known	unknown	known	unknown	known	unknown	known	unknown	known	unknown	known	unknown	IoU [%] ↑	Recall [%] ↑
Euclidean		0.484	0.263	0.503	0.276	0.637	0.393	0.688	0.434	0.681	0.429	0.761	0.493	0.569	0.618
Quick shift		0.064	0.221	0.067	0.232	0.119	0.347	0.136	0.385	0.184	0.396	0.247	0.468	0.158	0.185
HDBSCAN		0.424	0.388	0.444	0.402	0.645	0.565	0.713	0.617	0.710	0.605	0.819	0.683	0.594	0.657
Range image		0.100	0.213	0.106	0.225	0.223	0.408	0.257	0.462	0.278	0.455	0.349	0.541	0.236	0.274
4D-PLS	✓	**0.667**	0.003	**0.683**	0.003	**0.753**	0.003	**0.788**	0.004	**0.792**	0.004	**0.854**	0.004	0.613	0.644
Hu et al.	✓	0.510	**0.410**	0.528	**0.424**	0.670	0.562	0.724	0.608	0.721	0.596	0.806	0.664	**0.624**	0.676
Ours†		0.459	0.402	0.480	0.415	0.663	**0.569**	0.728	**0.619**	0.721	**0.608**	0.823	**0.683**	0.612	0.672
Ours		0.461	0.381	0.481	0.393	0.686	0.546	0.755	0.595	0.745	0.591	0.850	0.668	**0.624**	**0.686**

Table 5.5: Comparison between the instance proposals of our method and our results without and with ground removal post-processing step. Evaluation on open-world instance segmentation benchmark test set.

	S_{assoc} ↑		IoU [%] ↑	
Method	known	unknown	known	unknown
HDBSCAN	0.660	0.601	0.607	0.531
Ours†	0.677	**0.605**	0.628	**0.538**
Ours	**0.720**	0.599	**0.646**	0.517

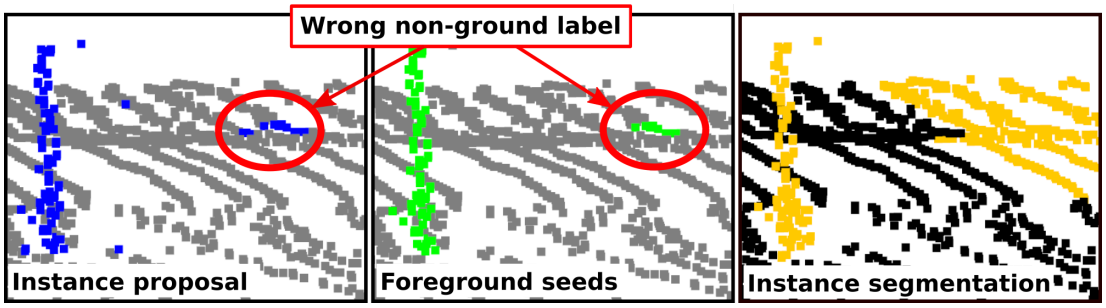


Figure 5.8: Sometimes ground is wrongly assigned to an instance proposal (blue), we sample those ground points as foreground (green), affecting our final instance segmentation (orange).

methods. Tab. 5.4 shows the results for IoU and recall for different thresholds. With a high threshold, i.e., IoU_{90}, the supervised methods perform better than the others. Since it is a panoptic segmentation method, 4D-PLS achieves the best performance for the known instances over all the thresholds. However, it fails in segmenting unknown instances. With a lower threshold, our method performance gets closer to the supervised methods and surpasses them in the unknown instances. On the overall evaluation, our method achieves an IoU on par with supervised methods and best performance in terms of recall.

5.3.4 Ablation Studies

We additionally compare our method with and without the ground removal post-processing to discuss the limitations. As seen in the previous section, without ground removal post-processing our method achieves the best overall performance, especially for known instances. However, it performs better on unknown instances with the ground removal.

Tab. 5.5 compares both setups with the HDBSCAN proposals. Even though removing the ground points leads to better instance segmentation for unknown instances, it has a higher impact for known instances, decreasing the performance

by a large margin. In Fig. 5.8, we illustrate one example to explain such behavior. Since the ground segmentation is done by an unsupervised method, it may have points wrongly assigned as ground and non-ground. Therefore, our method can improve the segmentation by correctly assigning instance points previously erroneously considered ground. However, some proposals may have ground considered as instance points due to the imperfect ground segmentation. In this case, our method may sample ground points as foreground seeds, leading to segmenting a wider ground region as part of the instance. By removing the ground with the post-processing, we filter the ground region sampled as foreground, but also filter the points correctly assigned as instances points that were previously wrongly labeled as ground. Therefore, the main limitation of our method relies on the initial proposals used for sampling the foreground and background seeds.

5.4 Related Work

Open-world perception has gained considerable relevance recently. Detecting objects independently of their classes enables more robust perception systems and alleviates the need for data annotation. Prior works on 3D point cloud data have leveraged the task as a class-agnostic segmentation employing clustering methods. More recent learning based methods leveraged this task by first identifying the known classes in the scan and then clustering the unknown classes. In this chapter, we propose a method that uses a self-supervised trained network to extract point-wise features, followed by a graph-based clustering over both 3D point coordinates and the point-wise features computed by the network. In this section, we discuss prior clustering and learning-based open-world perception approaches related to the method presented in this chapter.

Closed-world perception assumes that the complete information about the classes in the task is given at training time. It means that all objects that appear at inference time were annotated and seen during training. In this setting, different tasks can be defined to extract semantic knowledge in the context of autonomous driving, such as semantic segmentation [168, 218] or panoptic segmentation [115, 122, 217]. Given the defined task and the defined closed set of classes, data can be annotated for the target task, training a neural network with this labeled data. Large labeled datasets [4, 14, 47, 51, 54, 167] helped to push the state-of-the-art for learning-based approaches forward, achieving solid results. Such learning-based methods perform well for frequently occurring objects with many training examples, such as cars and pedestrians. Often, however, they struggle with underrepresented classes for which few training examples exist, e.g., motorcyclists or bicyclists, and with the generalization to unknown objects. The comparably poor results in such underrepresented classes highlighted the

problem of data annotation scalability, especially for classes not present in the training set, i.e., unknown to the network.

Open-world perception focuses on identifying both known classes, i.e., classes seen at training time, and unknown classes, i.e., not seen at training time. In contrast to learning-based supervised methods operating on LiDAR data, bottom-up clustering methods rely on decomposing point clouds based on proximity cues. This is particularly relevant in automotive scenarios, where intelligent vehicles must react to all objects, including those that cannot be recognized semantically. Therefore, such methods can decompose the point cloud and define instance segments in a class-agnostic way. Several approaches exploit different representations, such as 2D grids [7, 169], voxel grids [44], or range images [11, 124], to determine neighboring points and efficiently cluster points using these implicit neighborhoods. Besides that, several methods operate directly on the point cloud. A simple yet effective approach is to use Euclidean clustering [151]. Chen et al. [25] apply ground removal for moving instance clustering. Alternatively, Wang et al. [180] propose first computing a minimum spanning tree of the point cloud and learning to remove links in the graph such that the recall on the labeled classes is maximized. Still, these methods generally only consider the spatial neighborhoods of points, without any semantic cues, resulting in under- or over-segmentation.

In contrast, Hu et al. [72] propose an algorithm that finds a segmentation from a hierarchy of multiple segmentations, given a learned objectness regression function that provides an open-world scan interpretation. Such an algorithm can be used in conjunction with the aforementioned clustering methods to achieve class-agnostic instance segmentation. Differently, Wong et al. [188] propose an open-world LiDAR panoptic segmentation method, which learns to assign points to "thing" and "stuff" classes and clusters the remaining unknown points using HDBSCAN [15] to obtain segmentation hypotheses for unknown objects. This method is evaluated on the TOR4D [201] dataset, and its subset contains only rare objects, Rare4D. Unfortunately, neither the dataset nor the proposed model is publicly available. Even though learning some level of semantic information, those methods still rely on closed-world labels, optimizing the network to learn class-specific features. Similarly, our approach discussed in this chapter also uses both spatial and object-level information. Still, it mainly relies on self-supervised learned features to compute the points' similarities, without optimizing the network for a specific set of classes. By leveraging a graph representation with its edges weighted based on the point-wise features computed by the self-supervised trained network, we achieve better performance compared to prior work in class-agnostic instance segmentation. Additionally, we provide a benchmark for open-world LiDAR instance segmentation, supporting further research in this field.

This chapter proposes a new method based on self-supervised learned features and a graph optimization process to achieve class-agnostic instance segmentation. Unlike previous works, our approach does not rely on labeled data for training or background removal based on semantic segmentation predictions. Therefore, it is more suitable for class-agnostic instance segmentation, as it is not biased by the annotated classes and relies only on features learned in an unsupervised manner, enabling the identification of all the objects in the scene, regardless of their presence in the labeled classes.

5.5 Conclusion

In this chapter, we presented a novel approach for unsupervised class-agnostic instance segmentation. Our method employs clustering to define instance proposals and a graph optimization algorithm to refine these proposals into more accurate instance segmentation. We represent the point cloud as a graph and exploits self-supervised representation learning to extract point-wise features, mapping the similarities between neighboring points within the graph. It allows us to separate the foreground instance from the background points without labels. The experiments suggest that our method is more suitable for class-agnostic instance segmentation, achieving competitive performance compared to state-of-the-art supervised methods, and even surpassing them for unknown classes.

We evaluated our approach using our proposed benchmark to assess the predicted instances in comparison to previous methods. Following panoptic segmentation metrics, we evaluate the methods with the association quality and intersection-over-union metrics. In our experiments, we demonstrate that our method improves the instance proposals by incorporating object-level information from the self-supervised trained network, yielding better performance than the clustering method used to define the instance proposals. Additionally, our method demonstrated performance comparable to that of supervised methods, for both unknown and known classes, despite not requiring labels. Alongside our method, we also propose a benchmark for class-agnostic instance segmentation to support further research in the field.

Until now, we have primarily discussed discriminative methods for either training a network to learn representations from unlabeled data or leveraging those learned representations to achieve more concrete perception tasks, such as instance segmentation. Still, another limitation of 3D LiDARs compared to other sensors, such as cameras, is the sparsity of the data and the variability of resolutions between different LiDAR sensors. Different LiDARs may have different numbers of laser beams, influencing the density of the collected point cloud. Therefore, in addition to the scarcity of annotated data, data annotation is also

sensor-dependent, making it challenging to use annotated data collected with different LiDARs. In the next chapter, we will discuss how to tackle this using generative methods and transition from sparse, sensor-dependent point clouds to dense 3D scene representations. We then present our approach for generating a dense, complete point cloud from a single sparse LiDAR scan, enabling translation between data from different LiDAR resolutions by sampling point clouds from specific sensor configurations from the dense scene representation.

Chapter 6

Generative 3D Scene Completion

On top of the amount of annotated data available and the number of classes labeled, a further challenge regarding the scalability of labeled dataset from 3D LiDAR sensors stems from the differences in the point clouds collected by different LiDAR sensors. Different LiDAR sensors will measure and represent the same 3D environment as different point clouds, even within the same manufacturer. Those changes can be due to the laser beam patterns or sensor resolution, varying the number of beams used to measure the environment. Such changes drastically impact the geometric representation generated by the sensor, meaning that a network trained with data collected from one LiDAR sensor may not achieve reasonable performance when evaluated on data from a different sensor. Therefore, together with the scarcity of 3D annotated LiDAR data, the available datasets are also sensor-specific, adding another challenge that hinders the development of 3D LiDAR perception systems related to the domain gap between different LiDAR sensors.

To overcome this domain gap, several studies have investigated methods for transferring annotated data between different LiDAR sensors [2, 81, 88, 159, 203]. Classical approaches use pose information from a sequence of scans from a source LiDAR dataset, aggregating the point clouds and labels [2, 88]. Then, a point cloud resembling the data collected by the target LiDAR sensor is sampled from the densely aggregated point cloud with labels by casting rays that simulate the target LiDAR sensor. Finally, the network is trained with those sampled point clouds. Differently, Yi et al. [203] proposes training a network to achieve both tasks, predicting a dense, complete scene from a single LiDAR scan, followed by a second network to predict semantics from this completed scene. In this case, the semantic segmentation network is domain-independent, since the scene completion network guarantees that the input point cloud will be dense, independent of the LiDAR sensor used. Both domain adaptation strategies rely on a complete representation of the scene, achieving that either by aggregating scans or through

scene completion, bridging the gap between different LiDAR sensors by sampling sensor-specific point clouds.

The scene completion task has been studied for many applications, such as perception [197], localization [176], and navigation [140]. In general, scene completion aims at computing a complete representation from a partial observation of the scene. In this case, the partial observation comes from a 3D LiDAR point cloud, which is sparse and with many occlusions due to the objects in the scene. Achieving a dense and sensor-independent point cloud is a valuable representation for dealing with the domain gap inherent in LiDAR sensors. As discussed before, such representation enables either predicting semantic information directly from the dense point cloud [203] or by allowing the sampling of arbitrary LiDAR sensor point clouds [2, 88].

In this chapter, we propose a method to generate a dense and complete 3D scene from a single sparse LiDAR point cloud. We focus solely on the scene completion task for now, disregarding the semantic information, which will be addressed in the next chapter. Prior scene completion methods focused on predicting depth maps from RGB images, training a network with paired LiDAR scans and RGB images [53, 110, 111, 199], or approximating the dense scene from a LiDAR scan using an signed distance field (SDF) representation. We reframe the scene completion task as a generative task. Given recent advances in denoising diffusion probabilistic models (DDPMs), instead of predicting the missing parts of the scene, we propose using a DDPM to generate the missing parts from a LiDAR scan. We reformulate the denoising process from DDPMs as a local point denoising, enabling the scalability of point diffusion models for scene-scale point cloud data. Additionally, we propose a regularization term to stabilize the point diffusion model training, achieving more detailed scene reconstruction. We compare our method with prior scene completion approaches and demonstrate that our method achieves better scene reconstruction with more fine-grained details. Our results show the advantages of using generative models to generate the missing parts from the sparse LiDAR scans, enabling the generation of a dense, sensor-independent 3D point cloud representation.

6.1 Scaling Diffusion Models to 3D LiDAR Scene Completion

The main contribution of this chapter is a diffusion scheme for 3D data operating at the point level and at the scene scale, which we use to achieve scene completion. We reformulate the (de)noising scheme used in DDPMs by adding noise locally to each point without scaling the input data to the noise range, allowing the model

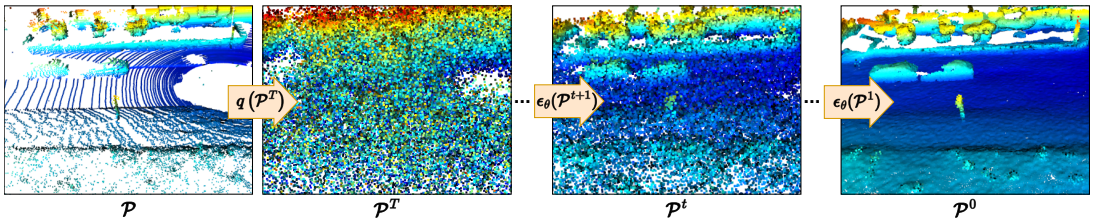


Figure 6.1: Starting from a single input scan $\mathcal{P}$, we add Gaussian noise to each point, defining the noisy input $\mathcal{P}^T$. Then, we use our trained noise predictor ϵ_θ to denoise $\mathcal{P}^T$ iteratively until arriving at $\mathcal{P}^0$, yielding a completed representation of the 3D scene.

to learn detailed structural information of the scene. Furthermore, we propose a regularization to stabilize the DDPMs training, approximating the predicted noise distribution closer to the real data. In summary, the contributions in this chapter are (i) a novel scene-scale diffusion scheme for 3D sensor data that operates at the point level, (ii) a regularization that approximates the predicted noise to the expected noise distribution, a method (iii) able to generate more fine-grained details compared to previous methods, (iv) achieving competitive performance in scene completion compared to previous diffusion and non-diffusion methods.

6.1.1 Our Approach

We propose using denoising diffusion probabilistic models to achieve scene completion from a single 3D LiDAR scan as input. First, we reformulate the diffusion process of DDPMs [106, 107, 214] to work at the scene scale. Instead of normalizing the input point cloud, we add and predict the noise locally for each point. During the denoising process, we condition the noise prediction with the input scan such that the final scene retains the structural information from the input scan while inferring the missing parts. In this formulation, the initial point cloud is a noisy version of the input scan and the networks task is to denoise it to get the complete scene as depicted in Fig. 6.1. Next, we provide the needed background on diffusion models and describe the individual components of our approach.

6.1.2 Denoising Diffusion Probabilistic Models

Denoising diffusion probabilistic models [39, 68, 127] formulate the data generation as an iterative denoising process. Commonly, the model starts from Gaussian noise [39, 68, 127] and iteratively removes noise from the input until it converges to the target output, e.g., images [39, 68, 127, 138, 144, 148, 215, 216] or shapes [106, 107, 156, 157, 198, 208, 214]. This is achieved by defining a forward diffusion process where noise is iteratively added T times to the target data. Then, the model is trained to predict the noise added at each step t. By

predicting the noise at each step t and removing it, the denoised sample should be closer to the target training data.

The diffusion process as formulated by Ho et al. [68] can be generally written as follows. Given a sample $\boldsymbol{x}^0 \sim q(\boldsymbol{x})$ from a target data distribution, the diffusion process adds noise to $\boldsymbol{x}^0$ over T steps, resulting in $\boldsymbol{x}^1, \dots, \boldsymbol{x}^T$, where $q\big(\boldsymbol{x}^T\big) \approx \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, where $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is a normal distribution with mean $\mathbf{0}$ and the identity matrix $\boldsymbol{I}$ as diagonal covariance. This diffusion process is parameterized by a sequence of defined noise factors $\beta_1, \dots, \beta_T$, where iteratively at each step t, Gaussian noise is sampled and added to $\boldsymbol{x}^{t-1}$ given β_t. This can be simplified to sample $\boldsymbol{x}^t$ from $\boldsymbol{x}^0$, without computing the intermediary steps $\boldsymbol{x}^1, \dots, \boldsymbol{x}^{t-1}$. To do so, Ho et al. [68] define $\alpha_t = 1 - \beta_t$ and $\overline{\alpha}_t = \prod_{i=1}^{t} \alpha_i$, and $\boldsymbol{x}^t$ can be sampled as:

$$\boldsymbol{x}^t = \sqrt{\overline{\alpha}_t}\boldsymbol{x}^0 + \sqrt{1 - \overline{\alpha}_t}\boldsymbol{\epsilon}, \tag{6.1}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Note that when T is large enough $q\big(\boldsymbol{x}^T\big) \approx \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, since $\overline{\alpha}_T$ gets closer to zero.

The denoising process aims to undo the T noising steps by predicting the noise $\boldsymbol{\epsilon}$ added at each step t [68]. Given an initial $\boldsymbol{x}^T$, we want to reverse the diffusion process and get to $\boldsymbol{x}^0$. The reverse diffusion step can be written as:

$$\boldsymbol{x}^{t-1} = \boldsymbol{x}^t - \frac{1 - \alpha_t}{\sqrt{1 - \overline{\alpha}_t}}\boldsymbol{\epsilon}_\theta\big(\boldsymbol{x}^t, t\big) + \frac{1 - \overline{\alpha}_{t-1}}{1 - \overline{\alpha}_t}\beta_t\mathcal{N}(\mathbf{0}, \boldsymbol{I})\,, \tag{6.2}$$

where $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}^t, t)$ is the noise predicted from $\boldsymbol{x}^t$ at step t.

This generation can also be guided given a condition $\boldsymbol{c}$. This conditional generation can either stem from a pre-trained encoder [39] or from classifier-free guidance [69], where the encoder is trained together with the noise predictor. In our case, we use the classifier-free guidance since it does not require a pre-trained encoder. With the classifier-free guidance, the model is trained to learn the conditional and unconditional noise distribution. In this case, at each training step the model has a probability ρ of predicting the unconditional noise distribution, where the conditioning is set to a null token, i.e., $\boldsymbol{c} = \emptyset$.

The training process optimizes the denoising model to predict the noise $\boldsymbol{\epsilon}$ added at step t to a given input. Given an input $\boldsymbol{x}^0$ and a condition $\boldsymbol{c}$, a random step $t \in [0, T]$ is sampled, and $\boldsymbol{x}^t$ is sampled from Eq. (6.1) with a Gaussian noise $\boldsymbol{\epsilon}$. Then, from $\boldsymbol{x}^t$, $\boldsymbol{c}$ and t, the model computes the noise prediction, supervising it with an $\mathcal{L}_2$ loss:

$$\mathcal{L}\big(\boldsymbol{x}^t, \tilde{\boldsymbol{c}}, t\big) = \big\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta\big(\boldsymbol{x}^t, \tilde{\boldsymbol{c}}, t\big)\big\|^2, \tag{6.3}$$

with $\tilde{\boldsymbol{c}}$ having a probability ρ of being the null token $\emptyset$ or the condition $\boldsymbol{c}$ otherwise, i.e., switching between unconditioned and conditioned generation.

The inference starts from an initial $\boldsymbol{x}^T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and iteratively denoise it to get $\boldsymbol{x}^0$. For the classifier-free guidance [69], we predict the conditional and unconditional noise distribution and compute the final predicted noise as:

$$\boldsymbol{\epsilon}'_\theta(\boldsymbol{x}^t, \boldsymbol{c}, t) = \boldsymbol{\epsilon}_\theta(\boldsymbol{x}^t, \emptyset, t) + s\big[\boldsymbol{\epsilon}_\theta(\boldsymbol{x}^t, \boldsymbol{c}, t) - \boldsymbol{\epsilon}_\theta(\boldsymbol{x}^t, \emptyset, t)\big], \tag{6.4}$$

where $s \in \mathbb{R}$ is a parameter that weights the conditioning to $\boldsymbol{c}$, and $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}^t, \emptyset, t)$ is the unconditional noise prediction.

With Eq. (6.4) we can compute the noise at any step t, from which we can use Eq. (6.2) to compute $\boldsymbol{x}^{T-1}, \dots, \boldsymbol{x}^0$, where $\boldsymbol{x}^0$ is a newly generated sample conditioned on $\boldsymbol{c}$.

6.1.3 Diffusion Scene Completion

In this work, we use the generative aspect of DDPMs to complete a scene measured in a single scan by a LiDAR sensor. Similarly to shape completion [106, 107, 214], the input is a partial point cloud $\mathcal{P} = \{\boldsymbol{p}_1, \dots, \boldsymbol{p}_H\}$ where $\boldsymbol{p}_h \in \mathbb{R}^3$, and the output should be the complete point cloud $\mathcal{P}' = \{\boldsymbol{p}'_1, \dots, \boldsymbol{p}'_R\}$ where $\boldsymbol{p}'_r \in \mathbb{R}^3$. In our case, the partial point cloud is a single LiDAR scan from which we want to achieve scene completion. Given a sequence of consecutive LiDAR scans and their poses, we can build a map and sample the complete scene ground truth $\mathcal{G}$ for an individual scan $\mathcal{P}$, where our scene completion $\mathcal{P}'$ should be as close as possible to $\mathcal{G}$.

Given the pair of input scan $\mathcal{P}$ and ground truth $\mathcal{G}$, we can train the DDPM to achieve scene completion. As detailed in Sec. 6.1.2, we can compute a noisy point cloud $\mathcal{G}^t$ at step t from the complete scene $\mathcal{G}$ in a point-wise fashion:

$$\boldsymbol{p}_r^t = \sqrt{\overline{\alpha}_t}\boldsymbol{p}_r + \sqrt{1 - \overline{\alpha}_t}\epsilon, \ \forall \boldsymbol{p}_r \in \mathcal{G}, \tag{6.5}$$

with $\mathcal{G}^t = \{\boldsymbol{p}_1^t, \dots, \boldsymbol{p}_R^t\}$.

In our case, we want to retrieve the complete scene $\mathcal{G}$ from $\mathcal{G}^T$. However, $\mathcal{G}^T$ retains little information from $\mathcal{G}$ due to the T diffusion steps. Therefore, we condition the generation with the scan $\mathcal{P}$ such that its structure guides the point cloud generation. From Eq. (6.4), the point-wise classifier-free noise prediction at step t can be rewritten as:

$$\boldsymbol{\epsilon}'_\theta(\mathcal{G}^t, \mathcal{P}, t) = \boldsymbol{\epsilon}_\theta(\mathcal{G}^t, \emptyset, t) + s\big(\boldsymbol{\epsilon}_\theta(\mathcal{G}^t, \mathcal{P}, t) - \boldsymbol{\epsilon}_\theta(\mathcal{G}^t, \emptyset, t)\big). \tag{6.6}$$

For training, at each iteration we select a random step $t \in [0, T]$ and compute $\mathcal{G}^t$ from $\mathcal{G}$ given Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Then, we use the model to predict the noise from $\mathcal{G}^t$ conditioned to the LiDAR scan $\mathcal{P}$ or a null token $\emptyset$ given a probability ρ as in Eq. (6.3), supervising with the loss:

$$\mathcal{L}_{\text{diff}}(\mathcal{G}^t, \tilde{\boldsymbol{c}}, t) = \big\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathcal{G}^t, \tilde{\boldsymbol{c}}, t) \big\|^2, \tag{6.7}$$

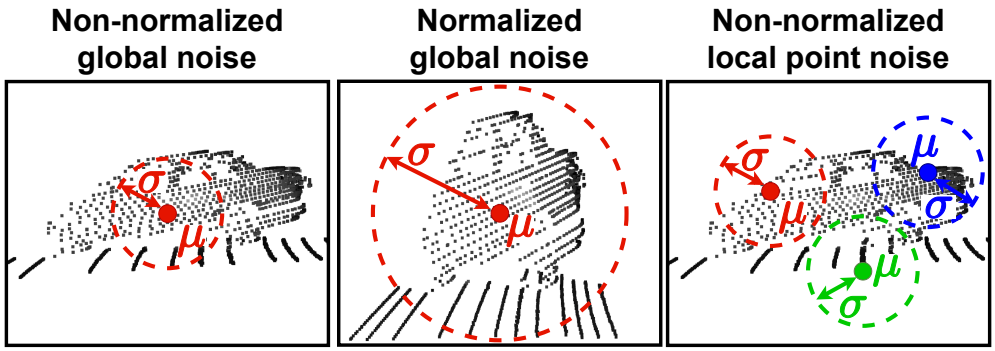


Figure 6.2: Comparison between Gaussian noise with standard deviation σ and mean μ over non-normalized and normalized input point cloud and our proposed local point-wise noise formulation.

where as in Eq. (6.3), with $\tilde{\boldsymbol{c}}$ having a probability ρ of being the null token $\emptyset$ or the LiDAR point cloud $\mathcal{P}$ otherwise. Our experimental setup follows the evaluation of prior scene-scale generative methods. We generate samples from the different methods and assess the quality of the generated scenes by comparing it with real data. Additionally, we compare the use of scenes generated by ours and prior state-of-the-art methods as annotated data to train a semantic segmentation network. During inference, as detailed in Sec. 6.1.2, we can generate a scene conditioned to a LiDAR scan $\mathcal{P}$, by denoising from $\mathcal{G}^T$ to $\mathcal{G}^0$ which is the predicted complete scene $\mathcal{P}'$.

6.1.4 Local Point Denoising

The formulation detailed in Sec. 6.1.3 is usually used for shape completion [107, 214]. Even though achieving promising results for shape completion, this formulation may not directly work at the scene scale. For single object shapes, the data is either normalized or is within a small range close to a Gaussian distribution with mean $\mu = \mathbf{0}$ and standard deviation $\Sigma = \boldsymbol{I}$. For scene scale, the LiDAR data has a much larger scale, and the data range differs depending on the point cloud axis. Therefore, the input data distribution is far from a Gaussian distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and if we normalize the data, we lose many details in the scene due to compressing it into a much smaller range as illustrated in Fig. 6.2.

To overcome this problem, we reformulate the diffusion process as a point-wise local problem. Instead of sampling $\boldsymbol{x}^t$ as a mixed distribution between $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and $\boldsymbol{x}^0$ as in Eq. (6.1), we formulate the diffusion process as a noise offset added locally to each point $\boldsymbol{p}_r \in \mathcal{G}$. In this case, from Eq. (6.1), we set $\boldsymbol{x}^0 = \mathbf{0}$ and add $\boldsymbol{x}^t$ to $\boldsymbol{p}_r$:

$$\boldsymbol{p}_r^t = \boldsymbol{p}_r + \left(\sqrt{\overline{\alpha}_t}\mathbf{0} + \sqrt{1 - \overline{\alpha}_t}\boldsymbol{\epsilon}\right) = \boldsymbol{p}_r + \sqrt{1 - \overline{\alpha}_t}\boldsymbol{\epsilon}. \tag{6.8}$$

With this formulation, the noise $\boldsymbol{\epsilon}$ is a random offset scaled with respect to the

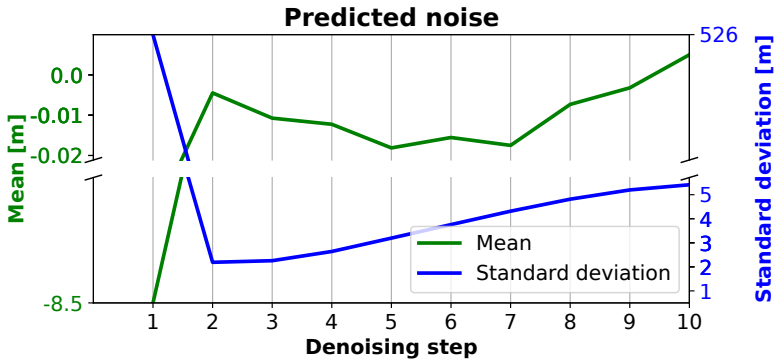


Figure 6.3: Mean and standard deviation of the predicted noise $\boldsymbol{\epsilon}_\theta$ without the noise regularization. In this experiment we use DPMSolver [103] to reduce the denoising steps from 1,000 to 10.

step t added to each point $\boldsymbol{p}_r$ in $\mathcal{G}$. The model needs to predict the noise at each step t, slowly moving the noisy points towards the target scene $\mathcal{G}$ conditioned to the LiDAR scan $\mathcal{P}$, still operating in the original scale.

During inference, due to this local diffusion formulation, $\mathcal{G}^T$ cannot be approximated by a Gaussian distribution. Instead, we can generate $\mathcal{G}^T$ from the LiDAR point cloud $\mathcal{P}$. Besides, to achieve a denser scene from a LiDAR scan, we need more points than the input scan. Therefore, given a single LiDAR scan $\mathcal{P}$ with H points, we increase its size to arrive at the same number of points as the target dense scene $\mathcal{G}$ with R points. We achieve this by concatenating the points in $\mathcal{P}$ by Q times to obtain $\mathcal{P}^* = \{\boldsymbol{p}_1^*, \dots, \boldsymbol{p}_{QH}^*\}$, where $R = QH$. Then, we sample a Gaussian noise for each point $\boldsymbol{p}_r^* \in \mathcal{P}^*$ and compute the initial noisy point cloud $\mathcal{P}^T$ from $\mathcal{P}^*$ with Eq. (6.8). Finally, we calculate the T denoising steps by predicting the noise at step t from Eq. (6.4), and then denoising it using Eq. (6.2) to obtain the complete scan $\mathcal{P}' = \mathcal{P}^0$.

Note that, as long as $\mathcal{P}^T$ is "noisy enough" to resemble $\mathcal{G}^T$ as seen during training, the generation process is the same independent of using $\mathcal{P}^*$ or the ground truth $\mathcal{G}$ to sample the initial $\boldsymbol{x}^T$.

6.1.5 Noise Prediction Regularization

DDPMs use a leveraged formulation to train the model to predict only the noise added to the data. This formulation has only to optimize an $\mathcal{L}_2$ loss between the added noise and the model prediction. However, this formulation optimizes the model to precisely predict the noise added to each point, ignoring the overall distribution of the noise sampled.

Given that the added noise $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, it is reasonable to expect that the prediction $\boldsymbol{\epsilon}_\theta(\mathcal{G}^t, \mathcal{P}, t) \approx \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. However, the model predicts a peaky distribution far from the expected, as shown in Fig. 6.3. The predicted noise starts

with a mean far from zero and with a large standard deviation. As the denoising starts the mean gets closer to zero but the standard deviation is still far from one. Therefore, we propose a regularization to approximate $\boldsymbol{\epsilon}_\theta(\mathcal{G}^t, \mathcal{P}, t)$ to $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. We compute the mean $\overline{\boldsymbol{\epsilon}}_\theta$ and the standard deviation $\hat{\boldsymbol{\epsilon}}_\theta$ over $\boldsymbol{\epsilon}_\theta(\mathcal{G}^t, \mathcal{P}, t)$ and calculate the regularization losses:

$$\mathcal{L}_{\text{mean}} = \overline{\boldsymbol{\epsilon}}_\theta{}^2 \quad \text{and} \quad \mathcal{L}_{\text{std}} = \left(\hat{\boldsymbol{\epsilon}}_\theta - 1\right)^2, \tag{6.9}$$

then our final loss becomes:

$$\mathcal{L} = \mathcal{L}_{\text{diff}} + \lambda \left(\mathcal{L}_{\text{mean}} + \mathcal{L}_{\text{std}}\right), \tag{6.10}$$

where λ is a weighting factor.

With this regularization, we aim at smoothing the peaky distribution from the predicted noise, and approximating it to the expected target distribution, i.e., $\boldsymbol{\epsilon}_\theta(\mathcal{G}^t, \mathcal{P}, t) \approx \mathcal{N}(\mathbf{0}, \boldsymbol{I})$.

6.1.6 Refinement Network

Despite the impressive results of DDPMs, the denoising process still demands time since it has to predict all the T steps. Recent efforts [80, 103, 104, 118, 154, 164] focus on increasing the inference speed. However, by reducing the inference time, the generation quality may also decrease. Besides, processing 3D scene-scale data demands many computational resources. This limitation hinders the number of points we can generate to represent the complete scene. Therefore, we follow the refinement upsampling scheme proposed by Lyu et al. [107]. We train an additional model to refine the scene generated by the diffusion process while upsampling it by predicting κ offsets $\boldsymbol{o}_\kappa \in \mathbb{R}^3$ for each point in the completed scene $\mathcal{P}'$. Then, we add the offsets to the completed scene points as $\{\boldsymbol{p}'_r + \boldsymbol{o}_0, \ldots, \boldsymbol{p}'_r + \boldsymbol{o}_\kappa\}$, $\forall\, \boldsymbol{p}'_r \in \mathcal{P}'$ refining the diffusion completion while upsampling it.

6.1.7 Noise Predictor Architecture

We parameterize the denoising process using a MinkUNet [34] as the noise predictor which uses sparse convolutions to process 3D data. To encode information from the conditioning scan $\mathcal{P}$, we use the encoder part from MinkUNet [34] with the same architecture as the noise predictor. The encoder downsamples $\mathcal{P}$ to a smaller version $\mathcal{C} = \{\boldsymbol{c}_{h'} \in \mathbb{R}^{N^\mathcal{C}} \mid 1 < h' < H'\}$, where $H' < H$ and $N^\mathcal{C}$ is the encoder output embedding size. To encode the denoising step t, similar to previous work [214], we use a sinusoidal positional encoding to compute the temporal embedding $\tau \in \mathbb{R}^{N^t}$. Then, before each layer l in the denoising model, we compute

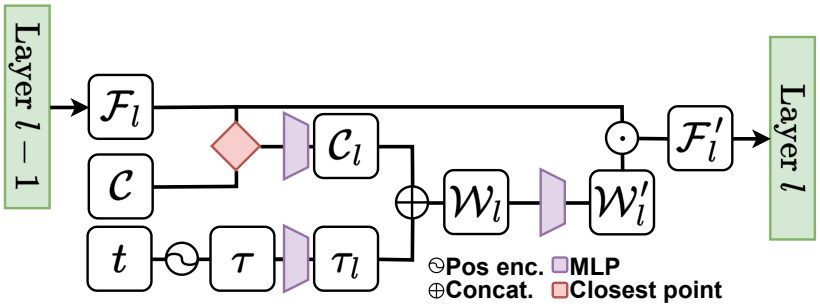


Figure 6.4: Diagram of the conditioning in each layer l.

the closest point between the layer input points $\mathcal{F}_l = \{\boldsymbol{f}_{h'_l} \in \mathbb{R}^{N^l} \mid 1 < h'_l < H'_l\}$ and the conditioning embeddings $\mathcal{C}$ to get a per-point guidance, passing it over an MLP to get $\mathcal{C}_l = \{\boldsymbol{c}_{h'_l} \in \mathbb{R}^{N^l} \mid 1 < h'_l < H'_l\}$. Then, we compute $\tau_l \in \mathbb{R}^{N^l}$ from the temporal embedding τ through another MLP, and concatenate τ_l to each point in $\mathcal{C}_l$ obtaining $\mathcal{W}_l = \{\boldsymbol{w}_{h'_l} \in \mathbb{R}^{2N^l} \mid 1 < h'_l < H'_l\}$. Finally, we use one more MLP layer to project $\mathcal{W}_l$ to the layer feature dimension N^l and get $\mathcal{W}'_l$. Then, we compute $\mathcal{F}'_l = \mathcal{W}'_l \odot \mathcal{F}_l$ as an element-wise multiplication, which is then feed as the input to layer l, as depicted in Fig. 6.4. As the refinement network, we use the same MinkUNet [34] architecture used for the noise predictor without the conditioning encoder. Fig. 6.5 depicts the diagrams for the noise predictor and condition encoder models.

6.2 Experimental Evaluation

In this section, we compare our novel scene completion method with prior diffusion and non-diffusion based approaches. We demonstrate in our experiments that our local point-diffusion method enables more detailed generation compared to standard point-diffusion methods. Also, we show in our evaluation that our method can generate more fine-grained details compared to previous approaches, achieving competitive performance even when evaluating on data collected from a LiDAR sensor different from the one used for training.

6.2.1 Implementation Details and Experimental Setup

Our experimental setup follows scene completion evaluation methods. We compute the scene completion results for the different methods and evaluate how close these results are to the target complete scene. Our evaluation assesses this scene reconstruction in two ways, by comparing how close the predicted completed point clouds are from the ground truth, and comparing the occupancy between the completed scene and the ground truth data.

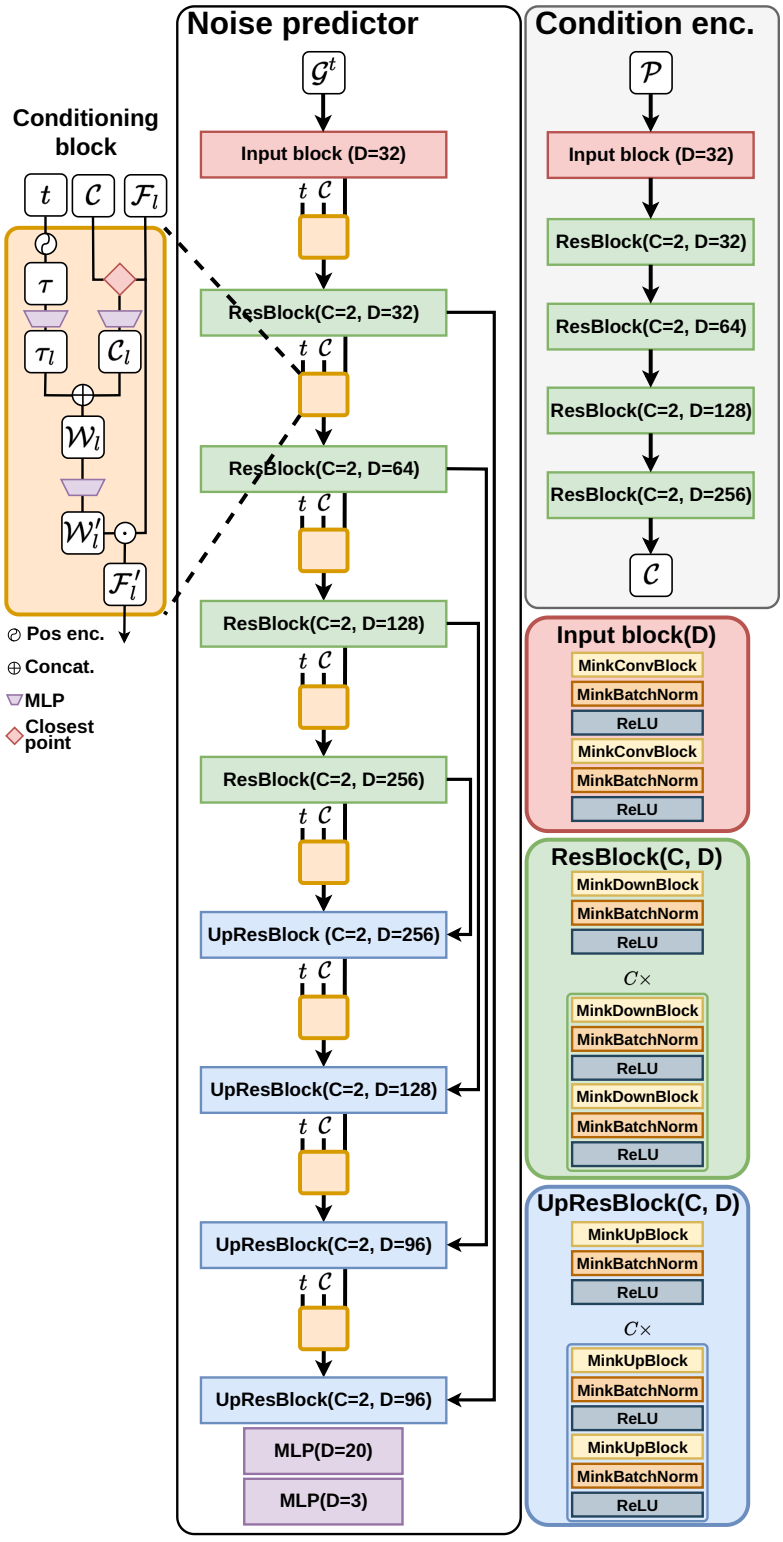


Figure 6.5: Noise predictor and condition encoder models architecture. The condition encoder receives the scan $\mathcal{P}$ and computes the conditioning point cloud $\mathcal{C}$. From t, we compute the positional embedding τ with a dimension $N^t = 96$. At each layer l, we give $\mathcal{C}$ and τ to the conditioning block together with the layer input features $\mathcal{F}_l$ to get $\mathcal{F}'_l$, which is then feed as input to the layer l.

Datasets. For training our DDPM, we used the SemanticKITTI dataset [5, 54], an autonomous driving benchmark with point-wise annotations over sequences of LiDAR scans collected in an urban environment. To generate the ground truth complete scans, we used the dataset poses to aggregate the scans in the sequence and remove moving objects with the semantic labels, building a map for each sequence. For evaluation, we used the validation set from SemanticKITTI, i.e., sequence 08. Additionally, we used sequence 00 from the KITTI-360 dataset [94] and collected our own data with an Ouster LiDAR OS-1 with 128 beams to further compare the approaches.

For SemanticKITTI and KITTI-360, we used the ground truth poses to build the map, and for our data, we used KISS-ICP [178] to get the scan poses for our sequence. To remove the moving objects from the map in KITTI-360 and our data, we used an off-the-shelf moving object segmentation [121]. To compute the evaluation metrics, for each scan in the sequences, we remove the moving objects using the semantic labels using only the static points as input to the scene completion methods. Then, we evaluate the completed scene by comparing it with the corresponding region in the ground truth map.

Training. We train our model for 20 epochs, using only the training set from SemanticKITTI. As optimizer, we used Adam [83] with a learning rate of 10^{-4} decreased by half every 5 epochs, and decay of 10^{-4}, with batch size equal to 2. For the diffusion parameters, we used $\beta_0 = 3.5 \cdot 10^{-5}$ and $\beta_T = 0.007$, with the number of diffusion steps $T = 1{,}000$, linearly interpolating between β_0 and β_T to define $\beta_1, \ldots, \beta_{T-1}$. We set the noise regularization $\lambda = 5.0$, and the classifier-free probability $\rho = 0.1$. For the MinkUNet [34] parameters, we set the quantization resolution to 0.05 m. For each input scan, we define the scan range as 50 m and sample 18,000 points with farthest point sampling. For the ground truth, we randomly sample 180,000 points without replacement. For the refinement network we use $\kappa = 6$ as the number of offsets.

Inference. During inference, we use DPMSolver proposed by Lu et al. [103], reducing the number of denoising steps T from 1,000 to 50. Besides, we set the classifier-free conditioning weight to $s = 6.0$. To maintain the same amount of points used during training, we again use the scan max range as 50 m and sample 18,000 points with farthest point sampling. Furthermore, as explained in Sec. 6.1.4, we set $Q = 10$ to define the input noisy scan $\mathcal{P}^*$.

Baselines. We compare our method with different scene completion methods, LMSCNet [147], PVD [214], Make It Dense (MID) [176], and LODE [91]. For all baselines, we used their official code and the provided weights also trained on SemanticKITTI. For PVD, we trained the approach with SemanticKITTI with their default parameters. We also follow their data loading, where the point clouds are normalized before the diffusion process. LMSCNet [147] and

Table 6.1: Mean chamfer distance and Jensen-Shannon divergence evaluation on validation set from SemanticKITTI.

Method	CD [m] ↓	JSD_{BEV} [m] ↓
LMSCNet [147]	0.641	0.431
LODE [91]	1.029	0.451
MID [176]	0.503	0.470
PVD [214]	1.256	0.498
Ours	**0.375**	**0.416**

LODE [91] are limited to a fixed voxel grid of 51.2 m × 51.2 m × 6.4 m. Given that our point cloud generation is done over a scan with a radius of 50 m, we divide the input scan into four quadrants over the 360° LiDAR field of view, generating the complete scene over each quadrant and finally gathering them together as the final prediction. All baselines and our method were trained only with SemanticKITTI, and later evaluated on SemanticKITTI, and on KITTI-360 and our data without fine-tuning.

6.2.2 Scene Reconstruction

In this experiment we evaluate how close is the predicted scan completion from the expected complete scene. To do so, we quantify it with two metrics, the Chamfer distance (CD) and the Jensen-Shannon divergence (JSD). The Chamfer distance evaluates the completion at point level, measuring the level of detail of the generated scene by calculating how far are its points from the expected scene. The JSD compares the points distribution between the generated and the ground truth scene. For the JSD, we follow the evaluation done by Xiong et al. [197], where the scene is first voxelized with a grid resolution of 0.5 m and then projected to a bird's eye view, evaluating over this projection.

Tab. 6.1 shows the results comparing our approach with previous state-of-the-art methods, where our method achieves the best performance in both metrics. First, we can notice that the state-of-the-art shape generation diffusion method, PVD, achieves the lowest performance, showing that current 3D diffusion methods cannot directly be applied to scene-scale data. The best performance of our method over the CD metric can be explained by the fact that our method operates directly on the points, which enables it to produce a more detailed scene compared to the baselines. The scene representation from SDF-based methods inherits artifacts from the surface approximation and voxelization, impacting the details in the reconstructed scene and therefore decreasing their performance with respect to the CD. The JSD evaluates the reconstructed scene points distribution over

Table 6.2: Mean chamfer distance and Jensen-Shannon divergence evaluation on KITTI-360 sequence 00 and our data.

	KITTI-360		Our data	
Method	CD [m] ↓	JSD_{BEV} [m] ↓	CD [m] ↓	JSD_{BEV} [m] ↓
LMSCNet [147]	0.979	0.496	0.826	0.439
LODE [91]	1.565	0.483	**0.387**	0.389
MID [176]	0.637	0.476	0.475	0.379
Ours	0.564	0.459	0.518	0.360
Ours refined	**0.517**	**0.446**	0.471	**0.341**

a voxelized grid comparing the overall scene distribution between the generated and the expected completion. Even though our method is not optimized over a voxel representation, we still achieve the best performance, showing that our scene completion is at the same time closer to the expected point distribution and can yield more details.

Tab. 6.2 compares the results of the scene completion methods on KITTI-360 and our collected data. Due to the poor performance of PVD over KITTI dataset, we do not evaluate it on those datasets. For KITTI-360 we notice the same behavior as in Tab. 6.1, where our method achieves the best performance in both metrics. When evaluating in our data, the performance of the SDF-based methods improve. This is expected since our data has denser point clouds, which is an advantage for such methods since they rely on the input points to approximate a surface to represent the scene. However, our method still achieves the best performance on the JSD metric and competitive performance on the CD metric. This evaluation shows that our method can still achieve scene completion over different datasets without fine-tuning since its generation is conditioned to the input scan. In Fig. 6.6 and Fig. 6.7 we can compare the scene completion generation between the methods. We can see that the diffusion baseline, PVD, fails on generating scene-scale data. SDF-based methods inherits artifacts from the voxelization, our method, especially after the refinement, can generate a scene closer to the expected, following the structural information from the input scan.

6.2.3 Scene Occupancy

In this experiment, we assess the scene completion by evaluating the occupancy of the predicted scene compared with the ground truth. To do so, we follow the evaluation proposed by Song et al. [165] where the intersection-over-union (IoU) is computed between the predicted and ground truth voxelized scene, classifying

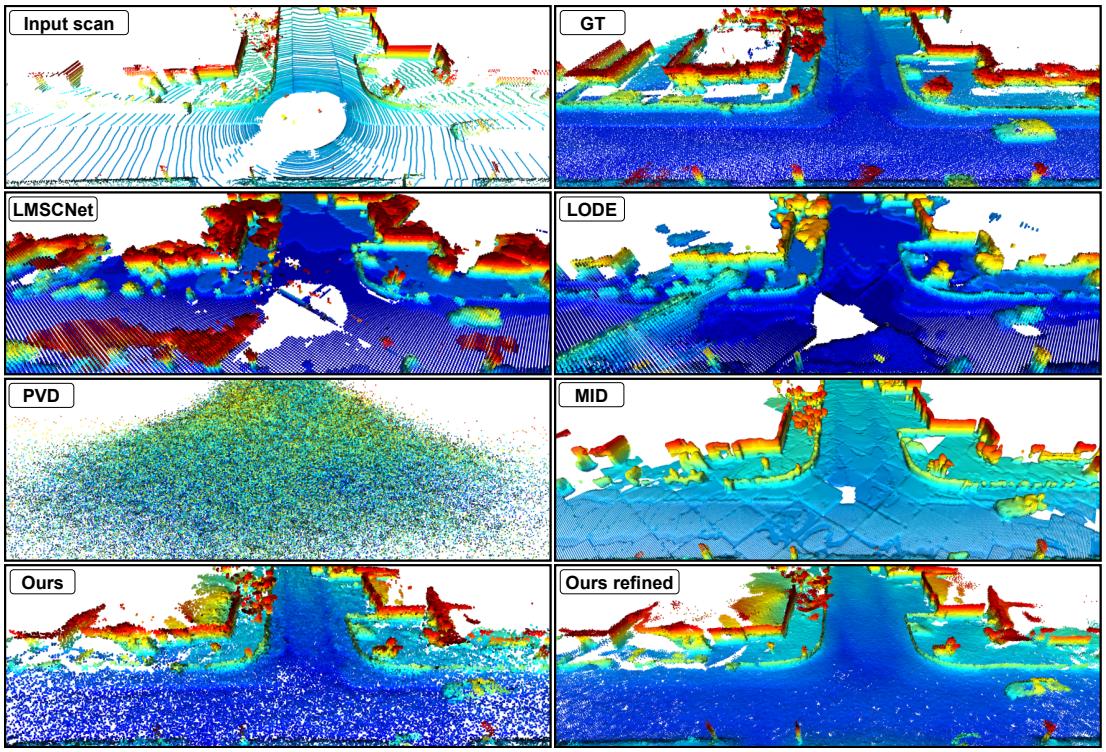


Figure 6.6: Qualitative results on one scan from KITTI-360. Colors depict point height normalized by the height range of each point cloud.

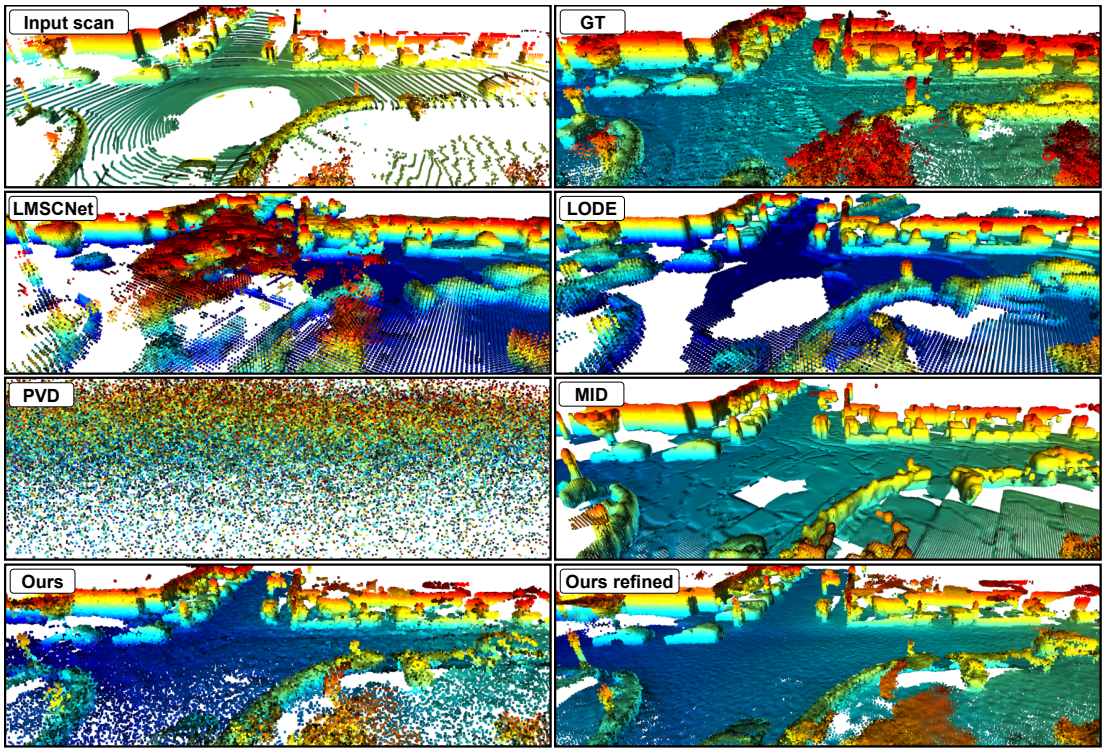


Figure 6.7: Qualitative results on one scan from KITTI. Colors depict point height normalized by the height range of each point cloud.

Table 6.3: Completion metric where the IoU is computed against the ground truth and prediction grids with different resolutions.

	IoU [%] ↑ Grid resolution [m]		
Method	0.5	0.2	0.1
LMSCNet [147]	30.83	12.09	3.65
LODE [91]	**33.81**	16.39	5.0
MID [176]	31.58	22.72	13.14
PVD [214]	15.91	3.67	0.6
Ours	31.47	16.79	4.67
Ours refined	32.43	**22.99**	**13.40**

each voxel as occupied or not. In this evaluation, we compute the IoU at three different voxel resolutions, i.e., 0.5 m, 0.2 m, and 0.1 m. With a voxel size of 0.5 m, we evaluate the occupancy over the coarse scene, where the scene details are not considered. As we decrease the voxel size, more fine-grained details are evaluated.

Tab. 6.3 shows the IoU of our method compared to the baselines at the different voxel resolutions. First, the diffusion baseline PVD has the lowest performance overall. This again shows that current state-of-the-art 3D shape completion diffusion methods cannot be directly applied to scene-scale data. At a higher voxel size, our approach stays behind some SDF-based baselines. This is reasonable in this evaluation since SDF methods use a voxel representation to reconstruct the scene. Therefore, its reconstruction is equally distributed over the point cloud and its voxel representation is denser compared to our result voxelized. As we decrease the voxel size, the baselines performance drops. At the lowest resolution, our method outperforms the baselines. LMSCNet [147] and LODE [91] are bound to a voxel resolution of 0.2 m, therefore with a voxel size of 0.1 m their performance drops drastically. Make It Dense [176] was trained with a voxel size of 0.1 m, however, our method still outperforms it at this resolution. This shows the advantage of our approach. Since it is trained at point level, it can produce a more detailed scene, not limited to a fixed grid size.

Due to the poor performance of PVD on SemanticKITTI, we compared our method only with non-diffusion approaches for the other two datasets. In Tab. 6.4, we compare the results from our method with the baselines for KITTI-360 and our own collected data. For KITTI-360 our method outperforms all baselines in all evaluated resolutions. For our data, SDF-based method have advantages due to the denser point clouds as discussed in Sec. 6.2.2, still, our method achieves competitive performance compared to the baselines especially at lower voxel size.

Table 6.4: Completion metric where the IoU is computed against the ground truth and prediction grids with different resolutions.

	IoU [%] ↑					
	KITTI-360 Grid resolution [m]			Our data Grid resolution [m]		
Method	0.5	0.2	0.1	0.5	0.2	0.1
LMSCNet [147]	26.17	9.21	2.88	21.55	8.44	2.38
LODE [91]	33.06	15.24	4.68	39.32	18.56	6.15
MID [176]	33.05	21.32	11.30	**41.37**	**29.88**	**18.05**
Ours	33.23	17.55	4.88	36.30	18.27	4.90
Ours refined	**33.43**	**22.04**	**11.84**	36.91	27.51	15.28

This suggests that our approach can achieve a detailed scene reconstruction, and it is able to generate data from a different dataset than the one it was trained on, since its generation is guided by the input LiDAR scan.

6.2.4 Noise Regularization

Next, we evaluate the impact of the proposed noise prediction regularization on the generated scene. We compare the predicted noise distribution with different regularization weights λ in Fig. 6.8 from the 10 denoising steps in one scan as in Fig. 6.3. As seen, without the regularization, i.e., $\lambda = 0$, the predicted noise starts far from the expected distribution, with a mean of around -9.0 and a standard deviation of about 526. As we denoise the input, the distribution gets closer to the expected, however, still with a high standard deviation. When we add our proposed regularization, the model already starts predicting a more reasonable noise distribution from the beginning, stabilizing the denoising process. From this evaluation, we noticed that using $\lambda = 5.0$ achieved a more stable distribution over the denoising steps. Fig. 6.9 compares the scene completion with the noise predictor trained with different regularization weights. With $r = 0.0$, the model can generate structural information with a noisy aspect, and, in this example, the points from the two parked cars are mixed together without a clear boundary. With $r = 1.0$, a less noisy scene completion is generated, but still, the surfaces in the structure present a noisy aspect. When comparing $r = 3.0$ and $r = 5.0$, both generated scene depicts a more detailed and less noisy scene, compared with lower regularization weights r. However, using $r = 5.0$ achieves more fine-grained structural details. The surfaces in the scene appear to have a flatter aspect, and the sidewalk curbs seem better defined. Also, the two parked cars retain more

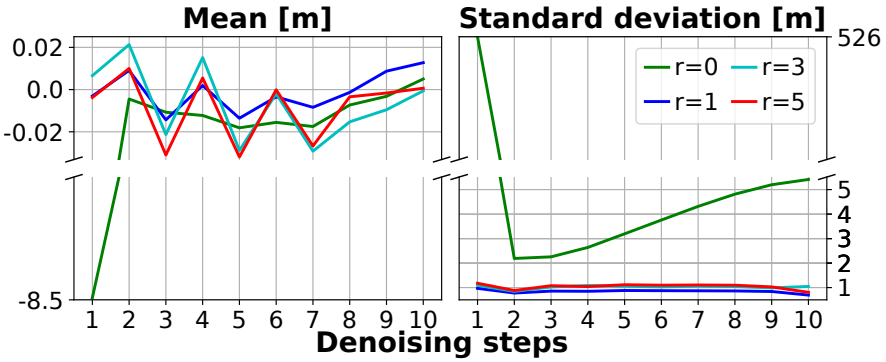


Figure 6.8: Mean and standard deviation of the predicted noise ϵ_θ over different regularization weights. In this experiment we use DPMSolver [103] to reduce the denoising steps from 1,000 to 10.

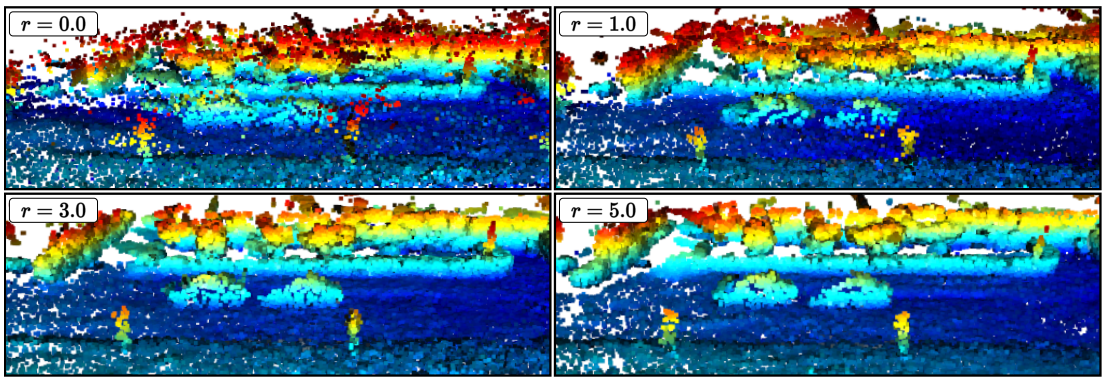


Figure 6.9: Comparison between results with different regularization weights r.

details, e.g., the windows space.

To evaluate how the regularization impacts the data generation, we compare the model performance over a short sequence from the SemanticKITTI validation set. We run the scene completion pipeline every one hundred scans without using the refinement network, evaluating only the regularization influence over the noise predictor. In Tab. 6.5, we compute the chamfer distance to compare the impact of the regularization over the quality of the generated scene. As we increase the regularization, the generation quality improves. Despite $\lambda = 3.0$ achieving a slightly better result in this evaluation, we stick with $\lambda = 5.0$ due to the analysis of the noise distribution from Fig. 6.8, and the qualitative comparisons in Fig. 6.9.

6.2.5 Condition Weights

This section compares the scene completion quality using different condition weights s qualitatively and quantitatively. Fig. 6.10 shows the qualitative comparison between the scene completion with different conditioning weights. When setting $s = 0.0$ we have the unconditional generation. In this case, the generated

Table 6.5: Mean chamfer distance over a short sequence from the validation set of SemanticKITTI.

λ	0.0	1.0	3.0	5.0
CD [m] ↓	0.529	0.470	0.441	0.445

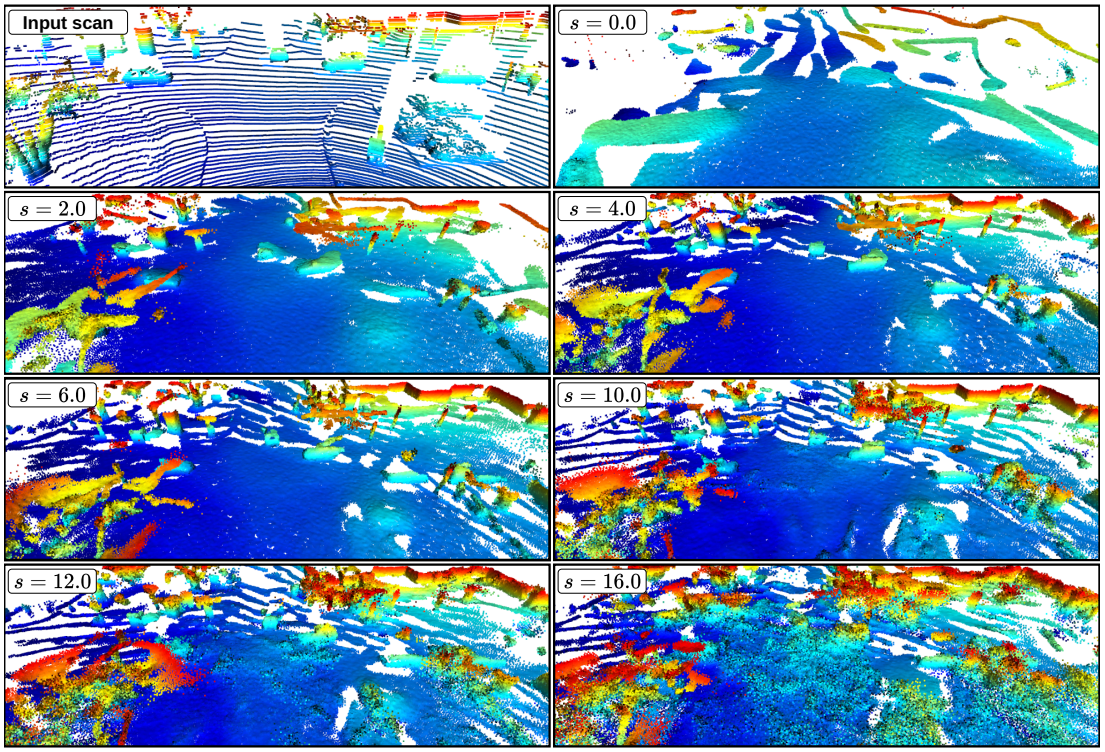


Figure 6.10: Comparison between results with different conditioning weights s.

scene has a flat surface distributed over the input scan borders without retaining structural information. As we increase s, the structure details are better defined. With $s = 2.0$ and $s = 4.0$ more details are generated but with a smooth aspect. When using $s = 6.0$ the generation follows structural information from the input scan and defines sharper boundaries over the structures. With $s = 10.0$ and $s = 12.0$, the generated scene gets noisy, generating artifacts over the scene.

We also evaluate the influence of the conditioning weight s in Tab. 6.6. As in Tab. 6.5, we compute the chamfer distance over the scene completion and the ground truth over a short sequence from the SemanticKITTI validation set, where we generate every one hundred scans. In this evaluation, having $s = 6.0$ and $s = 10.0$ achieves basically the same performance. However, from the qualitative evaluation presented in Fig. 6.10, we used $s = 6.0$ in the main experiments since it achieved the best performance qualitatively and quantitatively.

Table 6.6: Mean chamfer distance over a short sequence from the validation set of SemanticKITTI with different conditioning weights s.

s	0.0	2.0	4.0	6.0	10.0	12.0	16.0
CD [m] ↓	0.737	0.543	0.454	**0.433**	**0.432**	0.435	0.450

6.3 Related Work

Scene completion has been extensively studied in the literature, particularly in the robotics community, to achieve a more complete representation from sparse sensor measurements. Given the extensive availability of 3D LiDAR datasets with semantic annotations, the scene completion task is often reframed as semantic scene completion. Prior works mainly focus on SDF representations or image projections to approximate the dense scene from a sparse 3D LiDAR point cloud. Differently, in this chapter, we reformulated the task as a generative task, where instead of approximating the dense scene, we use a DDPM to generate the missing parts of the point cloud. In this section, we review related scene completion approaches and recent studies on diffusion models used as the basis for the method described in this chapter.

Scene completion aims at inferring missing 3D scene information given an incomplete 3D sensor measurement. This inference of unseen information is helpful for perception tasks [197], localization [176], or navigation [140], as it provides more complete information compared to the sparse and limited information collected by 3D sensors. To achieve this, some works [53, 110, 111, 199] have tackled this task by jointly extracting information from paired RGB images and LiDAR point clouds, predicting a depth map from an RGB image supervised by the LiDAR data. Using the LiDAR data as supervision, the model then learns to predict depth information for each pixel in the RGB image, predicting a dense point cloud from the RGB data. Differently, other methods [91, 176] approach the problem by optimizing an SDF given only the LiDAR measurements. The SDF represents the scene as a voxel grid, where each voxel stores its distance to the closest surface in the scene. In this case, the model is optimized to approximate a surface that fits the scene, given the sparse LiDAR point clouds. From the SDF, the LiDAR scan can be drawn as a dense and complete point cloud of the scene. However, such SDF methods are limited by the voxel resolution at which the model is trained and lose details in the scene due to the voxel discretization. Differently, the method presented in this chapter operates directly over the points, leveraging the generative properties of DDPMs to complete the unseen data without relying on a voxel grid representation.

Semantic scene completion has gained significant interest recently due to

the availability of large datasets with semantic labels [4, 5, 14, 51, 54, 94, 167]. This task extends the scene completion task by predicting a semantic label for each occupied voxel [91, 146, 147]. Such approaches enable the completion of the scenario measured by a LiDAR sensor with the prediction of semantic information within its voxels. As for scene completion, those methods are mainly based on SDF representations, approximating the scene as a surface given the training voxel resolution, while predicting a semantic class for each voxel. Despite providing semantic information along with a complete scene representation, these methods are also tightly bound to the voxel grid resolution, which typically has a low resolution due to memory limitations. Given the similarity between the two tasks, most recent scene completion approaches directly formulate the problem to also predict a semantic class within the dense scene voxels. The approach described in this chapter still only leverages scene completion, without considering semantic classes, and focuses mainly on the geometry completion of the scene. In Chapter 7, we discuss an approach for both semantic scene completion and novel semantic scene generation.

Denoising diffusion probabilistic models have gained attention due to their high-quality results in image generation [39, 68, 127, 138, 144, 148, 215, 216]. Ho et al. [68] define a diffusion process to corrupt an image, iteratively adding noise to it over a fixed number of steps, such that the corrupted image approximates a Gaussian distribution. Then, a network is trained to reverse this process by predicting the noise added at each step. This formulation enables the generation of novel samples by sampling random Gaussian noise and using the trained network to denoise it until arriving at a novel sample from the training data distribution. Given the quality of the generated images, conditioned diffusion models have gained even more relevance due to the possibility of generating data towards a specific input condition [3, 69, 211]. Those conditioned DDPMs are trained together with a conditioning token to guide the generation process towards a specific target. Although achieving impressive results, the drawback of DDPMs is typically the time required during the denoising process, given its iterative nature. For this reason, many efforts have been made to achieve faster generation [80, 103, 104, 118, 154, 164]. Some methods tackle this computational cost problem by distilling the denoising model, training the network to predict multiple denoising steps simultaneously [118, 154]. Other approaches analytically approximate the denoising steps solution, reducing the number of steps needed without requiring further training [80, 103, 104, 164].

Diffusion models for 3D data have been investigated due to their promising performance in the image domain. Early methods [106, 107, 214] focus on single object shapes, aiming to generate or complete object shapes. Compared to object-level diffusion, fewer works [89, 126, 219] target the generation of real-

world data. Some works [126, 219] rely on projecting 3D data into an image-based representation, such as range images, so that the methods proposed in the image domain can be directly applied. For such approaches, 3D scene completion cannot be directly achieved, as reprojecting the image to the 3D world results in some regions missing information due to occlusions in the projected point cloud. Lee et al. [89] achieve scene-scale 3D data generation using a discrete diffusion model formulation and a fixed voxel grid representation of the environment. In this formulation, the DDPM is trained to predict for each voxel in the fixed 3D grid, whether it is occupied and its corresponding semantic class. Despite overcoming the limitations of image-projection methods, the resolution of this approach is limited since the DDPM operates over all voxels in the defined 3D grid, regardless of whether they are occupied or not. Different from previous works, our method operates directly at the point level and does not rely on a grid representation or projection to the image domain. Therefore, it is not bound by a voxel resolution and can deal with occluded regions inherent in LiDAR data.

Given the recent advances in DDPMs for data generation, this chapter proposes a point-level formulation of the denoising diffusion process, achieving competitive performance in scene-scale diffusion scene completion. Our formulation enables the use of DDPMs to generate scene-scale, real-world-like data without relying on any discretization or projection of the LiDAR data.

6.4 Conclusion

In this chapter, we proposed a novel point-level denoising diffusion probabilistic model to achieve scene completion for autonomous driving data. Our method enables the use of DDPMs for generating dense point clouds from single sparse LiDAR measurements, achieving a complete scene representation. This dense scene representation allows for sampling sensor-specific point clouds, serving as a proxy for bridging the domain gap between different LiDAR sensors. We exploit the generative capabilities of DDPMs to generate the missing parts from a single sparse LiDAR scan. We reformulate the diffusion process as a local problem. Each point is defined as the origin of the sampled Gaussian noise, allowing for an iterative denoising process to gradually predict offsets and reconstruct the scene from the input noisy LiDAR scan. This formulation enables the processing of scene-scale 3D data, retaining more details during the denoising process, distinguishing it from previous state-of-the-art diffusion approaches by allowing the generation of scene-scale 3D data at the point level.

We compare the method proposed in this chapter with both diffusion and non-diffusion methods, demonstrating that this approach achieves more accurate scene completion compared to prior methods. We evaluate all methods regard-

ing the scene reconstruction details and scene occupancy. The experiments show that our method can reconstruct the scene with more details compared to prior scene completion approaches, as it generates the missing parts rather than approximating them with an SDF representation. Furthermore, we evaluate our approach when completing the scene with a point cloud collected using a different LiDAR sensor than the one used to collect the training data. In this case, our method also achieves detailed reconstruction, with performance comparable to prior state-of-the-art works. Our approach enables the use of point diffusion methods for 3D scene-scale data, generating a dense scene representation from a sparse LiDAR scan, bridging the domain gap between different LiDAR sensors.

Although valuable, the dense scene representation achieved by the method proposed in this chapter does not account for the labels from the LiDAR scan. Therefore, the point-wise labels from one LiDAR sensor cannot be easily transferred to another LiDAR sensor. In the next chapter, we will discuss how to use DDPMs to not only generate a complete scene from a LiDAR point cloud but also assign a semantic label to each point in the dense scene representation, achieving semantic scene completion. Additionally, we leverage the generative capabilities of DDPMs to generate novel, semantically annotated scenes. We then use these unconditionally generated semantic scenes to train a semantic segmentation network with both real and randomly generated samples, assessing the use of generative models to generate more labeled data for training perception models.

Chapter 7

Generating 3D Semantically Annotated Training Data

A dense scene representation can be useful for perception systems to generalize across different LiDAR sensors. However, the scene completion method discussed in the previous chapter does not account for the semantic information in the generated complete scene. Even though helpful, having only a geometric dense representation of the scene still does not allow for transferring annotated data between different LiDAR sensors. To account for that, a model should be able to achieve both, complete the scene and generate the labels for it, achieving semantic scene completion. Apart from using generative models to achieve semantic scene completion from a single LiDAR scan, these models could also be trained to generate random novel semantically annotated scenes. This would enable the generation of more annotated data to be used to train perception models, either conditioned on LiDAR scans or completely novel scenes.

The semantic scene completion task has been studied in the robotics and computer vision communities recently [78, 93, 181, 195]. These works focus on predicting a dense scene representation with its semantic classes from a single LiDAR point cloud. Such complete scene representation with semantic information can be exploited to bridge the domain gap between point clouds collected with different LiDAR sensors. Specifically, given a complete scene with its semantic classes, prior ray casting domain transfer works [2, 81, 88, 159, 203] could directly be applied to those dense point clouds, generating sensor-specific annotated 3D data.

As discussed in Chapter 6, generative methods [129] present advantages compared to standard scene completion approaches, achieving more detailed scene completion. Recent works on DDPMs investigate latent diffusion approaches [10, 148], enabling more efficient computation and generating more complex samples.

Latent diffusion approaches first train a variational autoencoder (VAE) network to encode the data into a latent representation. Then, a DDPM is trained to generate samples from this learned VAE latent space. The sampled latents are then decoded by the VAE decoder to retrieve samples from the original data distribution. This latent diffusion scheme is exploited in our case to achieve semantic scene completion conditioned on LiDAR scans, or to generate novel annotated dense point clouds without any prior condition.

Using DDPMs to generate novel 3D point clouds with semantic annotation has been studied in recent works [89, 90, 99, 119, 145]. These methods leverage different ways to enable 3D semantic scene generation. They either rely on image projections of the real 3D scenes to train the latent diffusion model [89, 90], or on intermediate coarser scene generation [99, 119, 145], where multiple decoupled DDPMs are trained separately at different resolutions of the target data, generating novel samples via a hierarchical generation. However, both projection and hierarchical generation approaches can degrade the generated results due to the loss of information within their intermediate representations. Besides, despite generating scenes with semantic labels, none of the prior methods investigate the use of the generated annotated scenes in real-world perception tasks.

In this chapter, we follow recent latent diffusion approaches to enable conditional and unconditional 3D semantic scene generation. However, distinct from prior works, we avoid image projections or intermediate coarser scene generation by explicitly supervising the VAE decoder upsampling layers to learn how to prune the scene at each upsampling resolution, while also predicting its semantic class. This supervision over the VAE decoder layers optimizes the model to encode meaningful information at each upsampling resolution, achieving a more accurate scene reconstruction. With the trained VAE, we then train a DDPM over the learned latent space to achieve both conditional and unconditional generation. The conditional generation enables the transfer of annotated data between different LiDAR sensors by first achieving semantic scene completion from an input LiDAR scan and then sampling sensor-specific labeled point clouds from the generated dense scene. The unconditional generation enables the sampling of novel annotated scenes without requiring any input. Both conditional and unconditional scene generation enable the sampling of semantically annotated point clouds to be used as training data.

We evaluate our method against previous methods by comparing the generated samples with real-world data, and also assess the usability of our generated samples to train a network for semantic segmentation [128]. We show that our generated samples are not only closer to real-world data compared to prior works, but also improve the performance of the semantic segmentation network by using the generated annotated scenes as training data together with the real data.

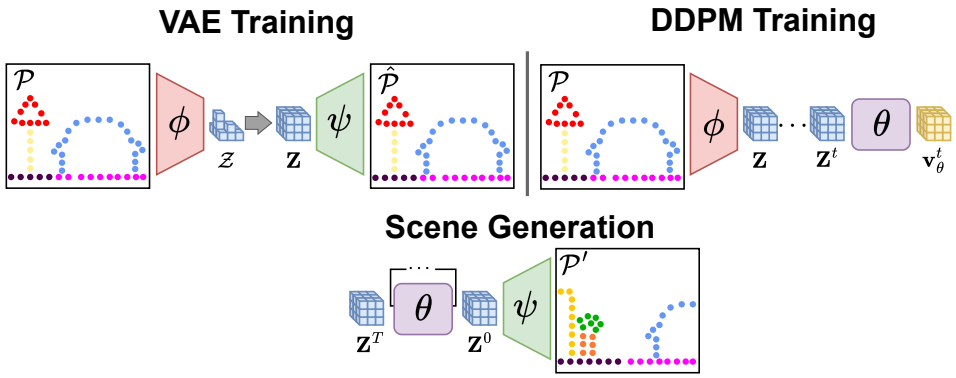


Figure 7.1: Our scene generation pipeline. First we train a VAE with an encoder ϕ and a decoder ψ using the dense scenes $\mathcal{P}$ to reconstruct it as $\hat{\mathcal{P}}$ and to learn the sparse $\mathcal{Z}$ and dense $\boldsymbol{Z}$ latent spaces. Next, we train a DDPM θ over the latent $\boldsymbol{Z}$, sampling a random step t to compute the noisy latent $\boldsymbol{Z}^t$, where the model θ predicts $\mathbf{v}_\theta^t$, following the $\mathbf{v}$-parameterization formulation [154]. Finally, we generate novel scenes by sampling random noise $\boldsymbol{Z}^T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ and denoising it with θ over T times, arriving to $\boldsymbol{Z}^0 = \boldsymbol{Z}_\theta$, which we decode with the VAE decoder ψ to get to the generated scene $\mathcal{P}' = \psi(\boldsymbol{Z}_\theta)$.

7.1 Generating Realistic 3D Semantic Training Data

The main contribution of this chapter is a latent diffusion model for semantic scene generation that achieves more realistic samples by avoiding image projections or intermediate coarser scene generation. We train a VAE model to learn both the scene layout and the semantic information at each decoder upsampling layer, learning the coarse-to-fine scene data nature within a single model. Additionally, we compare the use of synthetic scenes against real data as training labels, and evaluate its use to extend the training set, assessing the impact of synthetic data in real-world applications. In sum, the main contributions in this chapter are (i) a new method for generating 3D scene-scale semantic data without relying on image projections or intermediate coarser scene generation, (ii) that generates more realistic scenes that are better suited for real-world tasks, (iii) improves the performance on semantic segmentation when using both real and our generated samples as training data, and (iv) we compare real and generated data distributions and identify current gaps between generated and real data to be addressed in future work.

7.1.1 Our Approach

In this section, we describe our approach to generate semantic scene-scale data. We first train a VAE to encode the scene with its semantic labels into a descriptive

latent space. Then, we train a DDPM to learn the latent space of the VAE to generate new latent samples. Finally, we use the VAE decoder to turn the sampled latent into the semantic scene. Fig. 7.1 depicts an overview of our pipeline for scene-scale generation.

7.1.2 Semantic Scene Variational Autoencoder

DDPMs are known for being computationally demanding. A more recent approach to simplify the generation process is to train an autoencoder to learn a descriptive latent representation of the target data and then train the DDPM to generate novel samples from the learned latent space [148].

Let us define a voxelized 3D point cloud $\mathcal{P}$ with R voxels with coordinates $\mathcal{P}_\mathcal{C} = \{\boldsymbol{c}_1, \dots, \boldsymbol{c}_R\}$, $\boldsymbol{c}_r \in \mathbb{R}^3$ within a fixed range over $[x, y, z]$, and the features $\mathcal{P}_\mathcal{F} = \{\boldsymbol{f}_1, \dots, \boldsymbol{f}_R\}$, $\boldsymbol{f}_r \in \mathbb{R}^4$ as $\boldsymbol{f}_r = (x, y, z, s)$ where $s \in \{1, \dots, C\}$ is the semantic label for that voxel. We want to train a VAE with an encoder ϕ and a decoder ψ, optimizing both such that $\psi(\phi(\mathcal{P})) = \hat{\mathcal{P}} \approx \mathcal{P}$.

Ren et al. [145] model this training as U independent sparse 3D VAE models trained at different resolutions of $\mathcal{P}$ as $\hat{\mathcal{P}}_u = \psi_u(\phi_u(\mathcal{P}_u))$. Then, U independent DDPMs θ_u are trained to hierarchically generate a novel scene $\mathcal{P}'$ by conditioning the finer resolutions' scene generation to previous coarser stages as $\mathcal{P}'_u = \psi_u\left(\theta_{u-1}\left(\mathcal{N}(\mathbf{0}, \boldsymbol{I}), \mathcal{P}'_{u-1}\right)\right)$, until the final scene resolution. The motivation behind this formulation is to model the coarse-to-fine scene data nature and to simplify the scene-scale data generation. However, this multi-resolution formulation can lead to incremental errors since the U models are trained independently. Hence, we argue that this coarse-to-fine data aspect can be modeled by a single encoder-decoder network sequential downsampling and upsampling layers. Therefore, we train a single 3D VAE model at the $\mathcal{P}$ original resolution.

The encoder ϕ, comprised by consecutive downsampling sparse convolutional layers, receives the voxelized point cloud $\mathcal{P}$ as input and encodes it into the latent representation $\mathcal{Z}$ with coordinates $\mathcal{Z}_\mathcal{C} = \{\boldsymbol{c}_1^{\boldsymbol{z}}, \dots, \boldsymbol{c}_Z^{\boldsymbol{z}}\}$ with $\boldsymbol{c}_z^{\boldsymbol{z}} \in \mathbb{R}^3$ and features $\mathcal{Z}_\mathcal{F} = \{\boldsymbol{f}_1^{\boldsymbol{z}}, \dots, \boldsymbol{f}_Z^{\boldsymbol{z}}\}$ with $\boldsymbol{f}_z^{\boldsymbol{z}} \in \mathbb{R}^{N^z}$, where $Z < R$, and N^z being the latent feature dimension. The latent coordinates $\mathcal{Z}_\mathcal{C}$ are a lower resolution version of the input voxel coordinates $\mathcal{P}_\mathcal{C}$ after the encoder sparse convolutional layers. During the DDPM training, we want to generate novel scenes from scratch, not being constrained by a prior scene shape. Therefore, we pad the latent representation $\mathcal{Z}$ into a dense latent grid $\boldsymbol{Z}$. Given that the voxelized point cloud $\mathcal{P}$ has a fixed range over the axes and a fixed resolution of 0.1 m, the dense latent grid dimensions $\boldsymbol{Z} \in \mathbb{R}^{H \times W \times D \times N^z}$ are known. We initialize $\boldsymbol{Z}$ with zeros and set the occupied cells as $\boldsymbol{Z}(\boldsymbol{c}) = \mathcal{Z}(\boldsymbol{c})\ \forall\, \boldsymbol{c} \in \boldsymbol{Z}_\mathcal{C} \cap \mathcal{Z}_\mathcal{C}$.

The goal of the decoder ψ is to reconstruct $\mathcal{P}$ from the dense latent grid $\boldsymbol{Z}$. To decode and upsample $\boldsymbol{Z}$ would require extensive computational resources due

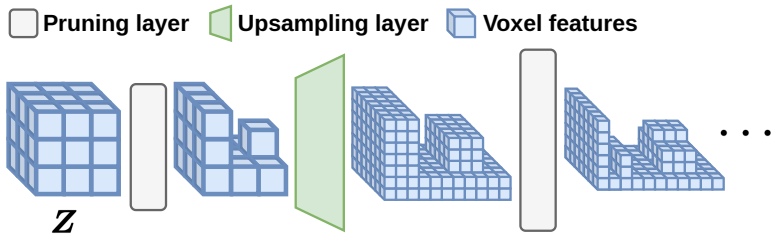


Figure 7.2: Diagram of the pruning process. The pruning layer predicts and prune the unoccupied voxels before each upsampling layer, starting from the dense latent $\boldsymbol{Z}$.

to the exponential increase of the grid size within each upsampling. We overcome this issue by predicting a pruning mask $\hat{\boldsymbol{m}}^l \in [0,1]^{H^l \times W^l \times D^l}$ with a pruning layer before each upsampling convolutional layer l from ψ, as shown in Fig. 7.2. This way, we upsample the dense latent grid $\boldsymbol{Z}$ back to the original input resolution, pruning the unoccupied voxels. At the same time, the supervision of the pruning masks $\hat{\boldsymbol{m}}^l$ optimizes the network towards learning the scene layout at each upsampling convolutional layer l. We then compute $\boldsymbol{Z} = \phi(\mathcal{P})$, $\hat{\mathcal{P}} = \psi(\boldsymbol{Z})$ and the predicted masks $\hat{\mathcal{M}} = \{\hat{\boldsymbol{m}}^1, \ldots, \hat{\boldsymbol{m}}^L\}$ from each upsampling layer l.

7.1.2.1 Pruning Loss

The pruning masks $\hat{\mathcal{M}}$ aim at predicting unoccupied voxels and remove them to reduce memory consumption when upsampling the dense latent $\boldsymbol{Z}$. We also want the decoder ψ to learn the scene layout at each layer l while upsampling the scene. The goal is to learn the coarse-to-fine nature of the scene within a single decoder, previously done with multiple independent VAE models [145]. Therefore, we supervise the prediction of the set of pruning masks $\hat{\mathcal{M}}$ with the target masks $\mathcal{M} = \{\boldsymbol{m}^1, \ldots, \boldsymbol{m}^L\}$, which are computed from the downsampling of the original scene $\mathcal{P}$. We then train the model with the binary cross-entropy and dice losses following recent mask-based segmentation approaches [16, 31, 113, 114] computed between the prediction $\hat{\boldsymbol{m}}^l$ and the target $\boldsymbol{m}^l$. The binary cross-entropy loss aims to predict whether an individual voxel is occupied or not at each upsampling layer l, and is computed as:

$$\mathcal{L}^l_{\text{bce}} = -(\boldsymbol{m}^l \log(\hat{\boldsymbol{m}}^l) + (1 - \boldsymbol{m}^l) \log(1 - \hat{\boldsymbol{m}}^l)). \tag{7.1}$$

The dice loss aims to predict the scene layout by computing the dice coefficient, which approximates the IoU, and maximizes it as:

$$\text{dice}^l = \frac{2\left|\hat{\boldsymbol{m}}^l \cap \boldsymbol{m}^l\right|}{\left|\hat{\boldsymbol{m}}^l\right| + \left|\boldsymbol{m}^l\right|}, \tag{7.2}$$

$$\mathcal{L}^l_{\text{dice}} = 1 - \text{dice}^l. \tag{7.3}$$

Finally, the final pruning loss is computed as the sum between both binary cross-entropy and dice losses for each layer l as:

$$\mathcal{L}_{\text{prune}} = \sum_{l=1}^{L} \lambda_l \left(\mathcal{L}_{\text{bce}}^{l} \left(\hat{\boldsymbol{m}}^l, \boldsymbol{m}^l \right) + \mathcal{L}_{\text{dice}}^{l} \left(\hat{\boldsymbol{m}}^l, \boldsymbol{m}^l \right) \right), \tag{7.4}$$

where λ_l is the weight for pruning loss at the upsampling layer l. Both losses complement each other – while the binary cross-entropy loss optimizes the learned features over individual voxels to predict its occupancy, the dice loss targets the scene layout, optimizing for the masks IoU. With this, we optimize the decoder ψ to learn to prune individual voxels while also learning the entire scene layout at each layer l.

7.1.2.2 Semantic Loss

In addition to the pruning masks, at each upsampling layer l we supervise the VAE model to predict the semantic classes $\hat{\mathcal{S}}^l$ for the voxels not pruned by the predicted pruning mask $\hat{\boldsymbol{m}}^l$. We compute the cross-entropy loss between the set of semantic predictions $\hat{\mathcal{S}}^l$ and the target semantics $\mathcal{S}^l$ computed from the downsampling of the original scene input features $\mathcal{P}_{\mathcal{F}}$. Given the class imbalance intrinsic to scene-scale data, first we supervise the VAE with the weighted cross-entropy loss:

$$\mathcal{L}_{\text{sem}} = -\sum_{l=1}^{L} \frac{1}{R^l} \sum_{r=1}^{R^l} \lambda_{s_r^l} \log \frac{\exp \hat{s}_{r,s_r^l}^l}{\sum_{c=1}^{C} \exp \hat{s}_{r,c}^l}, \tag{7.5}$$

with $\lambda_{s_r^l}$ as the weight of the target class from the training set at the r-th voxel from the R^l voxels in the prediction $\hat{\mathcal{S}}^l$, computed as a ratio to the number of total points in the dataset. Then, for the second half of the training we train the model with the unweighted loss, i.e., $\lambda_{s_r^l} = 1$. This unweighted loss training allows the VAE to learn the class imbalance from the training data to properly account for the scene data distribution.

7.1.2.3 Latent Loss

With the pruning and semantic losses, we optimize the VAE to reconstruct structural and semantic information from the scene. However, for the DDPM to generate meaningful scenes, the learned latent space has to be representative and continuous. Therefore, we regularize encoder ϕ during training using the KL divergence [125] to approximate the encoder latent distribution $q_\phi(\mathcal{P})$ to a target distribution, which is set to $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ as:

$$\mathcal{L}_{\text{latent}} = \mathbb{KL}(q_\phi(\mathcal{P}) \,||\, \mathcal{N}(\mathbf{0}, \boldsymbol{I})), \tag{7.6}$$

where $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$ is a zero-centered isotropic normal distribution. With this latent loss we optimize the latent space towards a continuous representative distribution. This facilitates the training of the DDPM and the generation of new samples from the latent distribution q_ϕ, which later can be decoded by ψ.

7.1.2.4 VAE training

Finally, the VAE is trained to reconstruct the original scene while regularizing the learned latent space. The final loss is a weighted combination of all three losses as:

$$\mathcal{L}_{\text{VAE}} = \lambda_{\text{prune}}\mathcal{L}_{\text{prune}} + \lambda_{\text{sem}}\mathcal{L}_{\text{sem}} + \lambda_{\text{latent}}\mathcal{L}_{\text{latent}}. \tag{7.7}$$

After the VAE is trained, we do a refinement stage similar to previous approaches [129, 145]. We repeat the training, but with added noise to the latent $\boldsymbol{Z}$, optimizing only the decoder ψ to reconstruct the scene $\mathcal{P}$ from the noisy latent. With Eq. (7.7), by learning the pruning at each layer l, we achieved the coarse-to-fine modeling of recent multi-resolution VAE approaches [99, 145] within a single model. In our experiments, we show that we were able to generate more realistic scenes compared to multi-resolution approaches, avoiding the problem from independent model training where the finer models inherit mistakes from coarser stages.

7.1.3 Semantic Scene Latent Diffusion

Given the training data distribution p and the learned VAE latent distribution q_ϕ, we want to train a diffusion model θ to generate a novel dense latent from Gaussian noise as $\boldsymbol{Z}_\theta = \theta(\mathcal{N}(\mathbf{0}, \boldsymbol{I}))$, such that $\boldsymbol{Z}_\theta \sim q_\phi$. Then, the VAE decoder ψ reconstructs the scene from the generated latent $\mathcal{P}' = \psi(\boldsymbol{Z}_\theta)$, where $\mathcal{P}' \sim p$ is a novel scene reconstructed from the generated latent $\boldsymbol{Z}_\theta$.

7.1.3.1 Diffusion Training

DDPMs formulate data generation as a stochastic denoising process [68]. A fixed number of T diffusion steps is defined, and the training data is corrupted by iteratively sampling Gaussian noise and adding it to the data over the T steps. Then, the model is trained to predict the noise added at each one of the T steps, iteratively removing the noise until arriving back to the uncorrupted sample. During inference, Gaussian noise is sampled as the corrupted data at step T, and the model is used to denoise it, generating novel samples.

Latent diffusion training uses the trained VAE to encode a sample $\mathcal{P}$ from the target semantic scene data distribution p into $\boldsymbol{Z} = \phi(\mathcal{P})$. Then, the DDPM training process operates directly on the VAE encoder dense latent grid $\boldsymbol{Z}$.

At each training iteration, a random step t is uniformly sampled from the T diffusion steps. Gaussian noise $\boldsymbol{\epsilon}$ is sampled, and the corrupted data $\boldsymbol{Z}^t$ at step t is computed given pre-defined noise factors $\beta_1, \ldots, \beta_T$, with $\alpha_t = 1 - \beta_t$, and the cumulative product $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$ as:

$$\boldsymbol{Z}^t = \sqrt{\bar{\alpha}_t}\boldsymbol{Z}^0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon},\ \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}). \tag{7.8}$$

From the corrupted latent $\boldsymbol{Z}^t$, the model θ is trained to predict the added noise $\boldsymbol{\epsilon}$. However, predicting only $\boldsymbol{\epsilon}$ leads to slow convergence since the target distribution is only implicitly learned. Salimans et al. [154] propose predicting $\mathbf{v}$ instead for faster convergence, defined in terms of $\boldsymbol{\epsilon}$, $\boldsymbol{Z}^0$ and t. Thus, the model θ is optimized to learn the noise and target data distributions, with $\mathbf{v}^t$ computed as:

$$\mathbf{v}^t = \sqrt{\bar{\alpha}_t}\boldsymbol{\epsilon} - \sqrt{1 - \bar{\alpha}_t}\boldsymbol{Z}^0, \tag{7.9}$$

from which $\boldsymbol{\epsilon}$ can be computed from $\mathbf{v}^t$ and $\boldsymbol{Z}^t$ as:

$$\boldsymbol{\epsilon} = \sqrt{\bar{\alpha}_t}\mathbf{v}^t + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{Z}^t, \tag{7.10}$$

changing the model target without affecting the inference process. Therefore, the model θ is trained to predict $\mathbf{v}_\theta^t = \theta\left(\boldsymbol{Z}^t, t\right)$. To stabilize the $\mathbf{v}$-prediction training, we follow the Min-SNR-γ weighting strategy [62]. We weight the loss to balance the training at the different t diffusion steps, computing it as:

$$\mathcal{L}_{\text{diff}} = \lambda_{\text{SNR}(t)} \left\| \mathbf{v}^t - \mathbf{v}_\theta^t \right\|, \tag{7.11}$$

explicitly learning both the $\boldsymbol{\epsilon}$ and $\boldsymbol{Z}^0$ data distributions, where $\lambda_{\text{SNR}(t)}$ is the weight with respect to the signal-to-noise ratio (SNR) at the diffusion step t.

Latent diffusion inference starts from random noise as the corrupted data as $\boldsymbol{Z}^T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. From $\boldsymbol{Z}^T$ the model θ iteratively predicts $\mathbf{v}_\theta^t$ at each step t. From Eq. (7.10), the predicted noise $\boldsymbol{\epsilon}_\theta^t$ added at step t is computed from the prediction $\mathbf{v}_\theta^t$, iteratively removing noise from $\boldsymbol{Z}^T$ as:

$$\boldsymbol{Z}^{t-1} = \boldsymbol{Z}^t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta^t + \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}}\beta_t \mathcal{N}(\mathbf{0}, \boldsymbol{I}), \tag{7.12}$$

until arriving at the uncorrupted data sample $\boldsymbol{Z}^0 = \boldsymbol{Z}_\theta \sim q_\phi(\boldsymbol{Z})$ from the VAE latent space. Then, we decode the generated latent $\boldsymbol{Z}_\theta$ with the VAE decoder ψ, generating a novel semantic scene $\mathcal{P}'$ from the target scene data distribution $\psi(\boldsymbol{Z}_\theta) = \mathcal{P}' \sim p$.

Conditioned latent diffusion uses the diffusion model θ to generate samples conditioned to a target input data. In this context, we condition the semantic scene generation on an input LiDAR scan to generate a dense and semantically annotated scene. Following the classifier-free guidance [69], we train the diffusion

model θ, conditioning it to a LiDAR point cloud $\mathcal{C} = \{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_P\}$ with $\boldsymbol{c}_p \in \mathbb{R}^3$ being the points coordinates. For the conditioned training, the prediction $\mathbf{v}_\theta^t$ in Eq. (7.11) then becomes $\mathbf{v}_\theta^t = \theta(\boldsymbol{Z}^t, \tilde{\mathcal{C}}, t)$, with $\tilde{\mathcal{C}}$ having a probability ρ of being the null token $\emptyset$ or the LiDAR point cloud $\mathcal{C}$ otherwise, i.e., switching between unconditioned and conditioned generation.

7.2 Experimental Evaluation

In this section, we compare our semantic scene generation approach with prior state-of-the-art methods. We evaluate how close the scenes generated by the different methods are to the real data, showing the advantages of our method compared to previous works. Additionally, we assess the usability of the semantically annotated generated scene as training data, showing that our method generates annotated samples that can be used as semantic segmentation training data.

7.2.1 Implementation Details and Experimental Setup

Our experimental setup follows the evaluation of prior scene-scale generative methods. We generate samples from the different methods and assess the quality of the generated scenes by comparing them with real data. Additionally, we compare the use of scenes generated by our approach and prior state-of-the-art method as annotated data to train a semantic segmentation network.

Datasets. We use the SemanticKITTI dataset [5, 54] to generate our ground truth dense scenes, a 3D LiDAR semantic segmentation dataset in the context of autonomous driving. The dataset provides sequences of 3D LiDAR scans with semantic labels. To generate the dense scenes, we use an off-the-shelf SLAM [136] method to compute scan poses throughout the whole sequence. With the labeled scans and the corresponding poses, we aggregate the scan sequences with the labels, generating a voxel map of the scene with 0.1 m resolution. We also use the labels to remove moving objects from the scene to avoid aggregating them and generating large artifacts from the moving objects trajectories. During training, we query a pose from one of the individual LiDAR scans and crop the corresponding dense scene from the map at the queried pose within a fixed range at $[x, y, z]$ between $[-25.6, -25.6, -2.2]$m and $[25.6, 25.6, 4.2]$m. For the semantic segmentation experiments, we use the same dataset, sequence 00 from KITTI-360 dataset [94], and collected our own data with an Ouster LiDAR OS-1 with 128 beams. We use LiDAR scans from KITTI-360 and our collected data as conditioning to generate the semantic dense scene, and to later use the conditionally generated samples as labeled data to train a semantic segmentation model.

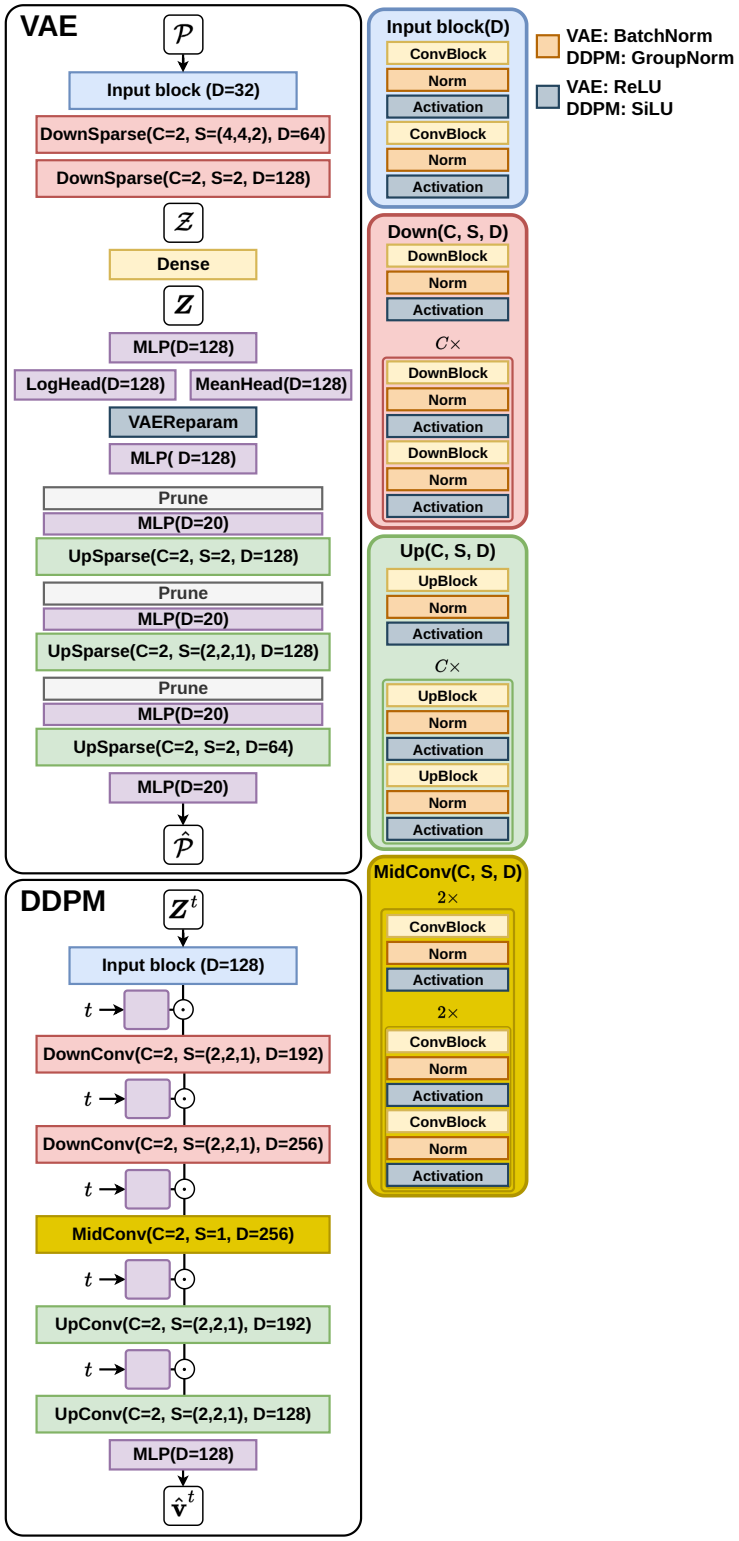


Figure 7.3: VAE and DDPM model architectures. The VAE receives the voxelized point cloud $\mathcal{P}$, encode it to the latent $\mathcal{Z}$ which is densified to $\boldsymbol{Z}$ and decoded to $\hat{\mathcal{P}}$. The DDPM receives the noisy latent $\boldsymbol{Z}^t$ at step t and predicts $\hat{\mathbf{v}}^t$ following the $\mathbf{v}$-parameterization formulation [154].

Training. We use a MinkUNet [34] as our VAE model and train our VAE model for 50 epochs, using the Adam optimizer [83] with a learning rate of 10^{-4}, multiplying it by 0.9 every 5 epochs. For the first 25 epochs, we use the weighted cross-entropy loss with the class weights provided by SemanticKITTI [5, 54]. For the last 25 epochs, we use the unweighted cross-entropy. For the losses weights we use $\lambda_1 = 1.0$, $\lambda_2 = 1.0$, $\lambda_3 = 2.0$, $\lambda_4 = 3.0$ for the individual pruning layers losses in Eq. (7.4), with $\lambda_{\text{prune}} = 1.0$, $\lambda_{\text{sem}} = 1.0$ and $\lambda_{\text{latent}} = 0.002$ in Eq. (7.7). We trained the VAE with 6 NVIDIA A40 GPUs. We use the same training scheme for the VAE refinement training.

For the DDPM, we use a regular UNet model [149] with 3D convolutions to learn the dense latent $\boldsymbol{Z}$, and we train the model for 150 epochs, using the AdamW optimizer [102] with a learning rate of $2 \cdot 10^{-4}$, multiplying it by 0.8 every 50 epochs. The DDPM is trained with $\beta_T = 0.015$ and $\beta_0 = 10^{-4}$, with $T = 1{,}000$, linearly interpolating it to get the noise factors $\beta_1, \ldots, \beta_{T-1}$ in between. For the Min-SNR-γ DDPM loss weighting, we use $\gamma = 5$. We trained the DDPM with 8 NVIDIA A40 GPUs. For the DDPM conditioned generation, we set the classifier-free guidance probability $\rho = 0.1$ and conditioning weight to 2.0. Fig. 7.3 shows the diagram with further details regarding the VAE and DDPM models.

Baselines. All baselines and our method are trained on the same set of dense scans. We compare our method with three baselines: SemCity [90], PDD [99], and XCube [145]. All baselines are trained using their official implementations and default configurations. SemCity [90] and PDD [99] are limited by the voxel resolution, with a maximum resolution of 0.2 m. For those baselines we downsample the training data to 0.2 m resolution. XCube [145] and our method can generate scenes up to 0.1 m voxel resolution. Therefore, we compare all methods at 0.2 m resolution, downsampling XCube [145] and our generated scenes, and compare our method with XCube [145] at 0.1 m resolution.

Semantic segmentation. For the semantic segmentation training, we used the pipeline described in Chapter 3 to train a MinkUNet [34]. We train the model for 15 epochs in all experiments with stochastic gradient descent optimizer with a learning rate of 0.24 and a cosine annealing learning rate scheduler [101]. For the experiments in Sec. 7.2.2, we generate 8,000 samples from each method to compute the metrics. For Sec. 7.2.3 and Sec. 7.2.4, we generate the same amount of samples as SemanticKITTI dataset [5, 54], i.e., 19,130 samples, to evaluate the performance with different subsets of real and generated samples. Also, since we remove moving objects to create the sequences maps, the occurrence of some classes decreases drastically, impacting the IoU over those classes. Therefore, in the mIoU computation, we ignore the classes that are mainly moving in the dataset, i.e., *bicycle*, *motorcycle*, *person*, *bicyclist* and *motorcyclist*.

Table 7.1: Results comparing the distribution between real and generated scenes given as maximum mean discrepancy (MMD) between synthetic data from different methods and real annotated samples from SemanticKITTI dataset.

Method	Evaluated resolution [m]	MMD ↓
PDD [99]	0.2	0.207
SemCity [90]	0.2	0.239
XCube [145]	0.2	0.160
Ours	0.2	**0.146**
XCube [145]	0.1	0.101
Ours	0.1	**0.073**

Table 7.2: Comparison between real SemanticKITTI annotated point clouds and synthetic samples from different methods given as mIoU computed over the generated semantic scenes and evaluated with the semantic segmentation model trained with real annotated data.

Method	Evaluated resolution [m]	mIoU [%] ↑
Validation set	0.2	55.59
PDD [99]	0.2	36.53
SemCity [90]	0.2	40.12
XCube [145]	0.2	40.02
Ours	0.2	**50.83**
Validation set	0.1	61.08
XCube [145]	0.1	27.24
Ours	0.1	**53.09**

7.2.2 Generated Scene Realism

As the first experiment, we want to assess the quality of the generated scenes by evaluating how close they are to the real data. To do so, we first train a semantic segmentation model following the training protocol used in Chapter 3 with the real semantic scenes created through the scans' aggregation from SemanticKITTI dataset [5, 54]. Then, we input both real and unconditionally generated scenes to the trained semantic segmentation network to compute their latent features, evaluating them with the maximum mean discrepancy (MMD). Also, we follow the evaluation by Liu et al. [99], using the trained semantic segmentation model to compute the mIoU over the generated scenes comparing it with the SemanticKITTI [5, 54] validation set results. Given that the model was trained

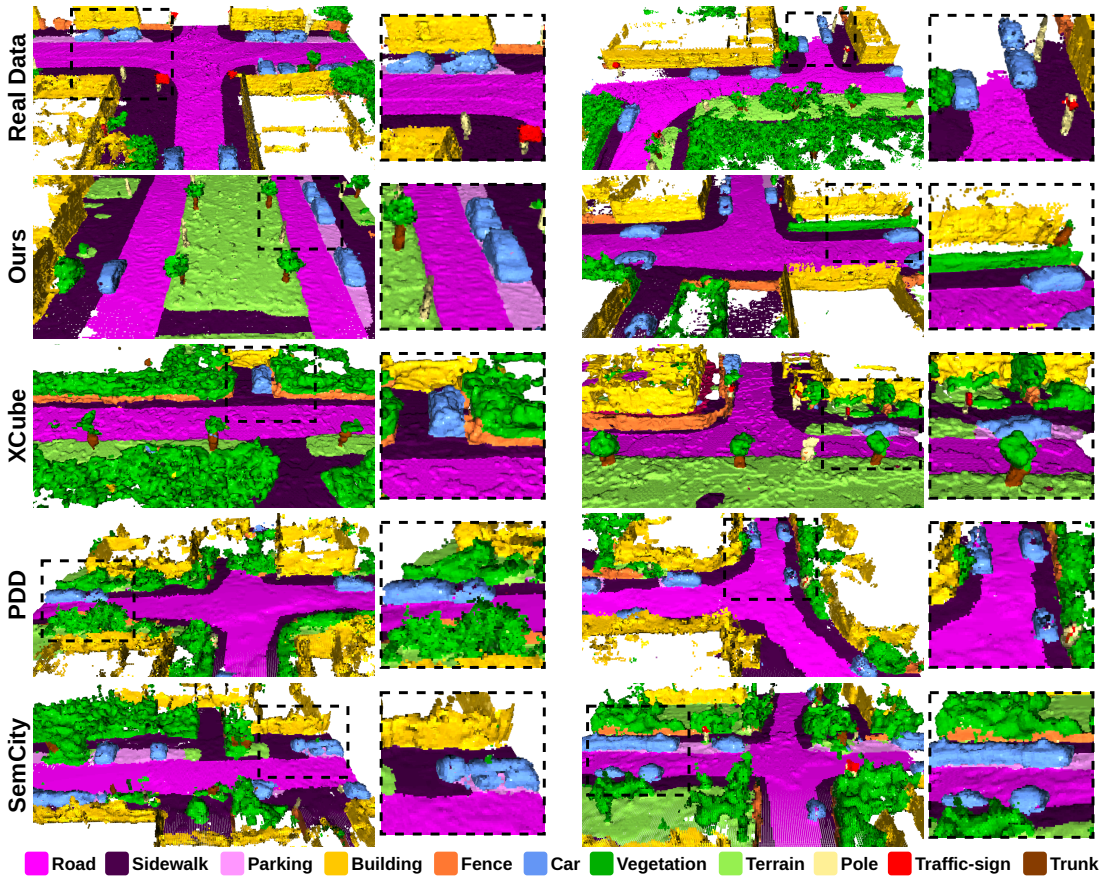


Figure 7.4: Comparison of real and unconditioned generated scenes from different methods. PDD [99] and SemCity [90] scenes are limited to 0.2 m resolution. The baselines scenes present rounder and unrealistically smooth shapes. Our method can generate more fine-grained details, closer to real data.

on real data, we expect similar mIoU over the real and generated scenes if the generated scenes are similar to the real data. We do both evaluations at two resolutions, 0.1 m, and 0.2 m, since PDD [99] and SemCity [90] can only generate scenes with resolution up to 0.2 m.

Tab. 7.1 shows the MMD metric between the generated and real scenes from SemanticKITTI dataset [5, 54]. At 0.2 m resolution, our method and XCube [145] surpass previous methods, showing the advantages of using latent diffusion and operating directly on the 3D data. At 0.1 m resolution, XCube [145] and our method performance improves, still our method achieves the best performance. Tab. 7.2 depicts the results when computing the mIoU over the generated data with the semantic segmentation model trained on SemanticKITTI dataset [5, 54]. At lower resolution, the baselines' performance are similar while our method outperforms them, achieving a mIoU close to the real data performance. At higher resolution, XCube [145] performance drops, while our method's performance increases compared to the performance at 0.2 m resolution, showing that

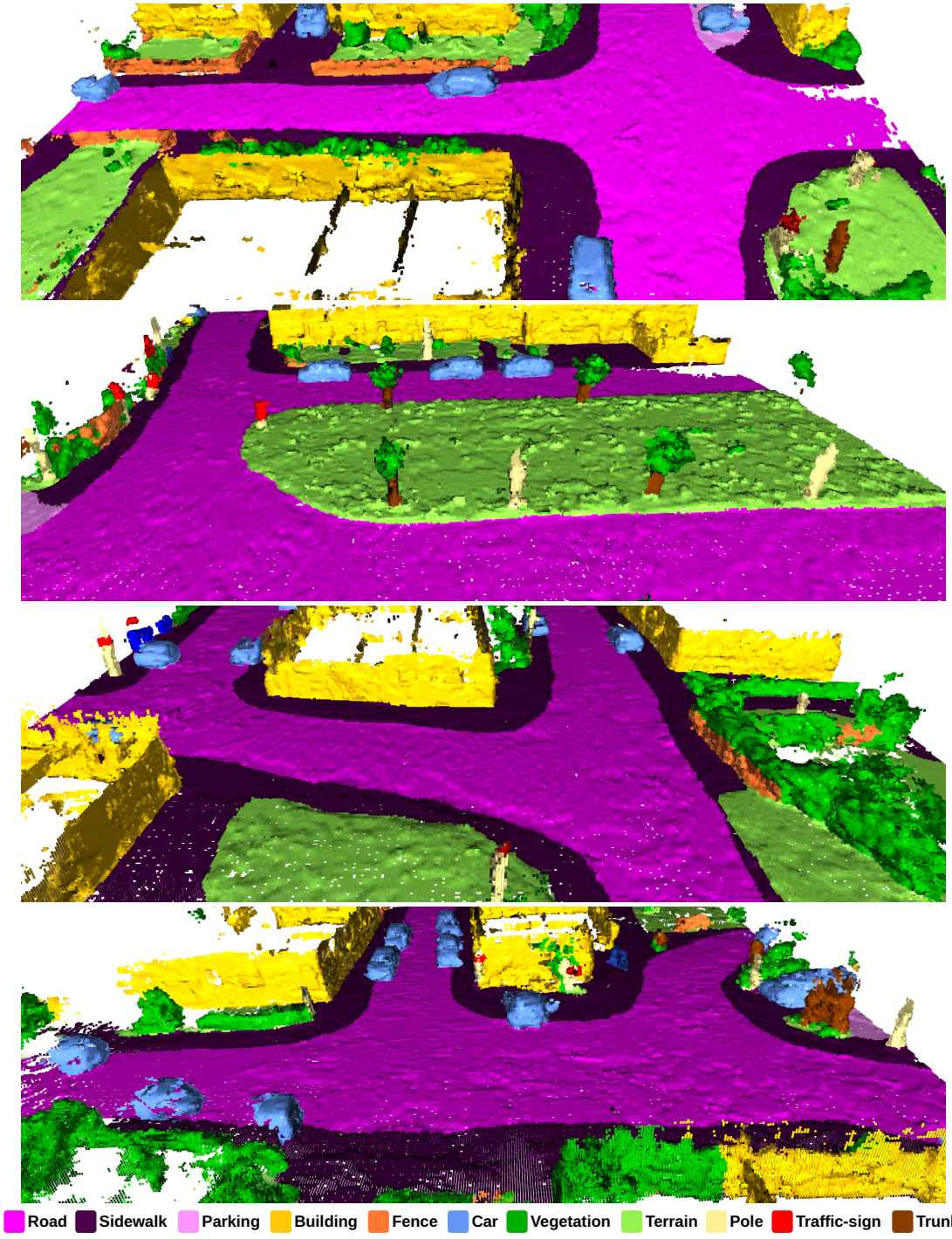


Figure 7.5: Unconditional scenes generated by our method.

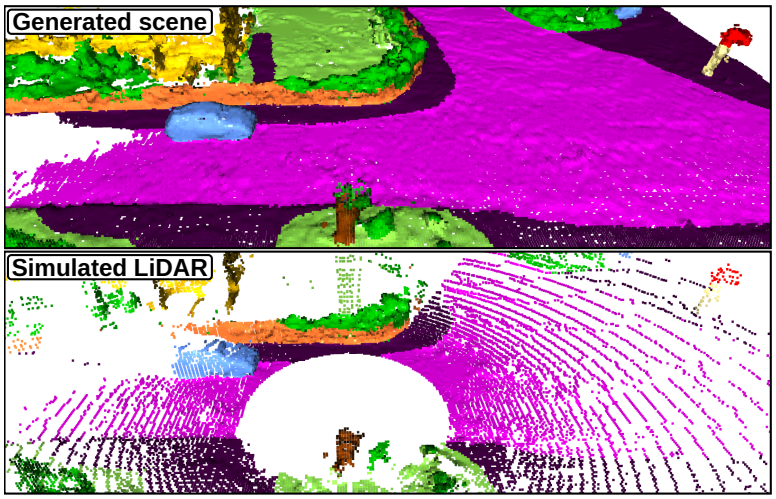


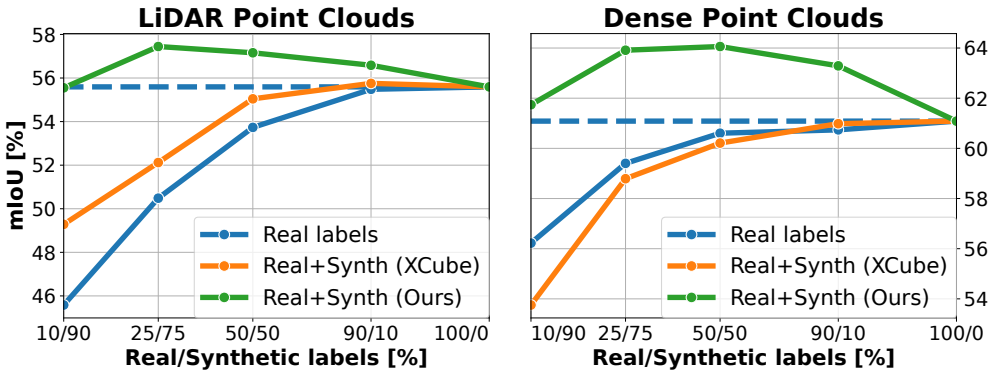
Figure 7.6: Simulated LiDAR point clouds from a dense generated scene.

Table 7.3: Semantic segmentation performance for varying percentages of real data used to train the network, and complementing it with synthetic data to achieve 100% size of the full training set trained with the LiDAR scans simulated from the densely generated scenes.

	mIoU [%] ↑				
	Real/Synth. [%]				
Synth. source	10/90	25/75	50/50	90/10	100/0
Real only	45.58	50.48	53.73	55.48	**55.59**
XCube [145]	49.28	52.12	55.04	55.75	**55.59**
Ours	**55.54**	**57.44**	**57.15**	**56.86**	**55.59**

our method is able to generate scenes which are closer to the real data compared to the baselines.

Fig. 7.4 shows scene examples from the real data compared with random scenes generated by our method and the baselines, where SemCity [90] and PDD [99] generation is limited to 0.2 m resolution. In those examples, all baselines are able to learn the overall structure in the scene and generate reasonable scenarios. However, while the baselines generate smoother and rounder shapes, our approach is able to generated more fine-grained details closer to the real data. By avoiding coarse intermediary representations and decoupled trained VAEs, our model can generate scenes with more details. Fig. 7.5 shows further unconditional generated samples from our method. These results show that our method generates closer-to-real scenes compared to the baselines.



(a) (b)

Figure 7.7: Semantic segmentation model performance trained with the different percentages of real data complemented with synthetic data from our model and XCube [145].

7.2.3 Generated Labels as Training Data

In this section, we aim at assessing the usability of the generated scenes as training data for downstream perception tasks. We define subsets of training data varying the percentages of real data from SemanticKITTI dataset [5, 54] available for training. For each subset, we complement the real data with synthetic data such that the total amount of samples is the same as the full real data training set, i.e., 100% of data. Then, we train the semantic segmentation network with both, the subsets of real data without any synthetic data, and the subsets complemented with generated data. We compare the results with generated data from our method and XCube [145], the only two methods able to generate scenes of 0.1 m resolution, computing the mIoU over the SemanticKITTI [5, 54] validation set. In addition to evaluating it over the dense scenes, we also aim at evaluating the impact using sparse LiDAR data. Therefore, we simulate a LiDAR point cloud from each dense scene from the real and generated scenes, as shown in Fig. 7.6, and replicate the experiments with the simulated LiDAR scans.

Tab. 7.3 shows the results of the semantic segmentation network trained with the simulated LiDAR scans with varying percentages of real data complemented with generated data. As shown, the use of synthetic data improves the network performance whenever using scenes generated by XCube [145] or by our method, converging to the same performance as the amount of real data gets closer to 100%. Still, the network trained with our synthetic data overall outperforms the network trained with the scenes generated by this baseline, as seen also in Fig. 7.7.a. For dense point clouds in Tab. 7.4, the network trained with synthetic data generated by XCube [145] achieves worse performance than the network trained only with real data. For such dense point clouds, the differ-

Table 7.4: Semantic segmentation performance for varying percentages of real data used to train the network, and complementing it with synthetic data to achieve 100% size of the full training set we train and evaluate with the densely generated scenes.

	mIoU [%] ↑				
	Real/Synth. [%]				
Synth. source	10/90	25/75	50/50	90/10	100/0
Real only	56.22	59.40	60.60	60.73	**61.08**
XCube [145]	53.75	58.79	60.20	60.98	**61.08**
Ours	**61.73**	**63.91**	**64.06**	**63.28**	**61.08**

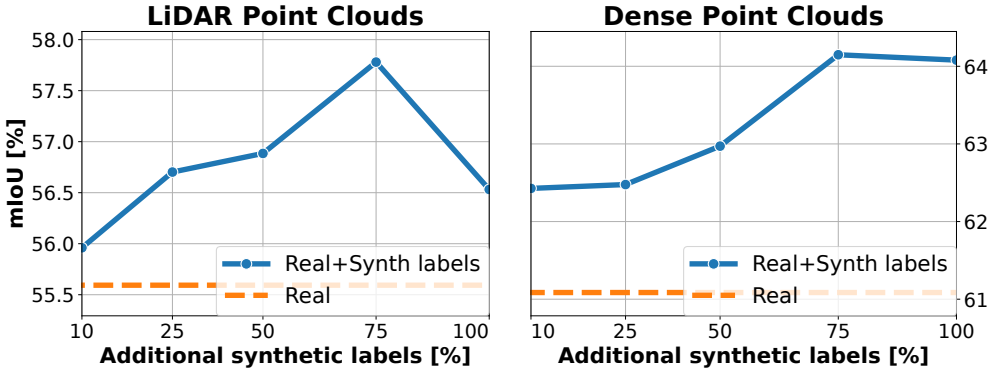
ences between the real and generated scene distributions have a higher impact on the network performance due to the amount of points in each generated sample. These results highlight the gap between the scenes generated by XCube [145] and the real scenes. In contrast, when training with a subset of real data and with our generated scenes, the model achieved better performance than the network trained only with the full real training set. At first, this may sound counterintuitive due to the results presented previously in Tab. 7.2, where our generated scenes presented a performance gap compared to the real data even though performing better than the baselines. Still, such improvement can be explained by the variability added to the training set when using generated data. The real scenes are collected sequentially, with few changes between consecutive point clouds, while the randomly generated scenes will be different from each other, adding variability to the training set. This increase in variability improves the network performance as long as enough real data is also seen by the model during training, as shown in Fig. 7.7b. These results show that our method can generate scenes better suited to be used for downstream tasks compared to the baseline, and also show the potential of using generated samples as training data.

7.2.4 Synthetic Training Set Extension

In this section, we extend the experiment done in Sec. 7.2.3. Given the real data training set, we would not expect to reduce the amount of real data available but rather enlarge this dataset to achieve even better performance in a downstream task. Therefore, in this experiment, we use the scenes generated by our method to enlarge the real data training set, evaluating the impact on the semantic segmentation model performance. We train the model with all available real training data while progressively adding synthetic data and evaluating the performance on the SemanticKITTI [5, 54] validation set. Given the results from previous sections, in this section we only evaluate the use of our generated data, performing

Table 7.5: Semantic segmentation results when training the model with the full real data training set and adding additional synthetic data, both with dense point clouds and LiDAR scans simulated from the dense point clouds.

Additional synth. data [%]	mIoU [%] ↑	
	Dense	LiDAR
0	61.08	55.59
10	62.42	55.96
25	62.47	56.70
50	62.97	56.88
75	**64.14**	**57.77**
100	64.07	56.53



(a) (b)

Figure 7.8: Semantic segmentation model performance when trained with the full real training set and adding different amount of additional synthetic data from our model.

the experiments over the dense and the simulated LiDAR point clouds.

Tab. 7.5 shows the mIoU from the models trained with the full real training set and adding different percentages of the synthetic scenes as additional data for both dense scenes and the LiDAR simulated point clouds. As shown, by using the synthetic scenes generated by our model, we were able to improve the semantic segmentation model for both dense and LiDAR point clouds. As the amount of synthetic data increases, the model performance also increases, saturating at 75% additional synthetic labels, as seen in Fig. 7.8a and Fig. 7.8b. When training with 100% of the generated data, the model performance starts to degrade, although still achieving better performance than the model trained only with real data. This can be explained by the fact that, even though better than previous state-of-the-methods, our generated samples still have differences compared to the real data, as shown in Tab. 7.2. Therefore, by increasing the

Table 7.6: Dense scenes class-wise IoU evaluated on the real data validation set comparing the network trained only with the full real training set with the training with the full training set with additional 75% synthetic data generated with our method. OV refers to other-vehicle.

	IoU [%] ↑												
	car	truck	OV	road	park.	sidewalk	build.	fence	veg.	trunk	terrain	pole	sign
Real only	94.25	48.60	23.99	90.36	33.34	67.83	86.74	37.29	85.56	**44.29**	65.77	**56.27**	**43.84**
Real + 75% Synth.	**94.67**	**59.81**	**28.45**	**91.28**	**42.02**	**70.14**	**87.59**	**42.36**	**86.14**	40.57	**67.38**	**56.27**	42.17

Table 7.7: LiDAR simulated data class-wise IoU evaluated on the real data validation set comparing the network trained only with the full real training set with the training with the full training set with additional 75% synthetic data generated with our method. OV refers to other-vehicle.

	IoU [%] ↑												
	car	truck	OV	road	park.	sidewalk	build.	fence	veg.	trunk	terrain	pole	sign
Real only	93.77	42.24	**49.00**	92.16	46.52	69.24	89.28	41.08	88.07	51.88	65.81	61.12	**58.36**
Real + 75% Synth.	**94.01**	**70.49**	48.27	**92.59**	**50.04**	**73.11**	**90.00**	**44.30**	**88.29**	**52.26**	**66.96**	**62.40**	57.51

Table 7.8: Number of point clouds in sequence 00 from KITTI-360 and in our collected dataset and number of conditional generated scenes selected during the curation process.

Source	Number of point clouds	Number of curated scenes
KITTI-360 [94]	11,518	3,939
Our data	5,276	843
Total	16,794	4,782

Table 7.9: Semantic segmentation trained with all the real SemanticKITTI training set together with unconditional randomly generated scenes and with curated conditional generated scenes. † refers to the curated conditional generated scenes.

Data source		mIoU [%] ↑	
Real [%]	Synth. [%]	Dense	LiDAR
100	0	61.08	55.59
100	25	62.47	56.70
100	75	64.14	57.77
100	25†	**64.47**	**58.11**

amount of synthetic data, the semantic segmentation model starts to learn more from the synthetic data distribution, and consequently decreasing its performance on the real data.

Tab. 7.6 and Tab. 7.7 show the class IoU when training only with the full real training set compared adding 75% synthetic data to the real training set, which achieved the best performance. In both cases, for LiDAR scans and for dense point clouds, the training with the synthetic data improves the performance in almost all the classes. These results show the potential of synthetic data in helping to improve the performance in the target downstream task. Still, due to a distribution gap between real and synthetic data discussed in Sec. 7.2.2, there is space for improvement.

7.2.5 Conditional DDPM for Data Annotation

Besides extending the training set with randomly generated samples as evaluated in Sec. 7.2.4, one useful application of DDPMs is to use conditional generation to annotate data. Target scenes could be recorded with a LiDAR sensor and the annotated scene could be generated conditioned to the collected point clouds. In this case, the exhaustive manual data annotation can be replaced

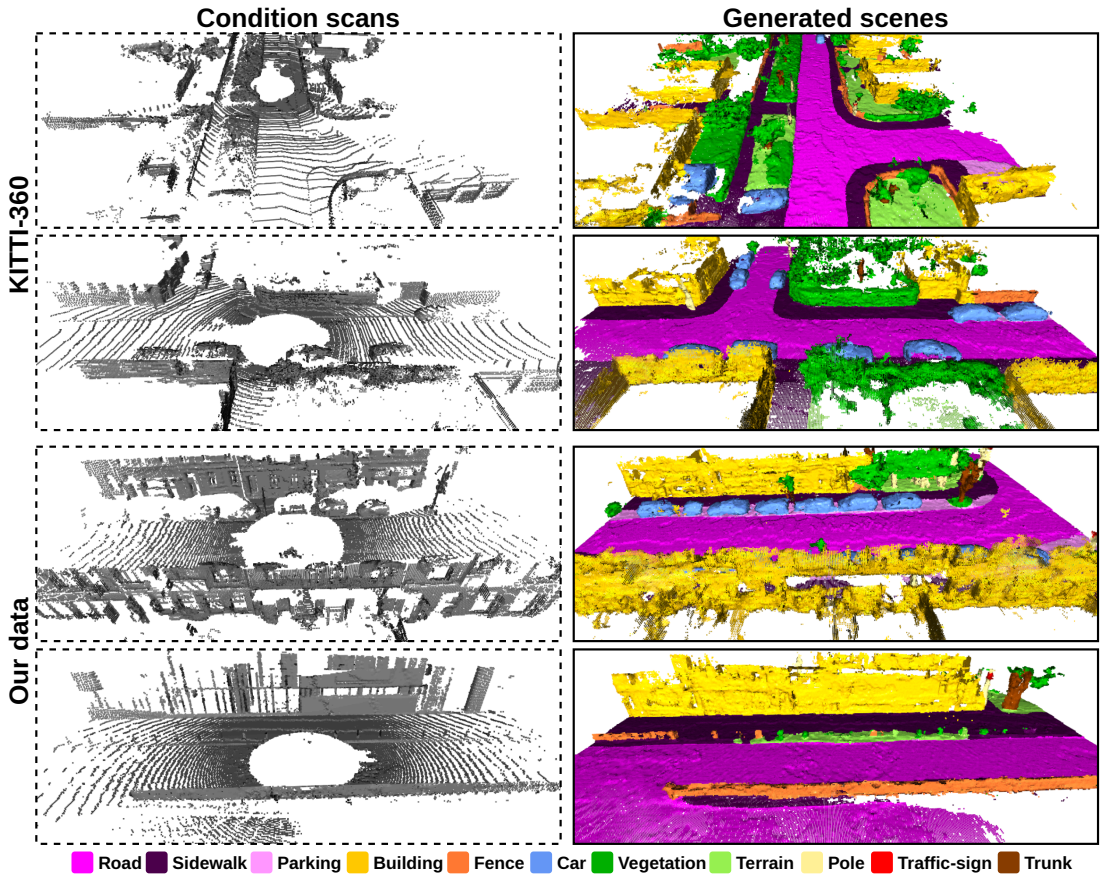


Figure 7.9: Generated scenes from our approach conditioned to KITTI-360 [94] scans and to our data collected with an Ouster LiDAR.

with a simpler data curation, selecting only the most realistic scenes generated by the DDPM. Therefore, in this section, we train the DDPM with the dense SemanticKITTI [5, 54] scenes conditioned to their corresponding LiDAR scans from the dataset. With this conditional DDPM, we generate novel scenes conditioned to sequence 00 from KITTI-360 [94] and to our data. Examples of conditional generated scenes are shown in Fig. 7.9. Next, we curate those conditional generated scenes by manually selecting scenes that look more realistic until arriving at a total of 4,782 point clouds, corresponding to approximately 25% of the size of the SemanticKITTI [5, 54] training set. In Tab. 7.8, we provide information regarding the amount of scenes selected during the curation process. Finally, we train the semantic segmentation model with the SemanticKITTI [5, 54] training set together with the curated generated scenes to evaluate the use of conditioned DDPM as data annotator. Similar to prior experiments, we repeat this evaluation for both dense scenes and their corresponding simulated LiDAR scans.

Tab. 7.9 shows the performance of the semantic segmentation network trained only with real data and with the 25% additional synthetic data from unconditional and conditional curated generated scenes, also comparing with the best-

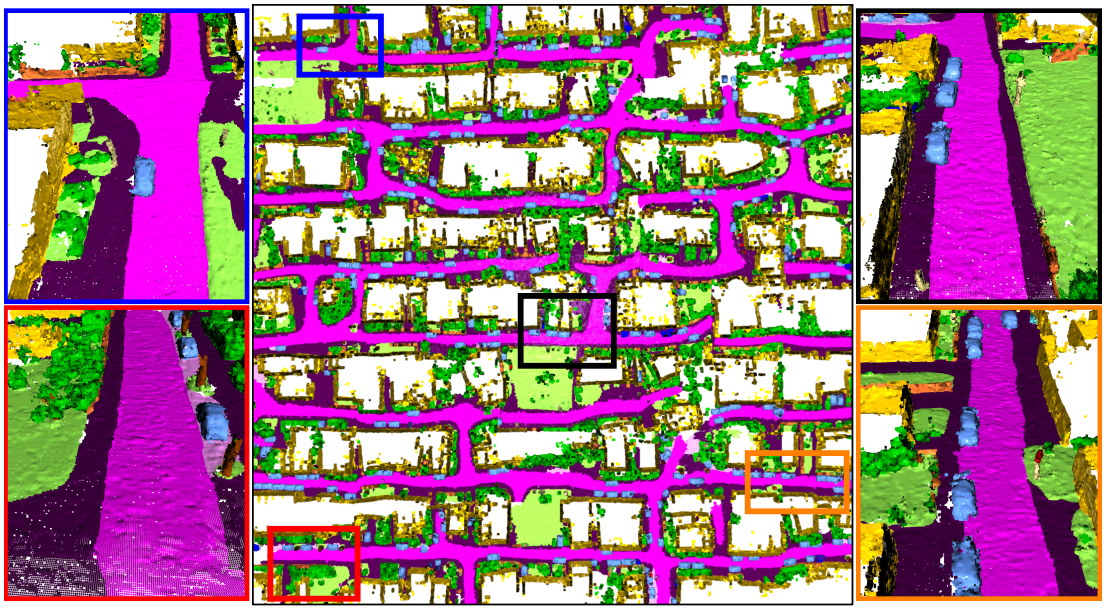


Figure 7.10: Unconditional generated large scenarios of arbitrary size generated through an incremental outpainting strategy.

performing model from Tab. 7.5 trained with additional 75% synthetic data. In this table, we notice that the network trained with the curated data achieves better performance than both uncurated training sets. Even though using fewer samples, the model trained with only 25% curated generated data achieves better performance than the model trained with additional 75% randomly generated data. These results show that the data curation process further improves the impact of the generated scenes on the network performance. This improvement shows the potential of using conditioned DDPMs as data annotator, where the exhaustive manual data annotation could be replaced by a simpler data curation process. Enabling the generation of annotated data conditioned to point clouds collected with different LiDAR sensors. This application could generate training data from specific scenarios, aiding the data annotation scalability by ignoring sensor-specific characteristics in the point cloud used as condition.

7.2.6 Generating Large-Scale Scenarios

Apart from generating independent 3D point clouds with semantic annotation, we follow recent outpainting diffusion strategies [90, 105] to generate continuous scenes with arbitrary sizes. We follow the diffusion outpainting strategy proposed by Lugmayr et al. [105], which allows for generating samples conditioned on a prior generated sample without retraining the model. We define a grid of arbitrary size, where each cell is a scene to be generated by our method. Then, this grid is incrementally generated, starting from an unconditional scene, and subsequently

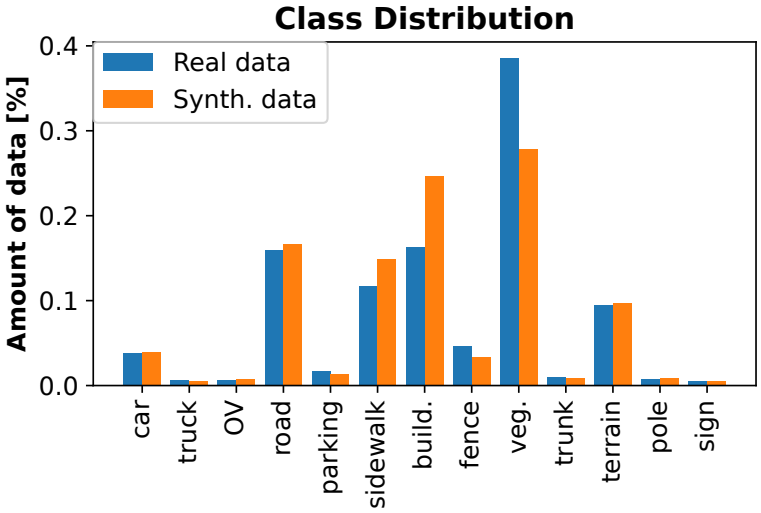


Figure 7.11: Class distribution over the real dataset compared with the synthetic data class distribution. OV refers to other-vehicle class.

generating the remaining cells in the grid conditioned on the neighboring cells already generated. Fig. 7.10 shows a city-scale scene generated by our method with the outpainting strategy, with a size of 8×8, i.e., 64 scenes generated by our method representing a single large-scale scene. As seen, the generated scenario maintains continuity through the individual generated cells, enabling the generation of large-scale city-like scenarios with continuous and consistent structures. Such a large generated scene also allows for simulating and generating training data of long and continuous trajectories with their corresponding semantic labels.

7.2.7 Generated and Real Data Gaps

In previous sections, we performed experiments to investigate the potential of generated data to augment the real training set, improving the performance of the models in the target downstream task. In this section, we aim at evaluating the opposite, comparing the data generated by our method with real data to identify the gaps between both. For that, we first compute the class distributions from the real and generated data. Then, we use the semantic segmentation network trained on real data to compute the IoU for each class in the generated and real scenes to identify the current gaps between synthetic and real data.

Fig. 7.11 shows the class distribution of SemanticKITTI [5, 54] training set and of the scenes generated by our method which was trained with SemanticKITTI [5, 54]. As seen, the generated scenes class distribution follows the overall distribution present in the dataset used to train the DDPM. Even though the generated scenes may increase the variability of the data compared to the real dataset, as discussed in Sec. 7.2.3, the ratio of data per class remains close to the train-

Table 7.10: Class-wise IoU evaluated on real data validation set and synthetic data generated by our method with semantic segmentation model trained on real data, and the difference in performance between both real and synthetic data evaluation.

	IoU [%] ↑												
	car	truck	OV	road	park.	sidewalk	build.	fence	veg.	trunk	terrain	pole	sign
Validation scenes	93.77	42.24	49.00	92.16	46.52	69.24	89.28	41.08	88.07	51.88	65.81	61.12	58.36
Generated scenes	92.29	10.42	24.10	92.48	33.74	77.34	89.75	47.73	81.81	42.99	61.72	43.55	28.79
Performance gap	-1.48	-31.82	-24.90	+0.32	-12.78	+1.64	+0.47	+6.65	-6.26	-8.89	-8.89	-17.57	-29.57

ing data distribution. Tab. 7.10 shows the IoU per class of the network trained with SemanticKITTI [5, 54] training set and with the synthetic scenes generated by our approach, evaluated with the validation set. From Fig. 7.11 and Tab. 7.10, we notice that the classes with high frequency in the synthetic scenes in Fig. 7.11, e.g., road, sidewalk, fence, are also the classes that achieve higher IoU in Tab. 7.10. For those classes, the performance is comparable or even better compared to the evaluation on the real data validation set. At the same time, the less occurring classes, e.g., truck, other-vehicle (OV), pole, traffic-sign, are the classes that perform worse in Tab. 7.10, and for which there is a bigger performance gap compared with the evaluation on the real validation set. The main gap between the generated and real data distributions therefore comes from those less occurring classes. This analysis shows a direct relation between the ratio of data samples per class and its generation quality. Hence, the DDPM training would have to be balanced to account for the semantic classes' imbalance present in scene-scale data to bridge this distribution gap in those underrepresented classes. Even though we account for this class imbalance in the VAE training, during the DDPM training this is not straightforward since the model is trained over the VAE latent, where there is no direct map between the semantic classes and the latent features. Therefore, this class imbalance is not addressed in this work. Future research can address this class imbalance in the DDPM training, which would improve the generation quality of underrepresented classes, increasing the generated scene quality.

7.3 Related Work

Many recent methods have employed latent DDPMs to enable 3D large-scale semantic scene conditional (for scene completion) and unconditional data generation. Despite also providing semantic information, none of the methods have evaluated the use of such synthetic scenes as training data to improve the performance in perception tasks. In this chapter, we present our method to achieve a more realistic scene generation and the impact of using the generated data to train perception models. In this section, we discuss the related works on 3D latent DDPMs for semantic scene-scale data generation.

Latent diffusion models have been studied to enable a more efficient training of DDPMs [138, 148, 215, 216]. Given the computational cost of computing all denoising steps during the diffusion data generation, Rombach et al. [148] propose latent diffusion to allow a more efficient computation. In this case, instead of training the DDPM over the image data, first a VAE is trained to learn a latent representation from the image data. Then, the DDPM is trained to generate novel samples from the VAE latent space, which the VAE can then

decode to novel image samples. With latent diffusion, the DDPM only has to learn a compressed latent space that represents the image data. This simplifies the generation process since the model has only to generate a small latent vector compared to generating the complete image.

Latent diffusion models for 3D data have also been studied to enable more efficient 3D data generation. Most of the methods in the field focus on the generation of single objects [156, 157, 198, 207, 208]. These methods follow the latent diffusion model first proposed in the image domain [10, 148], where a VAE is first trained to encode the point cloud into a latent space. The generation of 3D data imposes a more complex challenge compared to the image domain due to the higher dimensionality of the data and the sparsity inherent to 3D point clouds. Compared to standard DDPMs, latent diffusion approaches provide a more efficient solution, allowing for a decrease in the computational cost of generating novel 3D point cloud samples. Thus, in this chapter we employ a latent diffusion formulation distinct from the point diffusion scheme leveraged in Chapter 6.

Semantic scene-scale diffusion models have been studied recently [89, 90, 99, 119, 145]. Such methods generate large-scale 3D scenes with semantic labels. Lee et al. [90] tackle this problem by projecting the target semantic 3D data into a triplane image representation. DDPM image models are then used to generate new triplane samples, which are then unprojected to the corresponding 3D scene. However, this data projection loses information, limiting the details of the generated scene. Ren et al. [145] achieve scene-scale data generation using a hierarchical approach to model the coarse-to-fine nature of the scene. The generation process is formulated as a conditional distribution between different scene resolutions, training independently a VAE and a DDPM for each pre-defined resolution. However, this formulation can lead to errors during the generation process since the finer scene generation is unable to recover from mistakes made at coarser stages. Distinctly, Lee et al. [89] propose using a discrete diffusion formulation to generate semantic scene-scale 3D data. The scene is represented as a fixed 3D grid, and the DDPM is trained to generate the semantic labels for each cell in the grid, including an *unoccupied* class. Liu et al. [99] extend this formulation by also leveraging a hierarchical multi-resolution approach, enabling a more detailed generation. Yet, such a discrete formulation limits the scene resolution due to memory constraints since the diffusion process is computed for the whole 3D grid, even though most of the cells are empty. Such scene generation methods can be extended to achieve semantic scene completion by training the DDPM conditioned to a LiDAR point cloud, enabling the generation of a complete scene conditioned to a LiDAR scan [145]. In contrast to previous works, in this chapter we propose a 3D latent diffusion model, which does not rely on data

projections or decoupled multi-resolution models but uses a single sparse 3D VAE model to learn the target data distribution. By using a single sparse 3D VAE, we train the model with the target data while avoiding the increasing memory usage. Also, we supervise the model to prune the scene within each decoder upsampling layers during training, learning the hierarchical scene structure with a single model, and achieving more realistic scene generation.

Reducing labeling effort has been the focus of many recent works, given the challenge of scaling data annotation. To alleviate this problem, some works have propose using simulated training examples as labeled data [42, 150, 187]. However, such simulated data come with a domain gap compared to the real-world data [20, 189, 190, 212], often impeding the large-scale use of simulated training data. As also discussed in Chapters 3 and 4, other works try to alleviate this problem by proposing self-supervised pre-training methods [18, 21, 23, 64, 128, 132, 196, 213] to learn a representation from unlabeled data, which allows for later fine-tuning the pre-trained model to the target downstream tasks. More recently, due to the realistic data synthesis achieved by generative methods, some works have studied the use of generated samples as training data in the image domain [55, 158, 170, 171]. Especially in the 3D data domain, the reduction of annotation requirements can largely impact the field, given the more complex task of data annotation in 3D compared to 2D images. Previous semantic scene-scale diffusion models have motivated their work by arguing about the capability of generating semantically annotated scenes on demand [89, 90, 99, 145]. However, none of the prior works have studied the impact of using the generated semantic scenes as training data. Besides achieving more realistic scene generation, in this chapter, we evaluate the use of generated semantic scenes as training data and show their potential to alleviate the burden of data annotation.

In summary, we propose a semantic scene-scale diffusion method that can generate high-resolution scenes without intermediary projections and using a single VAE model. We train a 3D sparse VAE to learn the target scene distribution while supervising it to prune unoccupied voxels at each decoder upsampling layer. By doing so, the network learns to model the hierarchical nature of the scene with a single-resolution VAE while avoiding the increasing memory usage. Our experiments show that our method achieves a more realistic scene generation compared to previous state-of-the-art methods by using a single sparse 3D VAE. Additionally, we assess the use of our generated scenes as training data for semantic segmentation. We show that by using our generated scenes together with real data, we improve the performance of the semantic segmentation model. Finally, we perform an experiment to identify the gaps between real and generated data, providing insights into the challenges to bridge this gap in future work.

7.4 Conclusion

In this chapter, we propose a denoising diffusion probabilistic model that achieves state-of-the-art semantic scene generation without relying on image projections or multi-resolution decoupled autoencoder training. Our approach enables the generation of realistic scenes with semantic annotations that can be used as training data. This semantically annotated scene generation allows for scaling the amount of trained data without requiring manual data annotation, helping to decouple the development of perception systems from the data annotation process. We follow recent latent diffusion methods and mask-based 3D segmentation approaches, and train the model to learn the scenes' coarse-to-fine data nature within a single VAE decoder upsampling layers. Our method achieves closer-to-real scene generation compared to previous state-of-the-art methods by avoiding intermediary coarse representations and learning the scene data distribution at the original resolution. Additionally, we conducted several experiments to evaluate the use of both unconditional and conditionally generated scenes as training data for the semantic segmentation task. In our experiments, we demonstrated that combining the semantic scenes generated by our method with real data improves the model performance in the semantic segmentation task. Our results show the potential of using DDPMs to enlarge the training set from unconditional random samples and LiDAR-conditioned generated scenes.

In our experiments, we compared our method with previous works, evaluating how closely the generated samples resemble the real data samples. We also used the generated samples to train a semantic segmentation network to assess the potential of using generated data as training samples and assess its impact on real-world perception tasks. We evaluated both generated samples, conditioned on LiDAR scans, and unconditional samples randomly generated without requiring any input data. The results showed that these synthetic samples are a helpful source of data by generating realistic random scenes, with higher variability than the sequentially collected real-world data. Additionally, the semantic scene completion pipeline with the conditional DDPM serves as a powerful annotation tool, where an easier data curation process can replace exhaustive manual labeling.

Although outperforming prior works, there is still room for improvement over the generated scenes, as shown in the experiments. To assess this, we evaluated the relationship between the semantic classes distribution between real and generated scenes and the generation quality of each class. We demonstrate that the class imbalance inherent in the dataset directly affects the classes' generation quality. From our analysis, addressing this class imbalance during DDPM training could help bridge the gap between the distributions of real and generated scenes. More realistic scenes could be achieved by addressing this class imbal-

ance, enabling the large-scale use of generated scenes as training data, extending the already available labeled data with random scenes and scenes conditioned to target scenarios, and improving the performance on real-world perception tasks.

Chapter 8

Conclusion

Perception systems are crucial for the development and deployment of autonomous systems in the real world, from robots to autonomous vehicles. For a robot to safely interact with its surroundings, it must be able to perceive the environment and understand the scene to plan actions accordingly. The automation of tasks in highly dynamic and unstructured environments is currently hindered by the limited reliability of perception systems, which are still not sufficiently reliable for deployment in these scenarios. The performance of perception systems is often coupled with the amount of labeled data available to train the network to extract semantic information from the robot's surroundings. In this thesis, we studied and proposed novel methods to improve the performance of neural networks in different perception tasks without requiring additional data annotation, loosening the dependency between the perception system's performance and the amount of annotated data.

We divided the challenge of improving the performance of perception systems into three challenges regarding the data annotation. The first challenge regards the amount of labeled data available to train a model to achieve specific tasks. The more labeled data available, the more generalizable the trained model will be, and therefore, the more reliably it can be deployed in the real world. However, data annotation is not easily scalable since all possible scenarios, situations, and objects that can occur in the autonomous driving context have to be annotated for training learning-based methods. Especially when considering 3D LiDAR data commonly used in the autonomous driving context, this scalability is even more challenging. To address this scalability challenge, our work introduces two methods to train a neural network without requiring any annotated data. Our proposed strategy enables the training of neural networks with unlabeled data, optimizing them to learn a general representation from 3D point clouds that can later be fine-tuned for specific tasks, improving the model performance without requiring additional labeled data.

The second challenge is related to the number of defined classes during the data annotation. When annotating data, one has to define a closed set of classes to which the data can be annotated. Defining a closed set of classes to label an object in such an unstructured environment may lead to the occurrence of unexpected objects that were not labeled in the training data. Given that those objects were not seen during training, the behavior of the perception system towards those objects can be unpredictable. This sets the challenge of relying solely on closed-world perception systems in an open-world environment. We tackle this challenge by proposing a method that identifies instances of objects in a class-agnostic way. Our method represents the 3D point clouds as a graph, defining edges between points weighted by features extracted by a network trained without labels, embedding both geometric and object-level information into the graph. Then, we separate instances of objects from the remaining points by partitioning this graph based on its edge weights, instantiating all objects in the scene independently of their semantic class.

The third and last challenge arises from sensor-specific 3D point cloud characteristics, resulting in a domain gap between datasets collected with different LiDAR sensors. Such a domain gap hinders the development of perception systems, as the labeled data collected with a LiDAR sensor cannot be easily used to train a model for deployment with another LiDAR sensor. We proposed two methods that employ generative models to generate dense 3D point clouds, both conditioned on a LiDAR scan or completely novel scenes, not conditioned on any input. Such dense representation allows us to bridge the sensor domain gap inherent in LiDAR sensors by sampling sensor-specific point clouds from it. This method also addresses the first challenge related to the amount of labeled data available, as we can generate annotated 3D scenes conditioned on LiDAR scans or completely novel scenes without requiring any input. The generated annotated scenes can be used to train neural networks, along with the already available annotated data, improving the performance of perception systems.

8.1 Summary of the Key Contributions

In this thesis, we investigated and proposed methods to enhance the capabilities of perception systems without the need for scaling data annotation. Our *first contribution* is a method to train a neural network to learn a useful representation solely from unlabeled 3D LiDAR data, which can be used by different perception tasks, discussed in Chapter 3. We leverage an unsupervised ground segmentation method and a clustering algorithm to define coarse segments of objects within a point cloud. Then, the network is trained using a contrastive loss to learn an object-level representation from 3D LiDAR scans. While prior works leverage

point-level or scan-level optimization, our segment-level formulation optimizes the network to discriminate between different objects in the scene, embedding more semantic information. Our experiments demonstrate the advantage of our segment discrimination strategy compared to prior work, improving the model performance in various tasks. Our approach enabled the training of neural networks to learn object-specific information without requiring labels, learning a descriptive feature representation from raw 3D LiDAR point clouds.

The *second contribution* extends this segment-level discrimination strategy to the temporal domain in Chapter 4. The object's geometric representation drastically changes with respect to the LiDAR sensor viewpoint due to the sparsity of the collected point clouds. We exploit these geometric changes to learn a temporally consistent representation invariant to the LiDAR sensor viewpoint. We follow the same coarse segments extraction proposed in Chapter 3, however, we compute the relative transformations in a sequence of LiDAR scans, aggregating the scans and defining the segments over the aggregated point cloud. Then, we map the aggregated segments back to the individual LiDAR scans, computing the contrastive loss over segments of the same object seen at different points in time. Our method further improves performance on different downstream tasks while requiring substantially less annotated data. Both contributions give an important step towards decoupling the performance of perception systems from the data annotation scalability.

Our *third contribution* is presented in Chapter 5, where we exploit our proposed 3D representation learning method discussed in Chapter 3 to instantiate the different objects within a point cloud. Given a single LiDAR scan, we build a graph with points as nodes and edges defined between neighboring points. Then, we weight the graph edges based on the similarity between the point features computed using the network trained with our segment-level contrastive loss. This enables the graph to represent both the geometric and object-level information in the scene without any labels. Then, we achieve instance segmentation by partitioning this graph given the edge weights, separating individual objects in the scene in a class-agnostic way, tackling open-world scenarios. Additionally, we also propose an open-world LiDAR instance segmentation benchmark as our *fourth contribution* to evaluate and compare our method with the baselines and to support further research in the open-world field. Our experiments show that our method surpassed both unsupervised clustering-based and supervised models in the open-world scenario evaluation. Our proposed class-agnostic instance segmentation method enables the perception system to rely not only on the models trained with a closed set of labeled classes but also to identify objects that may not have been seen at training time, tackling the challenge of labeling all possible classes that may occur in the real world.

Chapter 6 presents our *fifth contribution*, where we employ recent state-of-the-art generative methods to generate a dense point cloud from a sparse LiDAR scan. Our method reformulates the diffusion process as a local-point denoising process. We define the scene completion task as an iterative denoising scheme, starting from a LiDAR scan corrupted with Gaussian noise added locally to each point. The diffusion model is trained to iteratively denoise the corrupted LiDAR scan, gradually moving the points from the noisy point cloud to a dense 3D point cloud that depicts the complete scene related to the input sparse LiDAR scan. Our method deviates from prior scene completion work by defining scene completion as a generative task. Our evaluation demonstrates that our generative formulation yields more detailed scene reconstruction compared to previous work. Enabling the generation of a dense and complete point cloud from a single LiDAR scan allows us to bridge the gap between data collected with different LiDARs.

Finally, as our *sixth and last contribution*, we aim to use generative models not only to generate a dense and complete scene but also to generate the corresponding semantic labels, presented in Chapter 7. We employ a VAE model first to learn a latent representation from the target annotated dataset and then train a diffusion model to generate novel samples from this learned latent representation. The generated latent is then decoded by the VAE, resulting in a novel 3D point cloud with its corresponding semantic annotation. We compare our proposed method with different baselines, showing that our generated scenes are closer to the real dataset distribution. Also, our experiments demonstrate that the generated annotated samples can be used as training data together with existing labeled datasets to train perception models, improving their performance. This approach enables the generation of novel samples as well as scenes conditioned on an input LiDAR scan, which can be used to train perception models, providing a way to scale annotated datasets without requiring manual annotation.

The methods presented in this thesis address different challenges related to the generalizability of perception systems' performance. Current perception systems must become more robust and reliable to enable the automation of tasks in unstructured and complex environments, such as in the context of autonomous driving. Often, increasing the performance of such systems is still solved by providing more annotated data. Still, labeling all possible situations and objects that may occur in the real world may be unfeasible. This thesis aims to enhance perception systems to support the development and deployment of reliable autonomous systems by leveraging discriminative and generative methods to reduce the requirements for manual data annotation. The methods proposed in this thesis enable neural networks to learn meaningful representations from unlabeled data, which can be used to achieve various perception tasks. In addition to the learned representations, this thesis also proposed the use of state-of-the-art gen-

erative methods to generate further annotated data that can be used to train and improve the performance of perception systems. Both discriminative and generative methods proposed in this thesis contribute to the development of reliable perception systems, enabling the generation of realistic 3D annotated data and learning descriptive representations from unlabeled 3D data that can be useful for achieving real-world tasks.

8.2 Future Work

The chapters in this thesis have contributed to different fields within both the robotics and computer vision communities and have pushed the state of the art forward. In addition to the key contributions discussed, the proposed methods are not limited to the specific experiments and tasks presented in the individual chapters. The different discriminative and generative methods described in this thesis can be explored and applied to other tasks and domains, setting a basis for further research in the robotics and computer vision fields.

In this thesis, we primarily focused on using the representations learned by the methods described in Chapters 3 and 4 to reduce the need for labeled data. However, such contrastive representation learning methods have been used by the computer vision community to train multimodal foundation models capable of embedding semantic information from text and image data. Similarly, the techniques described in this thesis could be extended also to account for multimodal data. Recent works have approached the multimodal 3D and text modalities relying mainly on projection between image and LiDAR data [30, 98], without effectively training a 3D data model. Our method could be extended to train a multimodal model directly between different sensor modalities commonly employed in robotics systems. This would enable the training of multimodal foundation models targeting robotics applications, bridging the advances from the computer community to the robotics field.

Furthermore, the class-agnostic instance segmentation method proposed in this thesis leverages a network trained solely with unlabeled LiDAR data. By extending the proposed representation learning method to a multimodal 3D and text foundation model, the class-agnostic instance segmentation could be extended also to predict a class for each instance, similar to the image domain [210]. The multimodal network could be used to compute point-wise features and define the objects instances in the point cloud. Then, a class for each instance could be computed using the text encoder from this multimodal model, going from class-agnostic to open-vocabulary instance segmentation. Such research direction can further decouple the data annotation scalability from the performance of perception systems.

Regarding the generative models, the methods proposed in this thesis aim to solve real-world tasks. The first method aims at achieving scene completion from a single LiDAR scan. This method could be further studied for application to more specific tasks, such as mapping, trajectory planning, and environment exploration. The second method generates large-scale 3D scenes with semantic labels. This method could be extended to not only generate semantics but also further data, such as RGB color, SDFs, and labels from other tasks. The generation of 3D points could also be replaced by Gaussian splats, enabling the creation of photorealistic scenes. Besides, the proposed method could be applied to generate 4D data. Recent works tackle this 4D generation, however, they are limited to low-resolution scenes [9].

Apart from generating labeled single point clouds, another practical application of generative models would be the generation of entire driving scenarios with consecutive point clouds. Such applications have been studied in the image domain, generating images of specific driving scenarios to assess the behavior of autonomous driving systems [100]. Our method could be extended to generate realistic scenarios with specific vehicles and agents' trajectories, with a wide range of applications, including behavior simulation and assessment, trajectory prediction, and planning.

Lastly, one bottleneck of 3D neural networks in general is the computation time. Compared to classical non-learning based approaches, learning-based methods are often time-consuming, limiting their application to real-time scenarios. Besides, the class-agnostic instance segmentation and generative methods are particularly time-consuming. In both cases, future work could investigate methods to accelerate the processing of such approaches. For the class-agnostic instance segmentation method, the graph definition and partitioning could be optimized for faster computation, or the class-agnostic instance prediction from our method could be used to train a model, enabling faster instance prediction, such as in the image domain [85]. Regarding the diffusion-based methods, recent works have explored ways to accelerate the diffusion inference process [104, 118]. Still, the current methods' inference speed is too slow for real-time applications. Research in the direction of faster diffusion inference would be beneficial for the methods proposed in this thesis, as well as for the research community in general.

In the end, the individual chapters of this thesis are not limited to particular applications, despite being evaluated on specific perception tasks. Instead, we hope this thesis will serve as a foundation for future research and development in the robotics and computer vision communities.

Bibliography

[1] M. Aygün, A. Osep, M. Weber, M. Maximov, C. Stachniss, J. Behley, and L. Leal-Taixe. 4D Panoptic LiDAR Segmentation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[2] B. Bešić, N. Gosala, D. Cattaneo, and A. Valada. Unsupervised Domain Adaptation for LiDAR Panoptic Segmentation. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2022.

[3] Y. Balaji, S. Nah, X. Huang, A. Vahdat, J. Song, K. Kreis, M. Aittala, T. Aila, S. Laine, B. Catanzaro, T. Karras, and M.Y. Liu. eDiff-I: Text-to-Image Diffusion Models with an Ensemble of Expert Denoisers. *arXiv preprint*, arXiv:2211.01324, 2022.

[4] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, J. Gall, and C. Stachniss. Towards 3D LiDAR-Based Semantic Scene Understanding of 3D Point Cloud Sequences: The SemanticKITTI Dataset. *Intl. Journal of Robotics Research (IJRR)*, 40(8–9):959–967, 2021.

[5] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[6] J. Behley and C. Stachniss. Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments. In *Proc. of Robotics: Science and Systems (RSS)*, 2018.

[7] J. Behley, V. Steinhage, and A. Cremers. Laser-based Segment Classification Using a Mixture of Bag-of-Words. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

[8] N. Behrmann, M. Fayyaz, J. Gall, and M. Noroozi. Long Short View Feature Decomposition via Contrastive Video Representation Learning. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[9] H. Bian, L. Kong, H. Xie, L. Pan, Y. Qiao, and Z. Liu. DynamicCity: Large-Scale 4D Occupancy Generation from Dynamic Scenes. In *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2025.

[10] A. Blattmann, R. Rombach, K. Oktay, J. Müller, and B. Ommer. Retrieval-Augmented Diffusion Models. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2022.

[11] I. Bogoslavskyi and C. Stachniss. Fast Range Image-Based Segmentation of Sparse 3D Laser Scans for Online Operation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2016.

[12] Y. Boykov and G. Funka-Lea. Graph Cuts and Efficient N-D Image Segmentation. *Intl. Journal of Computer Vision (IJCV)*, 70(2):109–131, 2006.

[13] C. Wei, H. Fan, S. Xie, C. Wu, A. Yuille, and C. Feichtenhofer. Masked Feature Prediction for Self-Supervised Visual Pre-Training. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[14] H. Caesar, V. Bankiti, A.H. Lang, S. Vora, V.E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuScenes: A Multimodal Dataset for Autonomous Driving. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[15] R.J. Campello, D. Moulavi, and J. Sander. Density-Based Clustering Based on Hierarchical Density Estimates. In *Proc. of the Conf. on Knowledge Discovery and Data Mining (KDD)*, 2013.

[16] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-End Object Detection with Transformers. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2020.

[17] M. Caron, I. Misra, J. Mairal, P. Goyal, P. Bojanowski, and A. Joulin. Unsupervised Learning of Visual Features by Contrasting Cluster Assignments. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

[18] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging Properties in Self-Supervised Vision Transformers. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[19] M. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays. Argoverse: 3D Tracking and

Forecasting with Rich Maps. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[20] S. Chen, H. Xu, R. Li, G. Liu, C. Fu, and S. Liu. SIRA-PCR: Sim-to-Real Adaptation for 3D Point Cloud Registration. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2023.

[21] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A Simple Framework for Contrastive Learning of Visual Representations. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2020.

[22] T. Chen, S. Kornblith, K. Swersky, M. Norouzi, and G.E. Hinton. Big Self-Supervised Models are Strong Semi-Supervised Learners. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

[23] X. Chen and K. He. Exploring Simple Siamese Representation Learning. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[24] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-View 3D Object Detection Network for Autonomous Driving. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[25] X. Chen, B. Mersch, L. Nunes, R. Marcuzzi, I. Vizzo, J. Behley, and C. Stachniss. Automatic Labeling to Generate Training Data for Online LiDAR-Based Moving Object Segmentation. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):6107–6114, 2022.

[26] X. Chen, A. Milioto, E. Palazzolo, P. Giguère, J. Behley, and C. Stachniss. SuMa++: Efficient LiDAR-based Semantic SLAM. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[27] X. Chen, H. Fan, R.B. Girshick, and K. He. Improved Baselines with Momentum Contrastive Learning. *arXiv preprint*, arXiv:2003.04297, 2020.

[28] Y. Chen, M. Nießner, and A. Dai. 4DContrast: Contrastive Learning with Dynamic Correspondences for 3D Scene Understanding. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2022.

[29] Z. Chen, T. Funkhouser, P. Hedman, and A. Tagliasacchi. MobileNeRF: Exploiting the Polygon Rasterization Pipeline for Efficient Neural Field Rendering on Mobile Architectures. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2022.

[30] Chen, R. and Liu, Y. and Kong, L. and Zhu, X. and Ma, Y. and Li, Y. and Hou, Y. and Qiao, Y. and Wang, W. CLIP2Scene: Towards Label-efficient 3D Scene Understanding by CLIP. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[31] B. Cheng, I. Misra, A.G. Schwing, A. Kirillov, and R. Girdhar. Masked-attention Mask Transformer for Universal Image Segmentation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[32] Y. Choi, M. El-Khamy, and J. Lee. Variable Rate Deep Image Compression With a Conditional Autoencoder. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[33] Y. Chong, L. Nunes, F. Magistri, X. Zhong, J. Behley, and C. Stachniss. Zero-Shot Semantic Segmentation for Robots in Agriculture. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2025.

[34] C. Choy, J. Gwak, and S. Savarese. 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[35] M. Cordts, S.M. Omran, Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[36] W. Curran, R. Pocius, and W. Smart. Autoencoders for Incremental Dimensionality Reduced Reinforcement Learning. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2017.

[37] P. Dellenbach, J. Deschaud, B. Jacquet, and F. Goulette. CT-ICP Real-Time Elastic LiDAR Odometry with Loop Closure. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2022.

[38] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[39] P. Dhariwal and A. Nichol. Diffusion Models Beat GANs on Image Synthesis. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2021.

[40] S. Ding, M. Li, T. Yang, R. Qian, H. Xu, Q. Chen, J. Wang, and H. Xiong. Motion-Aware Contrastive Video Representation Learning via Foreground-Background Merging. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[41] G. Donahue and E. Elhamifar. Learning to Predict Activity Progress by Self-Supervised Video Alignment. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[42] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An Open Urban Driving Simulator. In *Proc. of the Conf. on Robot Learning (CoRL)*, 2017.

[43] 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 *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2021.

[44] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel. On the Segmentation of 3D LIDAR Point Clouds. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2011.

[45] R. Dubé, A. Cramariuc, D. Dugas, J. Nieto, R. Siegwart, and C. Cadena. SegMap: 3D Segment Mapping Using Data-Driven Descriptors. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[46] M. Ester, H. Kriegel, J. Sander, and X.Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. of the Conf. on Knowledge Discovery and Data Mining (KDD)*, 1996.

[47] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C.R. Qi, Y. Zhou, et al. Large Scale Interactive Motion Forecasting for Autonomous Driving: The Waymo Open Motion Dataset. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[48] F. Liu, G. Lin, C.Foo. Point Discriminative Learning for Unsupervised Representation Learning on 3D Point Clouds. *arXiv preprint*, arXiv:2108.02104, 2021.

[49] H. Fan, Y. Yang, and M. Kankanhalli. Point 4D Transformer Networks for Spatio-Temporal Modeling in Point Cloud Videos. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[50] M. Fischler and R. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[51] W. Fong, R. Mohan, H. Valeria, L. Zhou, H. Caesar, O. Beijbom, and A. Valada. Panoptic nuScenes A Large-Scale Benchmark for LiDAR Panoptic Segmentation and Tracking. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2022.

[52] W. Förstner and B. Wrobel. *Photogrammetric Computer Vision – Statistics, Geometry, Orientation and Reconstruction*. Springer Verlag, 2016.

[53] C. Fu, C. Dong, C. Mertz, and J.M. Dolan. Depth Completion Via Inductive Fusion of Planar LIDAR and Monocular Camera. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[54] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[55] S. Geng, C.Y. Hsieh, V. Ramanujan, M. Wallingford, C.L. Li, P.W. Koh, and R. Krishna. The Unmet Promise of Synthetic Training Images: Using Retrieved Real Images Performs Better. *arXiv preprint*, arXiv:2406.05184, 2024.

[56] Z. Gojcic, O. Litany, A. Wieser, L.J. Guibas, and T. Birdal. Weakly Supervised Learning of Rigid 3D Scene Flow. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[57] B. Graham, M. Engelcke, and L. van der Maaten. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[58] J.B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, k. kavukcuoglu, R. Munos, and M. Valko. Bootstrap Your Own Latent - A New Approach to Self-Supervised Learning. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

[59] H. Kato and T. Harada. Image Reconstruction from Bag-of-Visual-Words. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[60] H. Wang, Q. Liu, X. Yue, J. Lasenby, and M. J. Kusner. Unsupervised Point Cloud Pre-Training via Occlusion Completion. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[61] H. Yin, A. Vahdat, J. Alvarez, A. Mallya, J. Kautz, and P. Molchanov. A-ViT: Adaptive Tokens for Efficient Vision Transformer. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[62] T. Hang, S. Gu, C. Li, J. Bao, D. Chen, H. Hu, X. Geng, and B. Guo. Efficient Diffusion Training via Min-SNR Weighting Strategy. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2023.

[63] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked Autoencoders Are Scalable Vision Learners. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[64] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[65] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[66] O.J. Hénaff, S. Koppula, J.B. Alayrac, A. van den Oord, O. Vinyals, and J. Carreira. Efficient Visual Pretraining with Contrastive Detection. *arXiv preprint*, arXiv:2103.10957, 2021.

[67] M. Himmelsbach, F.V. Hundelshausen, and H.J. Wuensche. Fast Segmentation of 3D Point Clouds for Ground Vehicles. In *Proc. of the IEEE Vehicles Symposium (IV)*, 2010.

[68] J. Ho, A. Jain, and P. Abbeel. Denoising Diffusion Probabilistic Models. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2020.

[69] J. Ho and T. Salimans. Classifier-Free Diffusion Guidance. In *Proc. of the NeurIPS Workshop on Deep Generative Models and Downstream Applications*, 2021.

[70] F. Hong, H. Zhou, X. Zhu, H. Li, and Z. Liu. LiDAR-Based Panoptic Segmentation via Dynamic Shifting Network. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[71] J. Hou, B. Graham, M. Niessner, and S. Xie. Exploring Data-Efficient 3D Scene Understanding With Contrastive Scene Contexts. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[72] P. Hu, D. Held, and D. Ramanan. Learning to Optimally Segment Point Clouds. *IEEE Robotics and Automation Letters (RA-L)*, 5(2):875–882, 2020.

[73] S. Huang, Y. Xie, S.C. Zhu, and Y. Zhu. Spatio-Temporal Self-Supervised Representation Learning for 3D Point Clouds. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[74] I. Misra and L. van der Maaten. Self-Supervised Learning of Pretext-Invariant Representations. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[75] J. Yang, S. Shi, Z. Wang, H. Li, and X. Qi. ST3D: Self-Training for Unsupervised Domain Adaptation on 3D Object Detection. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[76] J. Yin, D. Zhou, L. Zhang, J. Fang, C. Xu, J. Shen, and W. Wang. ProposalContrast: Unsupervised Pre-training for LiDAR-based 3D Object Detection. *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2022.

[77] S. Jenni and H. Jin. Time-Equivariant Contrastive Video Representation Learning. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[78] H. Jiang, T. Cheng, N. Gao, H. Zhang, T. Lin, W. Liu, and X. Wang. Symphonize 3D Semantic Scene Completion with Contextual Instance Queries. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[79] X. Kang, T. Yang, W. Ouyang, P. Ren, L. Li, and X. Xie. DDColor: Towards Photo-Realistic Image Colorization via Dual Decoders. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2023.

[80] T. Karras, M. Aittala, T. Aila, and S. Laine. Elucidating the Design Space of Diffusion-Based Generative Models. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2022.

[81] J. Kim, J. Woo, U. Shin, J. Oh, and S. Im. Density-Aware Domain Generalization for LiDAR Semantic Segmentation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2024.

[82] D. Kingma and M. Welling. An Introduction to Variational Autoencoders. *Foundations and Trends in Machine Learning*, 12(4):307–392, 2019.

[83] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. In *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2015.

[84] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár. Panoptic Segmentation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[85] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A.C. Berg, W.Y. Lo, P. Dollar, and R. Girshick. Segment Anything. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2023.

[86] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Proc. of the Conf. Neural Information Processing Systems (NIPS)*, 2012.

[87] L. Zhang and Z. Zhu. Unsupervised Feature Learning for Point Cloud Understanding by Contrasting and Clustering Using Graph Convolutional Neural Networks. In *Proc. of the Intl. Conf. on 3D Vision (3DV)*, 2019.

[88] F. Langer, A. Milioto, A. Haag, J. Behley, and C. Stachniss. Domain Transfer for Semantic Segmentation of LiDAR Data using Deep Neural Networks. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[89] J. Lee, W. Im, S. Lee, and S.E. Yoon. Diffusion Probabilistic Models for Scene-Scale 3D Categorical Data. *arXiv preprint*, arXiv:2301.00527, 2023.

[90] J. Lee, S. Lee, C. Jo, W. Im, J. Seon, and S.E. Yoon. SemCity: Semantic Scene Generation with Triplane Diffusion. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[91] P. Li, R. Zhao, Y. Shi, H. Zhao, J. Yuan, G. Zhou, and Y. Zhang. LODE Locally Conditioned Eikonal Implicit Scene Completion from Sparse LiDAR. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2023.

[92] R. Li, Y. Zhang, Z. Qiu, T. Yao, D. Liu, and T. Mei. Motion-Focused Contrastive Learning of Video Representations. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[93] Y. Li, S. Li, X. Liu, M. Gong, K. Li, C. Nuo, Z. Wang, Z. Li, T. Jiang, F. Yu, Y. Wang, H. Zhao, Z. Yu, and C. Feng. SSCBench A Large-Scale 3D Semantic Scene Completion Benchmark for Autonomous Driving. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2024.

[94] Y. Liao, J. Xie, and A. Geiger. KITTI-360: A Novel Dataset and Benchmarks for Urban Scene Understanding in 2D and 3D. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 45(3):3292–3310, 2022.

[95] H. Lim, O. Minho, and H. Myung. Patchwork: Concentric Zone-based Region-wise Ground Segmentation with Ground Likelihood Estimation Using a 3D LiDAR Sensor. *IEEE Robotics and Automation Letters (RA-L)*, 6(4):6458–6465, 2021.

[96] H. Lim, S. Hwang, and H. Myung. ERASOR: Egocentric Ratio of Pseudo Occupancy-Based Dynamic Object Removal for Static 3D Point Cloud Map Building. *IEEE Robotics and Automation Letters (RA-L)*, 6(2):2272–2279, 2021.

[97] H. Lim, L. Nunes, B. Mersch, X. Chen, J. Behley, H. Myung, and C. Stachniss. ERASOR2: Instance-Aware Robust 3D Mapping of the Static World in Dynamic Scenes. In *Proc. of Robotics: Science and Systems (RSS)*, 2023.

[98] 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. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2023.

[99] Y. Liu, X. Li, X. Li, L. Qi, C. Li, and M.H. Yang. Pyramid Diffusion for Fine 3D Large Scene Generation. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2024.

[100] 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 *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2024.

[101] I. Loshchilov and F. Hutter. SGDR: Stochastic Gradient Descent with Restarts. *arXiv preprint*, arXiv:1608.03983, 2016.

[102] I. Loshchilov and F. Hutter. Decoupled Weight Decay Regularization. In *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2019.

[103] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu. DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps.

In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2022.

[104] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu. DPM-Solver++: Fast Solver for Guided Sampling of Diffusion Probabilistic Models. *arXiv preprint*, arXiv:2211.01095, 2023.

[105] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and V. Gool. RePaint: Inpainting using Denoising Diffusion Probabilistic Models. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[106] S. Luo and W. Hu. Diffusion Probabilistic Models for 3D Point Cloud Generation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[107] Z. Lyu, Z. Kong, X. XU, L. Pan, and D. Lin. A Conditional Point Diffusion-Refinement Paradigm for 3D Point Cloud Completion. In *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2022.

[108] M. Afham, I. Dissanayake, D. Dissanayake, A. Dharmasiri, K. Thilakarathna, and R. Rodrigo. CrossPoint: Self-Supervised Cross-Modal Contrastive Learning for 3D Point Cloud Understanding. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[109] M. Zhuge, D. Gao, D. Fan, L. Jin, B. Chen, H. Zhou, M. Qiu, and L. Shao. Kaleido-BERT: Vision-Language Pre-Training on Fashion Domain. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[110] F. Ma, G.V. Cavalheiro, and S. Karaman. Self-Supervised Sparse-To-Dense Self-Supervised Depth Completion from LiDAR and Monocular Camera. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[111] F. Ma and S. Karaman. Sparse-To-Dense: Depth Prediction from Sparse Depth Samples and a Single Image. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[112] R. Marcuzzi, L. Nunes, E. Marks, L. Wiesmann, T. Läbe, J. Behley, and C. Stachniss. SfmOcc: Vision-Based 3D Semantic Occupancy Prediction in Urban Environments. *IEEE Robotics and Automation Letters (RA-L)*, 10(5):5074–5081, 2025.

[113] R. Marcuzzi, L. Nunes, L. Wiesmann, J. Behley, and C. Stachniss. Mask-Based Panoptic LiDAR Segmentation for Autonomous Driving. *IEEE Robotics and Automation Letters (RA-L)*, 8(2):1141–1148, 2023.

[114] R. Marcuzzi, L. Nunes, L. Wiesmann, E. Marks, J. Behley, and C. Stachniss. Mask4D: End-to-End Mask-Based 4D Panoptic Segmentation for LiDAR Sequences. *IEEE Robotics and Automation Letters (RA-L)*, 8(11):7487–7494, 2023.

[115] R. Marcuzzi, L. Nunes, L. Wiesmann, I. Vizzo, J. Behley, and C. Stachniss. Contrastive Instance Association for 4D Panoptic Segmentation for Sequences of 3D LiDAR Scans. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2022.

[116] R. Marcuzzi, L. Nunes, L. Wiesmann, I. Vizzo, J. Behley, and C. Stachniss. Contrastive Instance Association for 4D Panoptic Segmentation using Sequences of 3D LiDAR Scans. *IEEE Robotics and Automation Letters (RA-L)*, 7(2):1550–1557, 2022.

[117] E. Marks, L. Nunes, F. Magistri, M. Sodano, R. Marcuzzi, L. Zimmermann, J. Behley, and C. Stachniss. Tree Skeletonization from 3D Point Clouds by Denoising Diffusion. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2025.

[118] C. Meng, R. Rombach, R. Gao, D. Kingma, S. Ermon, J. Ho, and T. Salimans. On Distillation of Guided Diffusion Models. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[119] Q. Meng, L. Li, M. Nießner, and A. Dai. LT3SD: Latent Trees for 3D Scene Diffusion. *arXiv preprint*, arXiv:2409.08215, 2024.

[120] B. Mersch, X. Chen, I. Vizzo, L. Nunes, J. Behley, and C. Stachniss. Receding Moving Object Segmentation in 3D LiDAR Data Using Sparse 4D Convolutions. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):7503–7510, 2022.

[121] B. Mersch, T. Guadagnino, X. Chen, Tiziano, I. Vizzo, J. Behley, and C. Stachniss. Building Volumetric Beliefs for Dynamic Environments Exploiting Map-Based Moving Object Segmentation. *IEEE Robotics and Automation Letters (RA-L)*, 8(8):5180–5187, 2023.

[122] A. Milioto, J. Behley, C. McCool, and C. Stachniss. LiDAR Panoptic Segmentation for Autonomous Driving. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[123] D. Mitzel and B. Leibe. Taking Mobile Multi-Object Tracking to the Next Level: People, Unknown Objects, and Carried Items. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2012.

[124] F. Moosmann, O. Pink, and C. Stiller. Segmentation of 3D Lidar Data in non-flat Urban Environments using a Local Convexity Criterion. In *Proc. of the IEEE Vehicles Symposium (IV)*, 2009.

[125] K. Murphy. *Machine Learning – A Probabilistic Perspective*. MIT Press, 2012.

[126] K. Nakashima and R. Kurazume. LiDAR Data Synthesis with Denoising Diffusion Probabilistic Models. *arXiv preprint*, arXiv:2309.09256, 2023.

[127] A.Q. Nichol and P. Dhariwal. Improved Denoising Diffusion Probabilistic Models. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2021.

[128] 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 (RA-L)*, 7(2):2116–2123, 2022.

[129] L. Nunes, R. Marcuzzi, B. Mersch, J. Behley, and C. Stachniss. Scaling Diffusion Models to Real-World 3D LiDAR Scene Completion. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[130] L. Nunes, X. Chen, R. Marcuzzi, A. Osep, L. Leal-Taixé, C. Stachniss, and J. Behley. Unsupervised Class-Agnostic Instance Segmentation of 3D LiDAR Data for Autonomous Vehicles. *IEEE Robotics and Automation Letters (RA-L)*, 7(4):8713–8720, 2022.

[131] L. Nunes, R. Marcuzzi, J. Behley, and C. Stachniss. Towards Generating Realistic 3D Semantic Training Data for Autonomous Driving. *arXiv preprint*, arXiv:2503.21449, 2025.

[132] L. Nunes, L. Wiesmann, R. Marcuzzi, X. Chen, J. Behley, and C. Stachniss. Temporal Consistent 3D LiDAR Representation Learning for Semantic Perception in Autonomous Driving. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[133] O. Unal, D. Dai, and L. Van Gool. Scribble-Supervised LiDAR Semantic segmentation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[134] T. Pan, Y. Song, T. Yang, W. Jiang, and W. Liu. VideoMoCo: Contrastive Video Representation Learning With Temporally Adversarial Examples. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[135] Y. Pan, B. Gao, J. Mei, S. Geng, C. Li, and H. Zhao. SemanticPOSS: A Point Cloud Dataset with Large Quantity of Dynamic Instances. *arXiv preprint*, arXiv:2002.09147, 2020.

[136] Y. Pan, X. Zhong, L. Wiesmann, T. Posewsky, J. Behley, and C. Stachniss. PIN-SLAM: LiDAR SLAM Using a Point-Based Implicit Neural Representation for Achieving Global Map Consistency. *IEEE Trans. on Robotics (TRO)*, 40:4045–4064, 2024.

[137] T. Park, A.A. Efros, R. Zhang, and J.Y. Zhu. Contrastive Learning for Unpaired Image-to-Image Translation. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2020.

[138] W. Peebles and S. Xie. Scalable Diffusion Models with Transformers. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2023.

[139] M. Polic, I. Krajacic, N. Lepora, and M. Orsag. Convolutional Autoencoder for Feature Extraction in Tactile Sensing. *IEEE Robotics and Automation Letters (RA-L)*, 4(4):3671–3678, 2019.

[140] M. Popović, F. Thomas, S. Papatheodorou, N. Funk, T. Vidal-Calleja, and S. Leutenegger. Volumetric Occupancy Mapping With Probabilistic Depth Completion for Robotic Navigation. *IEEE Robotics and Automation Letters (RA-L)*, 6(3):5072–5079, 2021.

[141] L. Porzi, S.R. Bulo, A. Colovic, and P. Kontschieder. Seamless Scene Segmentation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[142] C.R. Qi, O. Litany, K. He, and L.J. Guibas. Deep Hough Voting for 3D Object Detection in Point Clouds. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[143] R. Qian, T. Meng, B. Gong, M. Yang, H. Wang, S. Belongie, and Y. Cui. Spatiotemporal Contrastive Video Representation Learning. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[144] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. Zero-Shot Text-to-Image Generation. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2021.

[145] X. Ren, J. Huang, X. Zeng, K. Museth, S. Fidler, and F. Williams. XCube: Large-Scale 3D Generative Modeling using Sparse Voxel Hierarchies. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[146] C. Rist, D. Emmerichs, M. Enzweiler, and D. Gavrila. Semantic Scene Completion using Local Deep Implicit Functions on LiDAR Data. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(10):7205–7218, 2021.

[147] L. Roldão, R. de Charette, and A. Verroust-Blondet. LMSCNet: Lightweight Multiscale 3D Semantic Completion. In *Proc. of the Intl. Conf. on 3D Vision (3DV)*, 2020.

[148] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-Resolution Image Synthesis With Latent Diffusion Models. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[149] O. Ronneberger, P.Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Proc. of the Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

[150] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. Lopez. The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[151] R.B. Rusu and S. Cousins. 3D is Here: Point Cloud Library (PCL). In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2011.

[152] S. Shi, Z. Wang, J. Shi, X. Wang, and H. Li. From Points to Parts: 3D Object Detection from Point Cloud with Part-aware and Part-aggregation Network. *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 43:2647–2664, 2021.

[153] S. Yun, H. Lee, J. Kim, and J. Shin. Patch-Level Representation Learning for Self-Supervised Vision Transformers. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[154] T. Salimans and J. Ho. Progressive Distillation for Fast Sampling of Diffusion Models. In *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2022.

[155] A. Sanghi. Info3D: Representation Learning on 3D Objects Using Mutual Information Maximization and Contrastive Learning. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2020.

[156] A. Sanghi, H. Chu, J.G. Lambourne, Y. Wang, C.Y. Cheng, M. Fumero, and K.R. Malekshan. CLIP-Forge: Towards Zero-Shot Text-To-Shape Generation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[157] A. Sanghi, R. Fu, V. Liu, K.D. Willis, H. Shayani, A.H. Khasahmadi, S. Sridhar, and D. Ritchie. CLIP-Sculptor: Zero-Shot Generation of High-Fidelity and Diverse Shapes From Natural Language. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[158] M.B. Sariyildiz, K. Alahari, D. Larlus, and Y. Kalantidis. Fake It Till You Make It: Learning Transferable Representations from Synthetic ImageNet Clones. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[159] A. Shaban, J. Lee, S. Jung, X. Meng, and B. Boots. LiDAR-UDA: Self-ensembling Through Time for Unsupervised LiDAR Domain Adaptation. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2023.

[160] S. Shi, X. Wang, and H. Li. PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[161] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. In *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2014.

[162] M. Sodano, F. Magistri, J. Behley, and C. Stachniss. Open-World Panoptic Segmentation. *arXiv preprint*, arXiv:2412.12740, 2024.

[163] M. Sodano, F. Magistri, L. Nunes, J. Behley, and C. Stachniss. Open-World Semantic Segmentation Including Class Similarity. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[164] J. Song, C. Meng, and S. Ermon. Denoising Diffusion Implicit Models. In *Proc. of the Intl. Conf. on Learning Representations (ICLR)*, 2021.

[165] S. Song, F. Yu, A. Zeng, A. Chang, M. Savva, and T. Funkhouser. Semantic Scene Completion from a Single Depth Image. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[166] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal on Machine Learning Research (JMLR)*, 15:1929–1958, 2014.

[167] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov. Scalability in Perception for Autonomous Driving: Waymo Open Dataset. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[168] H. Tang, Z. Liu, S. Zhao, Y. Lin, J. Lin, H. Wang, and S. Han. Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2020.

[169] A. Teichman, J. Levinson, and S. Thrun. Towards 3D Object Recognition via Classification of Arbitrary Object Tracks. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2011.

[170] Y. Tian, L. Fan, K. Chen, D. Katabi, D. Krishnan, and P. Isola. Learning Vision from Models Rivals Learning Vision from Data. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[171] Y. Tian, L. Fan, P. Isola, H. Chang, and D. Krishnan. StableRep: Synthetic Images from Text-to-Image Models Make Strong Visual Representation Learners. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2021.

[172] Z. Tong, Y. Song, J. Wang, and L. Wang. VideoMAE: Masked Autoencoders are Data-Efficient Learners for Self-Supervised Video Pre-Training. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2022.

[173] A. van den Oord, Y. Li, and O. Vinyals. Representation Learning with Contrastive Predictive Coding. *arXiv preprint*, arXiv:1807.03748, 2018.

[174] W. Van Gansbeke, S. Vandenhende, S. Georgoulis, and L. Van Gool. Unsupervised Semantic Segmentation by Contrasting Object Mask Proposals. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[175] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. In *Proc. of the Conf. Neural Information Processing Systems (NIPS)*, 2017.

[176] I. Vizzo, B. Mersch, R. Marcuzzi, L. Wiesmann, J. Behley, and C. Stachniss. Make it Dense: Self-Supervised Geometric Scan Completion of Sparse 3D LiDAR Scans in Large Outdoor Environments. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):8534–8541, 2022.

[177] I. Vizzo, B. Mersch, L. Nunes, L. Wiesmann, T. Guadagnino, and C. Stachniss. Toward Reproducible Version-Controlled Perception Platforms: Embracing Simplicity in Autonomous Vehicle Dataset Acquisition. In *Proc. of the Intl. Conf. on Intelligent Transportation Systems Workshops*, 2023.

[178] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss. KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way. *IEEE Robotics and Automation Letters (RA-L)*, 8(2):1029–1036, 2023.

[179] W. Wang, T. Zhou, F. Yu, J. Dai, E. Konukoglu, and L. Van Gool. Exploring Cross-Image Pixel Contrast for Semantic Segmentation. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[180] D.Z. Wang, I. Posner, and P. Newman. What Could Move? Finding Cars, Pedestrians and Bicyclists in 3D Laser Data. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2012.

[181] F. Wang, Q. Sun, D. Zhang, and J. Tang. Unleashing Network Potentials for Semantic Scene Completion. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[182] H. Wang, L. Yang, X. Rong, J. Feng, and Y. Tian. Self-Supervised 4D Spatio-Temporal Feature Learning via Order Prediction of Sequential Point Cloud Clips. In *Proc. of the IEEE Winter Conf. on Applications of Computer Vision (WACV)*, 2021.

[183] W. Wang, J. Dai, Z. Chen, Z. Huang, Z. Li, X. Zhu, X. Hu, T. Lu, L. Lu, H. Li, et al. InternImage: Exploring Large-Scale Vision Foundation Models with Deformable Convolutions. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[184] L. Wiesmann, A. Milioto, X. Chen, C. Stachniss, and J. Behley. Deep Compression for Dense Point Cloud Maps. *IEEE Robotics and Automation Letters (RA-L)*, 6(2):2060–2067, 2021.

[185] L. Wiesmann, L. Nunes, J. Behley, and C. Stachniss. KPPR: Exploiting Momentum Contrast for Point Cloud-Based Place Recognition. *IEEE Robotics and Automation Letters (RA-L)*, 8(2):592–599, 2023.

[186] L. Wiesmann, T. Läbe, L. Nunes, J. Behley, and C. Stachniss. Joint Intrinsic and Extrinsic Calibration of Perception Systems Utilizing a Calibration Environment. *IEEE Robotics and Automation Letters (RA-L)*, 9(10):9103–9110, 2024.

[187] J. Wilson, J. Song, Y. Fu, A. Zhang, A. Capodieci, P. Jayakumar, K. Barton, and M. Ghaffari. MotionSC: Data Set and Network for Real-Time Semantic Mapping in Dynamic Environments. *IEEE Robotics and Automation Letters (RA-L)*, 7(3):8439–8446, 2022.

[188] K. Wong, S. Wang, M. Ren, M. Liang, and R. Urtasun. Identifying Unknown Instances for Autonomous Driving. In *Proc. of the Conf. on Robot Learning (CoRL)*, 2020.

[189] B. Wu, A. Wan, X. Yue, and K. Keutzer. SqueezeSeg: Convolutional Neural Nets with Recurrent CRF for Real-Time Road-Object Segmentation from 3D LiDAR Point Cloud. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2018.

[190] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer. SqueezeSegV2: Improved Model Structure and Unsupervised Domain Adaptation for Road-Object Segmentation from a LiDAR Point Cloud. In *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2019.

[191] X. Chen, S. Xie, and K. He. An Empirical Study of Training Self-Supervised Vision Transformers. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[192] X. Wang, R. Zhang, C. Shen, T. Kong, and L. Li. Dense Contrastive Learning for Self-Supervised Visual Pre-Training. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[193] X. Yu, L. Tang, Y. Rao, T. Huang, J. Zhou, and J. Lu. Point-BERT: Pre-Training 3D Point Cloud Transformers With Masked Point Modeling. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[194] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer. Scaling Vision Transformers. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[195] Z. Xia, Y. Liu, X. Li, X. Zhu, Y. Ma, Y. Li, Y. Hou, and Y. Qiao. SCPNet: Semantic Scene Completion on Point Cloud. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[196] S. Xie, J. Gu, D. Guo, C.R. Qi, L. Guibas, and O. Litany. PointContrast: Unsupervised Pre-training for 3D Point Cloud Understanding. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2020.

[197] Y. Xiong, W.C. Ma, J. Wang, and R. Urtasun. Learning Compact Representations for LiDAR Completion and Generation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[198] J. Xu, X. Wang, W. Cheng, Y.P. Cao, Y. Shan, X. Qie, and S. Gao. Dream3D: Zero-Shot Text-to-3D Synthesis Using 3D Shape Prior and Text-to-Image Diffusion Models. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[199] Y. Xu, X. Zhu, J. Shi, G. Zhang, H. Bao, and H. Li. Depth Completion From Sparse LiDAR Data With Depth-Normal Constraints. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2019.

[200] Y. Lee, J. Kim, J. Willette, and S. J. Hwang. MPViT: Multi-Path Vision Transformer for Dense Prediction. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[201] B. Yang, M. Liang, and R. Urtasun. HDNet: Exploiting HD Maps for 3D Object Detection. In *Proc. of the Conf. on Robot Learning (CoRL)*, 2018.

[202] F. Ye and A.G. Bors. Wasserstein Expansible Variational Autoencoder for Discriminative and Generative Continual Learning. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2023.

[203] L. Yi, B. Gong, and T. Funkhouser. Complete & Label: A Domain Adaptation Approach to Semantic Segmentation of LiDAR Point Clouds. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[204] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[205] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny. Barlow Twins: Self-Supervised Learning via Redundancy Reduction. In *Proc. of the Intl. Conf. on Machine Learning (ICML)*, 2021.

[206] M.D. Zeiler and R. Fergus. Visualizing and Understanding Convolutional Networks. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2014.

[207] X. Zeng, X. Chen, Z. Qi, W. Liu, Z. Zhao, Z. Wang, B. Fu, Y. Liu, and G. Yu. Paint3D: Paint Anything 3D with Lighting-Less Texture Diffusion Models. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[208] X. Zeng, A. Vahdat, F. Williams, Z. Gojcic, O. Litany, S. Fidler, and K. Kreis. LION: Latent Point Diffusion Models for 3D Shape Generation. In *Proc. of the Conf. on Neural Information Processing Systems (NeurIPS)*, 2022.

[209] A. Zhang, Z.C. Lipton, M. Li, and A.J. Smola. Dive into Deep Learning. *arXiv preprint*, arXiv:2106.11342, 2021.

[210] H. Zhang, F. Li, X. Zou, S. Liu, C. Li, J. Gao, J. Yang, and L. Zhang. A Simple Framework for Open-Vocabulary Segmentation and Detection. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2023.

[211] L. Zhang, A. Rao, and M. Agrawala. Adding Conditional Control to Text-to-Image Diffusion Models. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2023.

[212] M. Zhang, W. Peng, G. Ding, C. Lei, C. Ji, and Q. Hao. CTS Sim-To-Real Unsupervised Domain Adaptation on 3D Detection. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2024.

[213] Z. Zhang, R. Girdhar, A. Joulin, and I. Misra. Self-Supervised Pretraining of 3D Features on any Point-Cloud. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[214] L. Zhou, Y. Du, and J. Wu. 3D Shape Generation and Completion Through Point-Voxel Diffusion. In *Proc. of the IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, 2021.

[215] Y. Zhou, B. Liu, Y. Zhu, X. Yang, C. Chen, and J. Xu. Shifted Diffusion for Text-to-Image Generation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[216] Y. Zhou, R. Zhang, C. Chen, C. Li, C. Tensmeyer, T. Yu, J. Gu, J. Xu, and T. Sun. Towards Language-Free Training for Text-to-Image Generation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[217] Z. Zhou, Y. Zhang, and H. Foroosh. Panoptic-PolarNet: Proposal-Free LiDAR Point Cloud Panoptic Segmentation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[218] X. Zhu, H. Zhou, T. Wang, F. Hong, Y. Ma, W. Li, H. Li, and D. Lin. Cylindrical and Asymmetrical 3D Convolution Networks for LiDAR Segmentation. In *Proc. of the IEEE/CVF Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[219] V. Zyrianov, X. Zhu, and S. Wang. Learning to Generate Realistic LiDAR Point Clouds. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2022.

List of Figures

1.1 Challenging scenarios in autonomous driving 2
1.2 Perception systems challenges and thesis contributions 4

2.1 2D convolution example 13
2.2 3D convolution example 16
2.3 3D sparse convolution example 17
2.4 Sparse convolutional neural network architecture diagram 18
2.5 Contrastive learning visual example 20
2.6 Variational autoencoder for image generation 23

3.1 Our pre-trained model fine-tuned with 0.1% of labels 27
3.2 Diagram of the pre-training and fine-tuning stages 28
3.3 Diagram of the point clouds segments sampling 30
3.4 Fine-tuned models qualitative comparison on SemanticKITTI . . 34
3.5 Performance of different contrastive methods on SemanticPOSS . 37

4.1 Performance improvement with our pre-training method 46
4.2 Diagram of our pre-training strategy 48
4.3 Diagram of the temporal segments extraction 49
4.4 Diagram of the temporal batch samples 50
4.5 Diagram of the implicit clustering scheme 52

5.1 Example of closed-world and open-world scenarios 68
5.2 Diagram of the proposed unsupervised instance segmentation method 70
5.3 Examples of instance proposals extract from a LiDAR scan 71
5.4 Saliency maps comparison 73
5.5 Example of foreground and background points sampling 74
5.6 Comparison of instance proposal and our method results 80
5.7 Qualitative results examples 81
5.8 Example showing the impact of wrong instance proposals 83

6.1 Example of scene completion with our method 91
6.2 Comparison between standard and our proposed local diffusion . . 94

6.3 Impact of the proposed noise prediction regularization 95
6.4 Diagram of the conditioning in each layer 97
6.5 Noise predictor and condition encoder models architecture 98
6.6 Qualitative results on KITTI-360 102
6.7 Qualitative results on KITTI 102
6.8 Noise regularization ablation results 105
6.9 Results with varying noise regularization weights 105
6.10 Qualitative comparison with varying conditioning weights 106

7.1 Diagram of our scene generation scheme 113
7.2 Diagram of the pruning process 115
7.3 VAE and DDPM model architectures 120
7.4 Real and generated scenes comparison 123
7.5 Unconditional scenes generated by our method 124
7.6 LiDAR scan sampled from dense generated scene 125
7.7 Semantic segmentation with real and generated data 126
7.8 Semantic segmentation with additional generated data 128
7.9 Conditional generated samples from KITTI-360 and our data . . 131
7.10 Unconditional large scenarios generated by our method 132
7.11 Comparison of real and generated data class distributions 133

List of Tables

3.1 Training epochs used for different label regimes on SemanticKITTI 34
3.2 Contrastive methods fine-tuned to SemanticKITTI 35
3.3 Linear evaluation on SemanticKITTI 36
3.4 Contrastive methods fine-tuned to SemanticPOSS 38
3.5 Linear evaluation on SemanticPOSS 38
3.6 Pre-trained model performance fine-tuned to SemanticPOSS . . . 39
3.7 Contrastive methods fine-tuned to KITTI object detection 40

4.1 Contrastive methods fine-tuned to SemanticKITTI 56
4.2 Linear evaluation on SemanticKITTI and nuScenes 57
4.3 Contrastive methods fine-tuned to nuScenes 58
4.4 Contrastive methods fine-tuned to panoptic SemanticKITTI . . . 59
4.5 Contrastive methods fine-tuned to panoptic nuScenes 60
4.6 Contrastive methods fine-tuned to KITTI object detection 61
4.7 Temporal views extraction ablation 62
4.8 Transformer projection head ablation 62
4.9 Number of aggregated scans ablation 63

5.1 Number of annotated samples in the proposed benchmark 78
5.2 Known and unknown instances segmentation evaluation 79
5.3 Class-wise known instance segmentation evaluation 80
5.4 Known and unknown instances evaluation with different thresholds 82
5.5 Post-processing results ablation study 83

6.1 Scene reconstruction results on SemanticKITTI 100
6.2 Scene reconstruction results on KITTI-360 and our data 101
6.3 Scene occupancy results on SemanticKITTI 103
6.4 Scene occupancy results on KITTI-360 and our data 104
6.5 Chamfer distance with varying noise regularization weights 106
6.6 Results with varying conditioning weights 107

7.1 Maximum mean discrepancy between real and synthetic data . . . 122
7.2 Semantic segmentation model evaluated with synthetic data . . . 122

7.3 Semantic segmentation model with real and generated LiDAR data 125
7.4 Semantic segmentation with real and generated dense data 127
7.5 Semantic segmentation with additional generated data 128
7.6 Network class-wise IoU trained with real and generated dense data 129
7.7 Network class-wise IoU trained with real and generated LiDAR data 129
7.8 Number of point clouds from KITTI-360 and our data 130
7.9 Semantic segmentation with real and conditional generated data . 130
7.10 Comparison of real and generated data class-wise IoU 134