SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Self-Supervised Feature Learning for 3D LiDAR Semantic Segmentation with Neural Radiance Fields

Cavit Çakır

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Self-Supervised Feature Learning for 3D LiDAR Semantic Segmentation with Neural Radiance Fields

Selbstüberwachtes Lernen von Merkmalen für Semantische Segmentierung von 3D Lidar-Daten mit Neural Radiance Fields

Author:	Cavit Çakır
Supervisors:	Markus Herb, MSc.
	Xavier Timoneda Comas, MSc.
Advisor:	PD Dr. Federico Tombari
Submission Date:	15 December 2023

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 15 December 2023 Cavit Çakır

Acknowledgments

In the completion of this thesis, I am deeply grateful for the support and opportunities provided by several key institutions. My sincere appreciation goes to the Chair for Computer Aided Medical Procedures & Augmented Reality (CAMP) at Technical University of Munich (TUM), under whose prestigious and innovative banner I had the privilege to conduct my research. Further, I extend my sincere thanks to the Perception & Fusion department at CARIAD. The environment, resources, and expertise offered by both these institutions have been instrumental in the successful completion of my thesis, enriching my research experience. Additionally, I must express my deep appreciation to my supervisors, whose guidance and insights have been pivotal at every stage of this project. Their support and constructive feedback have not only shaped this thesis but also my growth as a researcher and professional.

Lastly, I must express my profound gratitude to my family and friends. Their continuous support, understanding, and belief in my potential have been a fundamental source of motivation and strength. This thesis is not only a product of my work, but also a reflection of the persistent love and support they have so generously given.

Abstract

The digital age has led to the collection and easy accessibility of vast amounts of unlabeled data, creating a demand for scalable machine-learning models. Self-supervised feature learning methods provide an effective approach to unlock the potential of unlabeled digital resources. In 3D perception tasks, extracting meaningful information from scenes is crucial for the autonomous driving domain but often requires the availability of labeled real-world datasets, which are available in minimal quantities. What if a method could directly learn features for each point in the 3D scene from a collection of images and utilize these features for 3D perception tasks?

In this thesis, we propose a novel approach utilizing Neural Radiance Fields (NeRFs) to achieve self-supervised feature learning for the 3D Light Detection and Ranging (LiDAR) semantic segmentation task. Our method uses images and foundation models to learn volumetric feature maps for various scenes. These maps are then used as supervision for the semantic segmentation model. The approach consists of two main stages. In the first stage, we train a NeRF model for each scene using images and their feature maps which are extracted through self-supervised 2D image feature extractors. This process involves simultaneously learning a continuous volumetric feature map and reconstructing the 3D environment. In the second stage, we train a general LiDAR semantic segmentation model in a self-supervised fashion. We achieve this by supervising the semantic segmentation model with per-point features from the NeRF model for each LiDAR scan. This enables our 3D LiDAR semantic segmentation model to gain geometrically aware features for 3D LiDAR points. Unlike the first stage, the model's domain is no longer a single scene but rather an entire dataset.

We conducted experiments on autonomous driving scenes from the nuScenes dataset, using image and LiDAR data, to evaluate our method, identify its limitations, and benchmark it against baseline and 2D camera projection models. These experiments demonstrate the effectiveness of our approach and provide insights into its performance in real-world scenarios. Our proposed method illustrates the potential of distilling image features into a 3D domain and utilizing these features to train a self-supervised 3D LiDAR semantic segmentation model. While our model surpassed the performance of the baseline model, it was outperformed by the 2D camera projection model. This approach not only showcases the promising capabilities of leveraging image features but also highlights their significance as a pathway for further research and advancements.

Contents

1 Introduction

Autonomous driving aims to revolutionize how we commute by offering enhanced safety, efficiency, and convenience. A critical component of this technology is the vehicle's ability to perceive and interpret its environment accurately [1], [2]. This capability depends on advanced technologies like Light Detection and Ranging (LiDAR) and semantic segmentation. LiDAR, using pulsed laser light, measures distances to create high-resolution three-dimensional environmental maps. In autonomous driving, these maps are vital for safe navigation. Semantic segmentation in the context of LiDAR data, referred to as 3D LiDAR semantic segmentation, involves classifying each point in a LiDAR point cloud into different categories, such as vehicles, pedestrians, and roadways. This classification is essential for autonomous vehicles to understand and interact with their surroundings effectively. However, interpreting LiDAR data poses challenges due to the sparsity of point clouds, similarities in shapes between different classes, and varying data densities and intensities.

With technological advancements, collecting data for autonomous driving has become easier. However, despite the abundance of unlabeled data, a significant challenge remains in the labor-intensive process of labeling 3D point cloud data which are acquired from LiDAR sensors [3]. A significant amount of labeled data is essential for various perception tasks such as semantic segmentation, object detection, and motion prediction, where achieving high performance is crucial [4], [5]. In popular datasets, LiDAR data is often coexisting with other sensor inputs, such as cameras, providing a more comprehensive view [6]–[9]. Currently, researchers are exploring the use of these sensors to either improve model performance or reduce the dependence on large volumes of labeled data [10]–[12]. However, most 3D LiDAR semantic segmentation models are dependent on labeled datasets, requiring a significant amount of this data for effective performance. There are models that have attempted to address these challenges by supervising the 3D semantic segmentation models with only images from cameras [12]. Similarly, some models project LiDAR data onto images, using the features of these images to supervise the 3D model [11]. Yet, this approach also leads to the disregard of 3D information. Additionally, not all LiDAR points are captured in the camera's field of view, resulting in further information loss. These challenges highlight the complexity and ongoing efforts in autonomous driving, particularly in the accurate and efficient processing of 3D LiDAR data.

After discussing the challenges and the current state of technology in 3D LiDAR semantic segmentation for autonomous driving, this thesis seeks to answer the following question: 'How can we effectively reduce the reliance on extensively labeled datasets in training 3D LiDAR semantic segmentation models, by integrating novel approaches

such as Neural Radiance Fields (NeRF) [13] with foundation models?'

This research question is particularly significant given the crucial role of LiDAR technology in generating high-resolution 3D environments, a key factor in enhancing perception models for autonomous driving. Additionally, it addresses the challenge of accessing large volumes of labeled data, which is a common limitation in this domain. To answer it, we introduce a novel approach that integrates NeRF with foundation models to supervise a 3D LiDAR semantic segmentation model. NeRF is a cutting-edge method for creating detailed 3D models from 2D images. It works by synthesizing new views of a scene through a continuous volumetric scene representation, learned from the data. This approach is particularly beneficial in autonomous driving scenarios, where it can effectively represent complex spatial relationships.

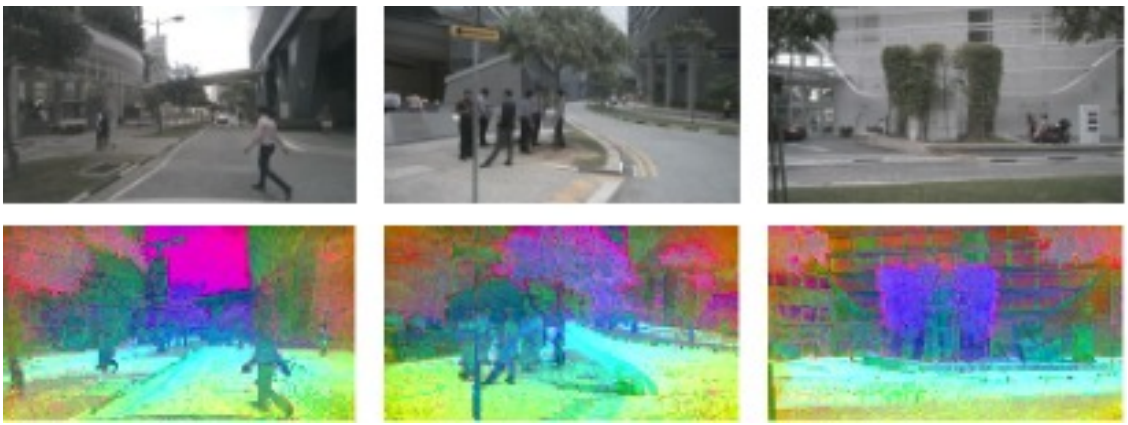

Figure 1.1: Example visualization of images and their DINO features. Principal Component Analysis (PCA) applied for feature visualizations.

To extract semantic information from camera data, we employed DIstillation with NO labels (DINO) [14], a self-supervised 2D feature extractor, as part of the process for learning a continuous volumetric feature space. DINO's authors claim extracted features include hints for semantic segmentation. Therefore, it is a suitable model for learning features from camera data to supervise a 3D LiDAR semantic segmentation model. Example visualization for extracted DINO features is shown in Figure 1.1. Further, we used a NeRF model that incorporates the DINO model, specifically Language Embedded Radiance Fields (LERF) [15]. We constructed 3D volumetric scenes by incorporating both images and their features. This method contrasts with traditional approaches that directly project 2D features extracted from images onto point clouds. Learning a volumetric feature map in 3D space offers advantages over directly projecting 2D features extracted from images onto a point cloud. The volumetric approach allows for a more comprehensive representation of the scene's spatial characteristics, capturing information lost in the projection process. By learning features in 3D space, the supervised model gains an understanding of the scene's depth and volumetric relationships. In contrast, direct 2D-to-3D projection lacks the depth cues essential for accurate spatial reasoning.

Therefore, adopting a volumetric feature map approach enables more comprehensive feature learning. An example of extracted DINO features from our NeRF models is visualized in Figure 1.2.

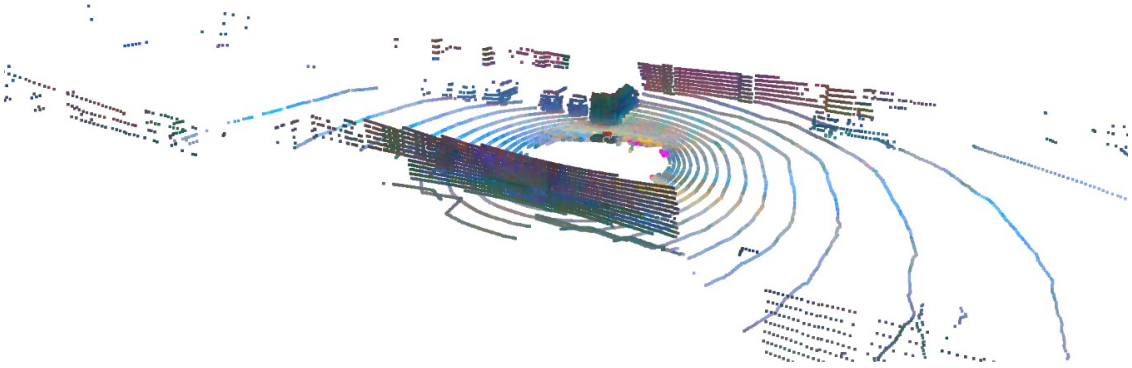

Figure 1.2: 3D Visualization of Extracted DINO Features from Our NeRF Models. PCA applied for feature visualizations.

In our 3D LiDAR semantic segmentation task, we have adapted the Cylinder3D [16] model to better suit our needs. Cylinder3D is a state-of-the-art model known for its efficiency and accuracy in processing point cloud data. It uses a cylindrical partitioning approach to better capture the spatial layout of the data, which is particularly effective for the diverse and complex structures encountered in autonomous driving environments. Our modification includes integrating a feature head with a shared backbone for both semantic analysis and feature extraction. This integration allows the model to use the learned DINO features from each point in LiDAR scans, derived from the NeRF model. Such an approach enhances the model's ability to interpret LiDAR data while reducing its dependence on extensively labeled datasets. One of the critical advantages of our approach is that, during inference, there is no need to train or query a NeRF. This is because we perform offline sensor fusion by training our semantic segmentation model with scene-specific NeRFs. As a result, the inference step becomes very fast, potentially enabling real-time inference performance. The details of the implementation, along with the reasons and advantages of our approach, are explained in the Approach Section.

We evaluated our model using the validation subset of the nuScenes dataset [6]. To comprehensively compare our model, we also trained a camera projection model, which has the same architecture as our approach, with the only difference being in feature supervision. Additionally, we trained a baseline model using only a labeled dataset.

Our tests showed that both our proposed model and the camera projection model performed better than the baseline model, which supports our claims. However, the camera projection model performed slightly better than our proposed model. We have detailed the evaluation setup and discussed possible reasons for these results in the Evaluation Section. Furthermore, we have outlined some potential improvements in the Conclusion and Future Work Section.

2 Related Work

This chapter will provide a literature review and recent techniques in the following broadly divided three categories: (1) Semantic Segmentation, (2) Neural Fields (3) Knowledge Distillation from Large Datasets and Models.

2.1 Semantic Segmentation

Semantic segmentation is an essential and challenging task in machine learning and computer vision. This technique involves assigning semantic labels to each pixel in an image, resulting in an understanding of the scene at the pixel level. The importance of this task is emphasized by its various applications, such as autonomous driving [17]–[19], robotics [20], [21], and medical imaging [22], [23].

Several different approaches exist to approach the semantic segmentation task. One of the first approaches is, Convolutional Neural Network (CNN) [24], which extracts hierarchical features from the image data. Specifically, deep learning-based segmentation methods like U-Net [22] have become standard architectures due to their high performance and ability to handle variable input sizes.

Recently, Vision Transformer (ViT) techniques have started to dominate the field of semantic segmentation [25]. ViT-based semantic segmentation models process the input image as a sequence of pixel patches and use self-attention mechanisms to catch global relations between patches. However, the challenge with ViT for semantic segmentation is the downsampled resolution in the final output due to the patch-based input. The ViT models are explained in detail in the Fundamentals Section.

In the following two paragraphs, two ViT-based semantic segmentation models will be introduced;

In the paper "Emerging Properties in Self-Supervised Vision Transformers" [14], the authors investigate the potential advantages of self-supervised learning for ViT compared to CNN. The authors of the paper observe that self-supervised ViT features explicitly contain information about the semantic segmentation of an image, which is not as evident in supervised ViTs or CNNs. They introduced a self-supervised method called DINO, which they use as a method of self-distillation without relying on labeled data. By combining DINO with ViTs, they achieve a top-1 accuracy of 80.1% on ImageNet [26] in linear evaluation with ViT-Base. The DINO model is explained in detail in the Fundamentals Section.

"Segment Anything Model (SAM)" is another method that is built on ViT. It is a powerful self-supervised model for image segmentation, capable of generating masks

for any object in images or videos. Trained on a diverse dataset of over 1 billion masks. Built on a lightweight architecture which makes SAM able to run on web browsers. It has a strong ability to extract features, accurately segment objects, and handle various scenarios [27].

3D LiDAR semantic segmentation, which involves classifying each point in a LiDAR point cloud into different categories, is also a critical and challenging task due to its wide-ranging applications and its 3D nature. LiDAR data, being 3D, adds an additional level of complexity compared to 2D image data. Moreover, LiDAR data is commonly sparse [28], especially compared to 2D images, which adds to the challenge. 3D LiDAR semantic segmentation helps in categorizing the objects detected by the LiDAR into classes. Perhaps the most significant application of 3D LiDAR semantic segmentation is in the development of self-driving vehicles [29]–[31].

Previous 3D LiDAR semantic segmentation approaches can be roughly grouped into three categories: point-based [30], range-based [32], and voxel-based methods [16]. Point-based techniques use points directly utilize point clouds to categorize every single point according to its semantic label. These methods usually extract significant characteristics from the initial point cloud data. Furthermore, the large volume of points in the cloud introduces computational difficulties.

Range-based techniques involve transforming LiDAR point clouds into a 2D space to generate a range view, enabling the utilization of 2D convolutions for further processing. Nonetheless, these methods face limitations in fully capturing the geometric information due to the 3D-to-2D projection process.

Voxel-based techniques include the transformation of LiDAR point clouds into voxel forms. These forms are then fed to 3D convolutions for the extraction of features. However, these techniques need huge memory capacity. Sparse convolutions address these challenges by significantly quickening the 3D convolution procedure and delivering good segmentation performance.

Another novel voxel-based approach to LiDAR semantic segmentation is Cylinder3D [16]. The method uses a 3D cylinder partition to encode the point clouds in a cylindrical coordinate system, which preserves the 3D topology and balances the point distribution. It also uses a 3D convolution network, which employs residual blocks and context modeling modules based on dimension decomposition. This is used to exploit the high-rank context information present in the point clouds. The Cylinder3D model is explained in detail in the Fundamentals Section.

Besides regular approaches, there are also methods that use image & laser fusion and supervision on the 3D LiDAR semantic segmentation task. With the help of extra 2D image features and laser scans, researchers improved their methods. 2DPASS [11] is a method that uses 2D priors from camera images to improve the semantic segmentation of LiDAR point clouds. In contrast to fusion-based methods that require point-to-pixel correspondences between LiDAR point clouds and camera images, 2DPASS operates on 2D images without paired data constraints. This enables it to perform semantic segmentation without the restrictions of paired data. The purpose of using RGB images

is to provide strong regularization and priors for the sparse LiDAR point clouds, which can enhance the representation learning and segmentation performance.

Lastly, when we analyze the leading 3D LiDAR semantic segmentation benchmarks, such as NuScenes [6] and SemanticKITTI [8], [9], they reveal a clear trend: voxel-based approaches [11], [29], [33] are mostly the top performers. This is also evident in models using Cylinder3D as their backbone [10], [34]. The dominance of voxel-based methods over point and range-based approaches is not only evident in pure segmentation tasks but also in sensor fusion models [10]. This outperformance has motivated us to adopt voxel-based approaches in LiDAR semantic segmentation for the research conducted in this thesis, especially the Cylinder3D model.

2.2 Neural Radiance Fields (NeRFs)

NeRF stands for Neural Radiance Fields. It is a fully connected neural network that can learn 3D scenes and generate novel views based on 2D images. The method includes inputting the complete scene into Multilayer Perceptrons (MLPs). This network uses the information to predict the RGB colors and brightness levels at various locations within the 2D image, resulting in the creation of 3D representations of the object or scene based on the 2D images [13]. NeRFs has multiple practical applications, such as scene and object segmentation, labeling, understanding, as well as editing [35] [36] [37] [38] [39]. NeRF models have several advantages over traditional 3D techniques. They can grasp complex lighting effects and surface details that are challenging to represent using standard methods. Moreover, they can render scenes with high geometric complexity and detail. However, training a NeRF model is computationally intensive, often requiring several hours or even days to process complex scenes. The NeRF model is explained in detail in the Fundamentals Section.

Nowadays, there are several methods that try to make training of NeRF fast but Instant NGP (Neural Graphics Primitives) [40] is the milestone for fastening training NeRF models. An effective multi-resolution hash encoding is the main contribution of this paper. The authors find they can increase sampling speed by 10-100x compared to naive approaches.

NeRF have proven to be effective in various areas, including semantic segmentation. One noteworthy advancement in NeRF's use cases is Semantic-NeRF. Their work has demonstrated that adding a semantic head into a scene-specific NeRF model that encompasses geometry and appearance enables the generation of high-resolution semantic labels for a scene. Their approach can be utilized for interactive labeling in scenes where labeling is limited. However, this technique is not object-aware and cannot handle dynamic scenes [38].

Also, another use case of NeRF is in the domain of panoptic segmentation in dynamic scenes. The approach is called Panoptic Neural Fields (PNF). The main aim of PNF is to break down a scene into objects and backgrounds. To achieve efficiency, they use instance-specific Multilayer Perceptron (MLP) for objects. Additionally, the model

incorporates prior knowledge specific to objects through learned initialization. The background is also represented using a similar MLP, which provides semantic labels. The model takes advantage of existing algorithms for predicting camera poses, object tracks, and 2D image semantic segmentation [41].

2.3 Knowledge Distillation from Large Datasets and Models

Knowledge distillation within the domain of machine learning denotes the process of compressing the knowledge present in large, complex models into smaller models while maintaining their performance. The principle of distillation, as suggested in the paper "Distilling the Knowledge in a Neural Network", contains training a 'student' model to imitate the original or 'teacher' model. The performance of the student model can exceed that of the teacher model. Various methods have been developed to apply this teacher-student paradigm to distill features in computer vision tasks. Several approaches distill the output of a 2D teacher network into a student network that operates in a 3D space [42].

In the domain of 3D semantic segmentation, several works use the supervision of 2D images on the 3D LiDAR semantic segmentation task. With the help of 2D image feature extractors, 3D models became capable of learning extra information. Some of the following works also didn't even use 3D labeled data which leads to reducing the need for labeled 3D data. In the following paragraphs, we will introduce several key models and explore their differences and similarities in relation to our work. This comparison will provide a deeper understanding of how our approach aligns with or diverges from ideas in the field.

The paper "Learning 3D Semantic Segmentation with only 2D Image Supervision" [12] presents a method for learning 3D semantic segmentation without any 3D labeled data. Due to limited and costly ground-truth 3D semantic annotations, the authors propose a method that leverages pseudo-labels derived from 2D image segmentations. Together with image features, they also use RGB images as input features to the 3D network, since they provide useful information for discriminating object classes based on their appearance. There are notable differences between our approach and [12]. While they limit their supervision to regions within the camera's field of view, our method extends supervision to encompass all points captured in a LiDAR scan. Furthermore, we incorporate a specific amount of labeled 3D data into our model, enhancing its accuracy and robustness. In contrast, they do not use labeled 3D data in their framework.

Image-to-Lidar Self-Supervised Distillation for Autonomous Driving Data (SLidR) [43] is a self-supervised method for segmenting and detecting LiDAR point clouds of an autonomous driving environment. Their approach utilizes synchronized images and LiDAR sensors to distill self-supervised pre-trained image features into 3D models, eliminating the need for annotations. By grouping visually similar regions using superpixels, they train a 3D network to match point and pixel features. Experiments on autonomous driving datasets demonstrate the effectiveness of their image-to-LiDAR

distillation strategy. Similar to the approach in [12], their supervision is limited to regions within the camera's field of view. In contrast, our method extends supervision to include all points captured in a LiDAR scan.

Point-to-Voxel Knowledge Distillation (PVKD) [34] introduces a method for distilling knowledge from a large teacher model to a compact student network for LiDAR semantic segmentation therefore it compresses LiDAR semantic segmentation models. PVKD addresses the challenges posed by point cloud data, such as sparsity and randomness, by transferring hidden knowledge from both the point and voxel levels. It incorporates output distillation and intermediate distillation to improve performance. Additionally, the point cloud is divided into supervoxels, and a difficulty-aware sampling strategy is used to prioritize supervoxels containing less frequent classes and faraway objects. Unlike the approaches mentioned earlier, their work only utilizes data that does not originate from cameras. A significant aspect of their research that aligns with our approach is the demonstration of feature learning using the Cylinder3D model.

Furthermore, there are recent works that successfully distilled self-supervised 2D image features to NeRF models. These works proved that NeRF is capable of reconstructing image features as well as the scene. This property of NeRFs motivated us to apply the methodology that the following papers have used, specifically the approach of the LERF [15] model.

In Decomposing NeRF for Editing via Feature Field Distillation (DFF) [44], they proposed a method for editing 3D scenes represented by NeRF using query-based semantic decomposition. The method involves using a teacher-student framework, where the teacher network is a pre-trained 2D image encoder and the student network is a 3D feature field. The method distills the knowledge of pre-trained 2D image feature extractors into a 3D feature field that is optimized in parallel to the radiance field. The feature field can segment and select regions in the 3D space based on user queries of text or image patches, and enable various appearance and geometry edits.

In Neural Feature Fusion Fields (N3F) [45], like [44], they distill 2D image features into a 3D representation via neural rendering. Their purpose for the distillation is to enhance the consistency, viewpoint independence, and occlusion awareness of the features, as well as to incorporate open-world knowledge from pre-trained feature extractors. The paper demonstrates the benefits of N3F for various tasks, such as 2D and 3D object retrieval, segmentation, and editing, on static and dynamic scenes.

In "Baking in the Feature" [46], they propose a method for volumetric segmentation of 3D scenes using NeRF and image features extracted from pre-trained models. They also distill 2D image features to the model by adding a feature head to the NeRF model. The purpose of the distillation is similar to [45] and [44], that is to leverage the semantic and spatial information encoded in the image features to improve the segmentation performance and reduce the amount of supervision required. Different from previous methods, Baking in the Feature has a segmentation head to infer semantic labels from sparse pixel annotations.

Also in FeatureNeRF [47], they introduce a method that uses pre-trained 2D vision models to learn 3D semantic features via NeRF. Similar to [45] and [44], the aim of this distillation is to use neural rendering to transfer knowledge from the 2D models to the encoder, enabling FeatureNeRF to map 2D images into continuous 3D semantic feature volumes. Unlike the previously mentioned papers, this model also extracts deep 3D features from NeRF MLPs. The paper validates the effectiveness of FeatureNeRF as a universal 3D feature extractor through evaluations on 2D/3D semantic keypoint transfer and object part segmentation tasks.

NeRF-SOS [48] is a self-supervised framework for object segmentation using neural radiance fields. The approach uses a contrastive loss that encourages the segmentation features to be consistent with both the appearance and the geometry of the scene. The appearance contrastive loss uses features from a pre-trained 2D image feature extractor to create clusters of visually similar objects. The goal of distillation is to generate object masks from any perspective, using the learned segmentation field. The paper shows that NeRF-SOS outperforms image-based co-segmentation techniques and results in more detailed segmentation outcomes.

The paper LERF [15] is a method for grounding language embeddings from a pre-trained model like Contrastive Language-Image Pretraining (CLIP) [49], as well as visual features from a model like DINO, into NeRF, which enables open-ended language queries in 3D. LERF learns a language field inside NeRF by volume-rendering CLIP embeddings along training rays simultaneously distilling features from a pretrained 2D image feature extractor. The LERF model is explained in detail in the Fundamentals Section.

In the previous paragraphs, we introduced approaches for 3D semantic segmentation and NeRF. Subsequently, we dived into knowledge distillation methods applicable to both 3D semantic segmentation and NeRF. These topics were presented to offer an overview of the current literature and to clarify the rationale behind our model selection. In the following paragraphs, we will explore some works that have approaches similar to ours, discussing their methodologies and comparison with our approach.

In recent studies [50]–[54], NeRF have been integrated into various architectures to develop compact and general models for a range of perception tasks. These approaches involve training a general 2D image feature extractor network, which learns features for any point in the scene by projecting that 3D point onto the image. Subsequently, these features are fed into neural fields consisting of several MLP layers. This methodology enables the models to query a point in their generalized feature extractor, obtaining relevant features for densities and other necessary aspects of the point, assisted by neural fields. Utilizing this approach, tasks such as semantic scene sompletion (SSC), Monocular SSC, and generating novel views from a single image have been successfully achieved.

The primary distinction between our research and the works conducted by [50]–[54] lies in the nature of the specific tasks each project addresses. Our focus is on 3D LiDAR

semantic segmentation, which differs from the tasks they pursue. Another key difference is in our methodology. We approach the problem by first constructing the entire scene, and then sampling features for each LiDAR point from our NeRF model. These sampled features are subsequently used as supervision for our general 3D semantic segmentation model. In contrast, the approaches in those works involve learning a general 2D feature extractor architecture, which is primarily used to learn densities and other required features.

Also, in the paper [55], the authors introduce Neural Semantic Fields (NeSF), a novel approach for generalizable semantic segmentation in 3D scenes using NeRF. They train a NeRF model for each scene and then sample a density grid to obtain the 3D scene representation. Subsequently, they train a shared 3D U-Net [22] architecture for geometric reasoning. The output of this network, which they refer to as the 'feature grid', is then transformed into semantic probability distributions through volumetric rendering.

The differences between our work and [55] begin with the type of data used; we utilize LiDAR data, whereas they rely solely on 2D images. In our approach, we sample features for each LiDAR point from our NeRF models and then use these features to supervise a general 3D semantic segmentation model. In contrast, [55] constructs a density grid in their NeRF world and attempts to align these with semantic labels using volumetric rendering. Additionally, our work is focused on the domain of self-supervised training, while their approach uses only labeled data.

Therefore, based on these works, it is evident that there are currently no other studies in the domain of self-supervised 3D LiDAR semantic segmentation that integrate NeRF and foundation models. This integration is novel in its approach to reducing the reliance on labeled LiDAR data. This gap in existing research highlights the potential impact of our study in advancing the field.

3 Fundamentals

In this section, we will introduce and describe the various technologies used in our study. Our focus will be on explaining what these technologies are and how they function individually. This overview aims to distinguish these technologies from our approach, enabling a better understanding of our contributions in contrast to the existing technologies.

3.1 Foundation Models

3.1.1 Vision Transformer (ViT)

Transformer models, as introduced in [56], represent a revolutionary artificial intelligence architecture in the domain of Natural Language Processing (NLP), significantly changing how machine learning models learn from data. Unlike traditional models that process data sequentially, transformers handle information in parallel, significantly improving efficiency and speed. A key component of transformers is the "attention mechanism", which allows the model to focus on different parts of the input data, determining how much attention to give to each different part. This attention mechanism enables the model to capture complex relationships in the data. By weighing the importance of various elements in the input, transformers can generate more accurate and contextually relevant outputs.

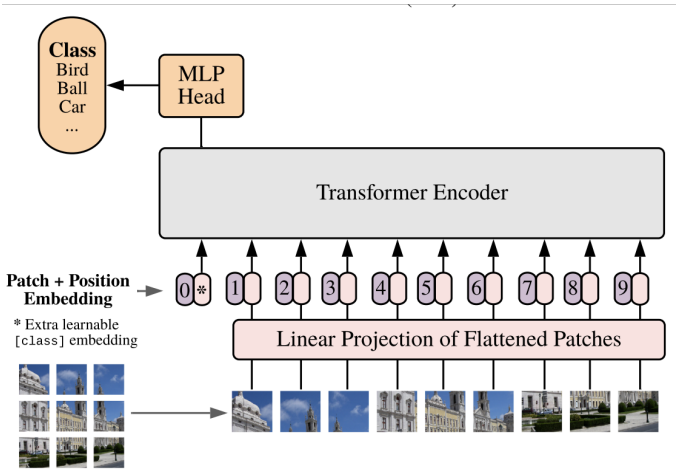

Figure 3.1: An overview of the ViT model. This figure is directly sourced from [25].

The Vision Transformer (ViT) [25] is a novel approach in the field of computer vision, diverging from the dominance of Convolutional Neural Networks (CNNs) [57]. CNN is a type of deep learning algorithm primarily used in image recognition and processing that is efficient and effective at handling spatial data. It detects and learns hierarchical patterns in data, such as edges and textures in images, through layers of convolutions and pooling operations. ViT's origin lies in the application of transformer models to the domain of image processing. The unique aspect of ViT is its treatment of images as sequences, much like how text is treated in NLP. This approach involves dividing an image into a series of fixed-size patches and processing these patches as if they were elements in a sequence. This method enables the model to capture global contextual information from the entire image, in contrast to CNNs which focuses on extracting local features.

The architecture of the ViT, as shown in Figure 3.1, begins by partitioning an image into patches and linearly embedding these patches. To these embeddings, positional information is added to maintain the spatial context of each patch. The important aspect of ViT is its stack of transformer layers, each consisting of an attention mechanism and a feed-forward neural network. This design allows the model to process different parts of the image simultaneously and understand the image globally. Including Layer Normalization and Residual Connections in each transformer layer is essential, as they contribute significantly to the robustness and efficiency of the model's training process. The output from the transformer layers is then utilized for tasks like image classification. Typically, this involves passing the output through a classification head, often including a linear layer. However, this head can be adapted based on specific requirements. What sets ViT apart is its ability to use the global context of images, allowing it to identify patterns and relationships that can be ignored by models that focus only on local features.

3.1.2 DIstillation with NO labels (DINO)

The DIstillation with NO labels (DINO) [14] model depicts a significant advancement in self-supervised learning, especially in the domain of visual representation learning. Self-supervised learning is very advantageous due to its ability to use unlabelled data, avoiding the costs and effort for the data annotation. This makes DINO not only innovative but also practical for real-world applications where labeled data is limited. DINO diverges from traditional supervised learning methods by focusing on knowledge distillation in a self-supervised context. Knowledge distillation, typically used to transfer knowledge from a larger, more complex model to a smaller one is adapted here to enable learning between two models without direct supervision.

The architecture of DINO contains two key components: a student network and a teacher network. These networks share the same architecture, usually a standard ViT [25] model, but their parameters differ. As explained in previous paragraphs, the ViT is a deep learning model that applies the transformer architecture, which is known for its success in NLP, to image recognition. It processes images by dividing them

into fixed-size patches, embedding them linearly, and then feeding the sequence of embeddings into a transformer block, finally to task specific head. During training, the student network aims to mimic the teacher network's output. The student network's parameters are updated with gradient descent, while the teacher's parameters are updated as an exponential moving average of the student's parameters. This strategy ensures the teacher provides stable targets for the student network to learn. A crucial detail of DINO's training pipeline is its reliance on input data augmentations. Both the student and teacher networks are fed augmented versions of the same image, including modifications in color, cropping, and other image transformations. These augmentations ensure the student model learns invariant and robust features, enhancing its ability to generalize from the training data.

In summary, compared to traditional supervised learning models, DINO presents crucial advantages. It eliminates the need for labeled datasets, reduces the risk of memorization for specific datasets, and encourages the model to learn more generalizable features.

3.1.3 Contrastive Language-Image Pretraining (CLIP)

The Contrastive Language-Image Pretraining (CLIP) [49] model presents a novel approach to computer vision, specifically in the field of multimodal learning where both visual and textual data are processed. The idea of CLIP is to connect the content of images and texts in a way that enables the model to perform a wide variety of vision tasks with minimal task-specific training. It is designed to learn visual concepts from natural language supervision, bridging the gap between vision and language understanding.

CLIP's architecture comprises two components: a vision encoder and a text encoder. The vision encoder can be a CNN or a ViT, designed to process images, while the text encoder is a transformer-based design, that reflects the progress made in the field of NLP. These encoders convert raw inputs into embeddings within a shared multi-modal space, bringing the representations of corresponding image and text pairs closer together. This process requires a comprehensive understanding of both visual features and textual semantics, enabling CLIP to effectively interpret both types of data.

The training of CLIP operates in the concept of contrastive learning as shown in Figure 3.2. During training, it is fed with a batch of images and a corresponding set of texts. The model learns by maximizing the similarity between the correct pairs of image and text embeddings, while simultaneously minimizing the similarity between mismatched pairs. This process is achieved through a contrastive loss. By training on a large dataset of images and their corresponding text descriptions, CLIP learns to understand and relate the content of images and texts in a manner that is broadly applicable across various vision tasks. This ability allows CLIP to be used for tasks like zero-shot classification, where it can accurately classify images into categories it has never seen during training.

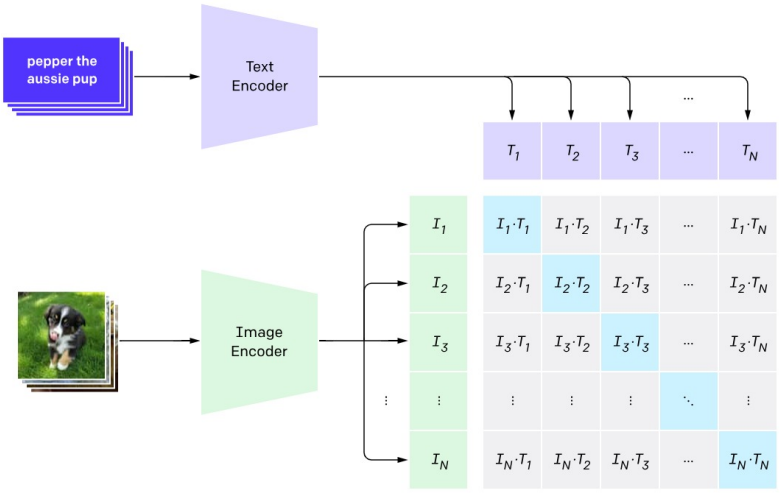

Figure 3.2: An overview of contrastive training of CLIP model, as depicted in this figure directly sourced from [49].

3.2 Neural Radiance Field (NeRF) Models

3.2.1 The Neural Radience Fields (NeRFs)

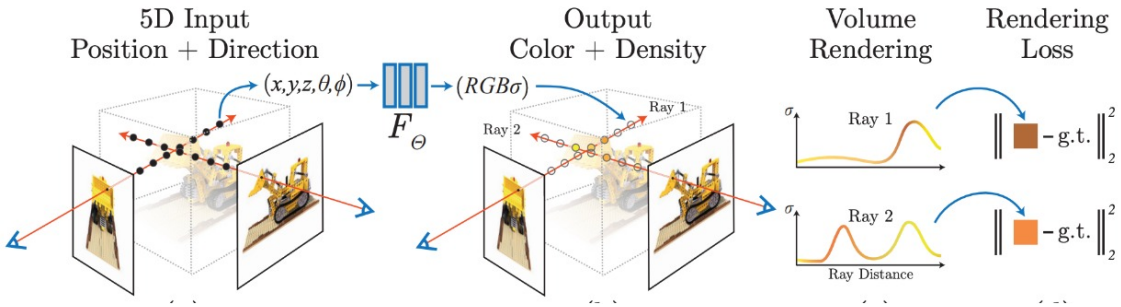

Figure 3.3: An overview of the NeRF scene representation and its differentiable rendering process, as depicted in this figure directly sourced from [13].

Neural Radiance Fields (NeRFs) [13] represent an innovative approach in the field of computer vision and 3D reconstruction. At its core, NeRF is a deep learning model that synthesizes highly realistic 3D scenes from a set of 2D images. The architecture of NeRF consists of a fully connected deep neural network, with MLPs as its fundamental component. These MLPs are tasked with the complex function of learning a continuous volumetric scene representation. What distinguishes NeRF's architecture is its unique handling of inputs and outputs. The MLP network that is shown in Figure 3.3 takes in both 3D coordinates (x, y, z) and 2D viewing angles (θ, ϕ) as input. With the help of the positional encoding layer, the input data is preprocessed for the MLPs. The

positional encoding layer is a key component of NeRF's architecture. This layer is crucial to enable the network to capture high-frequency details in the scene. In traditional networks, directly feeding in coordinates (x, y, z) and viewing angles (θ, ϕ) often leads to a loss of detail, especially in the higher frequency domain. The positional encoding layer addresses this by transforming these low-dimensional coordinates into a higher-dimensional space. It applies a series of sinusoidal functions of varying frequencies to each coordinate, to enable the network to learn the complex mappings between 3D coordinates and their corresponding color and density values. The enhanced representation is then fed into the following layers of the network, allowing the MLPs to learn a more detailed mapping from the input coordinates and angles to the predicted color and density values. The output of the network is two-headed: it predicts both the RGB color and the volume density at each point in space.

The heart of NeRF's functionality lies in its innovative use of volume rendering techniques as visualized in Figure 3.3. Volume rendering in NeRF involves shooting rays from the camera's viewpoint through each pixel in the image plane and into the scene. As these rays traverse the 3D space, the network evaluates color and density at several points along each ray. This process is important for simulating how light travels through the scene, accounting for factors like occlusions and varying densities. The final color of each pixel is an accumulation of these color contributions, integrated along the path of the ray. In Equation 3.1, the color value for a ray is depicted. The volume density function denoted as $\sigma(x)$, is defined as the likelihood of a ray being absorbed by an infinitesimally small particle at a given location $\mathbf{x}$. This concept is crucial for understanding how light interacts with the environment in a 3D space. Additionally, the color value $C(\mathbf{r})$ is computed for each camera ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, bounded by near and far limits t_n and t_f. The term $T(t)$ represents the cumulative transmittance along a ray, signifying the chance of a ray traveling uninterrupted from t_n and t_f, thereby not encountering any particles. To render an image using this continuous neural radiance field, it's essential to evaluate the integral $C(\mathbf{r})$ for each camera ray that passes through every pixel of the virtual camera. This method allows NeRF to produce new views of a scene with remarkable accuracy and detail, effectively capturing complex lighting effects and the properties of various materials.

$$C(r) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{r}(t))\mathbf{c}(\mathbf{r}(t), \mathbf{d})\, dt, \quad \text{where } T(t) = exp\left(-\int_{t_n}^{t_f} \sigma(\mathbf{r}(s))\, ds\right) \tag{3.1}$$

Moreover, an essential aspect of NeRF's process is the calculation of rendering loss between the ground truth color and the color produced by the model. This is achieved through a comparison of the rendered image against the ground truth images used for training. During training, NeRF optimizes the neural network by minimizing this loss, which is typically computed using a Mean Squared Error (MSE) metric. For each pixel, the model calculates the difference between the color value rendered by NeRF and the corresponding color value in the ground truth image. The square of these differences across all pixels and all training images forms the MSE loss as shown in Equation 3.2

where N is the number of data points, x is the predicted value, and y is the ground truth value.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(x_i - y_i)^2 \tag{3.2}$$

Furthermore, the continuous nature of the MLP representation in NeRF allows for smooth interpolations between different views, resulting in photorealistic renderings. The architecture, coupled with the volume rendering process, makes NeRF exceptionally powerful in creating detailed and dynamic 3D models from a collection of 2D images.

While the original NeRF model marked a significant advancement in 3D scene reconstruction, subsequent extensions have been developed to address its limitations and improve efficiency. One notable enhancement is Neural Graphics Primitives (Instant-NGP) [40]. Instant-NGP introduces several key innovations that dramatically enhance the efficiency and scalability of NeRFs. The main innovation of Instant-NGP is its use of a multi-resolution hash table to store neural network features. This approach allows for extremely fast access to these features during the rendering process, significantly accelerating the training and inference times compared to traditional NeRF implementations. The multi-resolution hash table in Instant-NGP effectively manages the trade-off between detail and memory usage. By storing features at multiple levels of resolution, the model can adaptively choose the appropriate level of detail needed for different parts of the scene. This not only improves rendering efficiency but also preserves the high-quality output that NeRF is known for. Additionally, Instant-NGP incorporates advanced techniques like gradient-based optimization and efficient data structures, which further contribute to its speed and performance.

Another efficiency-boosting extension is the use of hierarchical sampling strategies. Unlike the uniform or random sampling in the original NeRF, hierarchical sampling adaptively allocates more points in regions contributing more to the pixel's color. This optimization minimizes unnecessary computations and speeds up the rendering process.

Another noteworthy extension is Mip-NeRF [58]. It is an extension of the original NeRF model that significantly enhances its handling of texture details and anti-aliasing. By incorporating a level-of-detail mechanism, Mip-NeRF adjusts the rendering process based on the distance and viewing angle, improving the representation of fine details and textures. This mechanism functions by accumulating information across the entire area covered by a pixel, instead of focusing on a singular point, effectively minimizing aliasing artifacts. This leads to clearer and more accurate renderings, especially in scenarios where the same scene or object is viewed from different distances and perspectives. Mip-NeRF's approach makes it particularly adept at maintaining the integrity of texture details in complex 3D scenes.

Further state-of-the-art extensions for NeRFs such as pose refinement, piecewise sampler, and proposal sampler are detailed as part of Nerfacto's framework [59] in the subsequent subsection.

3.2.2 Language Embedded Radiance Field (LERF) Model

The Language Embedded Radiance Field (LERF) [15] introduces an innovative approach that integrates foundational models such as DINO and CLIP into the framework of Neural Radiance Fields (NeRF). This innovative approach enables open-ended language queries and learning image features in 3D volumetric space. By embedding foundation model features within a volumetric scene, LERF enables a more context-aware interpretation of 3D spaces, enhancing how machines understand and interact with environments.

LERF extends the capabilities of NeRF by learning a dense, multi-scale feature field. This is achieved by volume rendering DINO and CLIP embeddings along training rays, following a technique similar to Mip-NeRF, as outlined in the previous paragraphs and referenced in [58]. By ensuring that these embeddings remain consistent across multiple training views, LERF achieves a multi-view consistency. This consistency refines the smoothness and accuracy of the feature field, enabling the model to capture complex spatial and textual relationships within the 3D scene.

In the implementation of LERF, the authors adopted Nerfacto [59] as its backbone. Nerfacto is a framework tailored for capturing real static data. It is not an existing published work, but an ensemble of many published methods that authors found work well for real static data. The Nerfacto model combines various effective techniques, including camera pose refinement, proposal sampling, and scene contraction. Its ability to densely sample regions with significant contributions to the render, guided by a density function, is particularly beneficial for enhancing scene reconstruction. Another aspect behind keeping Nerfacto as a backbone; its proposal sampler and density function integration are particularly suited for learning scene representation. The visualization of the pipeline of the Nerfacto model is provided in Figure 3.4. A detailed explanation of the Nerfacto model is given in the following paragraphs.

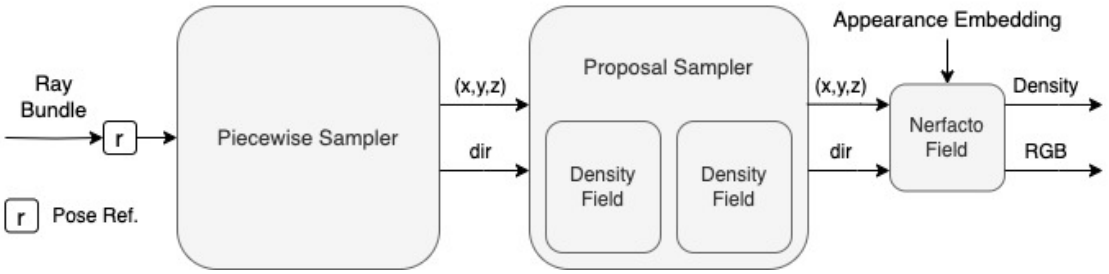

Figure 3.4: Nerfacto Model Pipeline [59].

A key aspect of the Nerfacto model within LERF is Pose Refinement. This feature addresses the common issue of errors in predicted camera poses, which can lead to artifacts and reduced sharpness in reconstructions. By using the NeRF framework's ability to backpropagate loss gradients to input pose calculations, Nerfacto refines these poses, thereby enhancing the overall quality of the reconstruction.

The Piecewise Sampler in Nerfacto, another crucial component, generates scene samples by distributing them strategically based on their distance from the camera. This adaptive allocation ensures a balanced and efficient sampling across the entire scene, enhancing the model's ability to reconstruct both near and distant objects with equal precision.

Furthermore, the Proposal Sampler focuses on sampling the most significant regions of the scene, typically the first surface intersection. This concentration of samples where they are most needed improves the reconstruction quality. The sampler operates based on a density function, which is implemented using a small fused-MLP with hash encoding from tiny-cuda-nn [60]. This method offers both accuracy and speed.

Lastly, the Density Field in Nerfacto is designed to provide a coarse representation of the scene's density, guiding the sampling process. By combining hash encoding with a small fused MLP, they achieve fast querying of the scene. Efficiency is further enhanced by reducing the encoding dictionary size and the number of feature levels. These simplifications do not significantly impact reconstruction quality, as the density function primarily guides initial sampling passes without the need to capture high-frequency details.

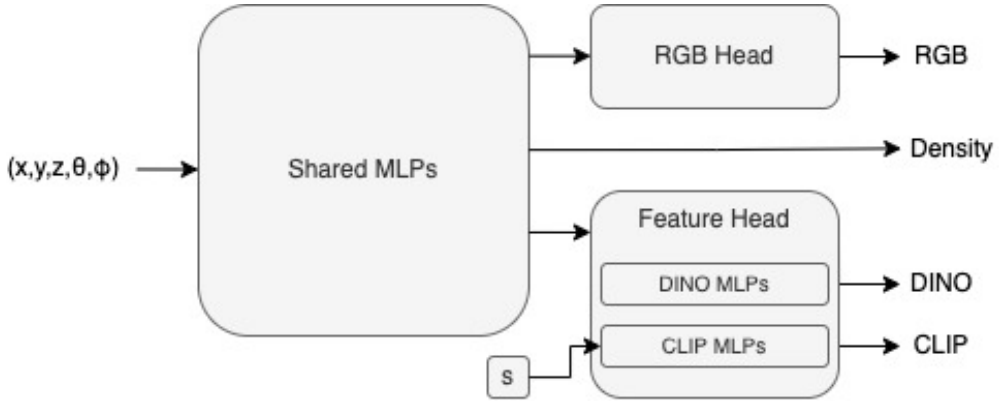

Figure 3.5: LERF model pipeline [15].

The architecture of LERF is illustrated in Figure 3.5, where its structural components are shown. As previously mentioned, LERF utilizes Nerfacto with its default configurations as a backbone. Within LERF, two distinct heads share a single shared component, known as the backbone or as a large MLP block. The first head is specifically trained to capture features for DINO and CLIP, while the second head focuses on generating standard NeRF outputs, such as color. This separation of heads ensures that the learning of features does not interfere with the scene representation process. In the feature head, there are two separate output MLPs, one for CLIP and the other for DINO. Scale 's' is passed into the CLIP MLP as additional information along with hashgrid features. This inclusion of scale information is due to the requirements of the CLIP model. In the context of CLIP, each rendered frustum is supervised with an image crop of scale 's', centered at the image pixel from which the corresponding ray originated. This approach

ensures that the feature extraction is closely aligned with the specific dimensions and perspectives of the input images, enhancing the accuracy and relevance of the features extracted for both DINO and CLIP. While preserving the RGB loss as in Nerfacto, LERF utilizes an MSE loss for both CLIP and DINO losses, with weightings assigned to each. These loss functions enable LERF to optimize feature learning in alignment with the visual correctness of the scene.

Rendering in LERF is the same as NeRF for color embeddings, and density but differs in handling language embeddings. Language embeddings are rendered along a ray with scale parameter 's' that changes based on focal length and distance, forming a frustum. As explained in the previous paragraphs, NeRF takes a position and view direction to output color and density, which are accumulated along a ray for pixel coloration. LERF extends this by adding a language embedding input with scale parameter 's' that are view-independent and invariant to viewing angles. This embedding allows for semantic consistency across different views. The scale 's', similar to the method employed in Mip-NeRF, adapt to altering scales within the image.

In the context of rendering language embeddings into an image, LERF calculation along the ray, $r(t) = o + td$. For rendering language embeddings, LERF operates over volumetric fields rather than discrete points, necessitating a scale parameter 's' at each point along the ray. An initial scale, s_{img}, is set within the image plane, and the scale $s(t)$ at each point is determined as shown in Equation 3.3, increasing in proportion to the focal length and the distance from the ray's origin. This setup geometrically constructs a frustum along the ray path.

$$s(t) = s_{\text{img}} \times \frac{f_{xy}}{t} \tag{3.3}$$

Rendering weights in LERF are computed similarly to NeRF, with $T(t)$ representing the transmittance function, as shown in Equation 3.4, and $w(t)$ as the rendering weight, computed as in Equation 3.5. The integration of LERF then yields the raw language outputs, as depicted in Equation 3.6. Subsequently, each embedding is normalized to the unit sphere, in line with the approach used in CLIP, where the normalization process is represented by Equation 3.7. A weighted Euclidean average, followed by normalization, is used for its simplicity in implementation.

$$T(t) = \int_{t_n}^{t} \exp\left(-\sigma(s)ds\right) \tag{3.4}$$

$$w(t) = \int_{t_n}^{t} T(t)\sigma(t)dt \tag{3.5}$$

$$\hat{\phi}_{\text{lang}} = \int w(t) F_{\text{lang}}(r(t), s(t))dt \tag{3.6}$$

$$\phi_{\text{lang}} = \frac{\hat{\phi}_{\text{lang}}}{\|\hat{\phi}_{\text{lang}}\|} \tag{3.7}$$

The rendering of ϕ_{dino} is conducted similarly to ϕ_{lang}, as outlined by the equation 3.6. However, a crucial difference is that ϕ_{dino} is not normalized to a unit sphere, and scale parameter 's' is not used which distinguishes it from the ϕ_{lang} rendering process. For ϕ_{dino}, rendering weights are computed using the same methodology as ϕ_{lang}, with the transmittance function given by Equation 3.4 and the weight calculation by Equation 3.5. The integration to obtain the raw outputs for ϕ_{dino} follows the process described in Equation 3.8, which does not involve the normalization step and scale parameter 's'.

$$\phi_{\text{dino}} = \int w(t) F_{\text{dino}}(r(t)) dt \tag{3.8}$$

In conclusion, LERF is a significant advancement in integrating foundation models with 3D reconstruction, opening new research domains in artificial intelligence and applications. Its unique approach allows reconstructing 3D scenes, and learning DINO and CLIP features within volumetric feature spaces.

3.3 Cylinder3D Model

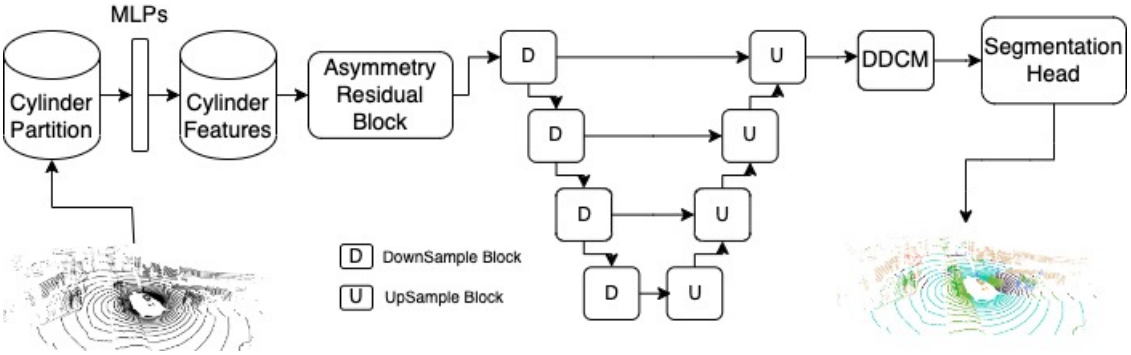

Figure 3.6: Cylinder3D Network Pipeline [16].

Cylinder3D's [16] architecture stands out for its novel approach to 3D semantic segmentation of LiDAR data. The framework initially transforms point clouds from a cartesian coordinate system to a cylindrical one, a conversion that is particularly effective in addressing the varying point densities typically found in LiDAR data from driving scenarios. This model is innovative in its coordinate transformation, its use of asymmetric residual blocks, and a unique context modeling technique known as Dimension-Decomposition based Context Modeling (DDCM). As illustrated in Fig. 3.6, the Cylinder3D framework comprises two main components: the 3D cylinder partition, which is responsible for obtaining the 3D representation, and the 3D U-Net-inspired [22] backbone, which processes this 3D representation. U-Net is a type of CNN originally developed for biomedical image segmentation, recognized for its effectiveness in classifying pixels in an image as semantic labels. Its architecture is unique, featuring a symmetric "U"-shaped design that allows for precise localization and

the use of context, making it highly efficient in tasks requiring detailed image analysis. Cylinder3D is specifically tailored to suit the properties of outdoor point clouds. This includes the Asymmetrical Residual Block, designed to match cuboid-based objects commonly found in driving scenes, such as cars, trucks, and motorcycles. Additionally, the DDCM module is used to exploit high-rank context information in point clouds in a decomposition-aggregation manner. This technique decomposes 3D space into separate dimensions, allowing the model to more effectively understand contextual information, thereby improving segmentation accuracy. The asymmetric residual blocks in Cylinder3D are essentially layers that enable the network to learn from residuals or errors from previous layers, significantly improving learning precision. Furthermore, the design of these components, including the 4-layer MLP network for cylinder partition and the U-Net-inspired 3D segmentation backbone, contributes to the performance of Cylinder3D in 3D semantic segmentation tasks. In the following sections, these components and their contributions to the model's performance will be discussed.

The Cylinder Partition process is a crucial component in the pipeline of Cylinder3D, designed to handle point cloud data effectively. This process begins by transforming points from the cartesian coordinate system to the cylinder coordinate system. This transformation is key to accommodating the unique spatial distribution of LiDAR data. Following this, a cylinder partition step is introduced, which performs voxelization – a process of converting the data into a grid format in the cylindrical space. The final stage in the Cylinder Partition process involves the generation of cylinder features. This is achieved through a simplified version of PointNet [61], which comprises four MLPs. PointNet is a deep learning architecture designed for processing point clouds. It directly consumes point clouds and is capable of classifying and segmenting them, making it innovative for its ability to handle unordered 3D points and recognize 3D shapes. These MLPs work together to process the voxelized data, finally outputting features in a 256-dimensional space. This method of feature extraction is particularly effective in capturing the details and characteristics of the point cloud data, forming the basis for the 3D semantic segmentation capabilities of Cylinder3D.

The Asymmetric Residual Block in Cylinder3D is designed to address the prevalence of cuboid objects in autonomous driving scenes. Additionally, it offers a significant reduction in computational cost compared to conventional 3D convolutional kernels. For instance, a sequence of convolutions with kernels sized 3×1×3 and 1×3×3 achieves the same receptive field as a 3×3×3 convolution but with 33% less computational expense. This asymmetrical residual block forms a fundamental component of both the downsample and upsample blocks in the network. In the downsample block, it is paired with a 3D convolution with a stride of 2 for downsampling. In the upsample block, it integrates low-level features and processes the fused features, enhancing the model's efficiency and effectiveness in semantic segmentation tasks.

The DDCM in Cylinder3D addresses the challenge of representing diverse and complex context information in 3D space, which typically requires a high-dimensional tensor. This approach simplifies the high-dimensional context into several low-dimensional

representations across three dimensions: height, width, and depth. Each of these dimensions is individually low-dimensional, making the overall context modeling more manageable. This decomposition-aggregate strategy effectively handles the complexity of high-dimensional context by breaking it down into simpler views. The model employs three rank-1 kernels (specifically, 3×1×1, 1×3×1, and 1×1×3) to create low-rank encodings in each of the three dimensions. The convolution results are then modulated by a Sigmoid function, which generates weights for each dimension using the rank-1 tensors. Finally, the model aggregates these low-rank activations, summing them up to represent the complete context features. This approach allows for efficient and effective modeling of complex 3D contexts, enhancing the model's ability to understand and segment 3D spaces.

In the Cylinder3D framework, the Semantic Head plays the main role in the segmentation process. It receives an input tensor of dimensions C × H × W × L from the segmentation backbone, where C denotes the feature dimension. Within the Semantic Head, a 3D convolution layer with a 3×3×3 kernel is used, serving as a lightweight and effective component for segmentation. This setup leads to the generation of voxel-based predictions, sized Class × H × W × L, where Class denotes the number of classes, which are essential for the final segmentation output.

To optimize the network, a combination of cross-entropy loss and Lovasz-Softmax loss [62] is utilized. Cross-entropy loss is depicted in Equation 3.9, where p is the number of pixels in the image, ${y_i}^* \in C$ is the ground truth class of pixel i, $f_i({y_i}^*)$ is the network's prediction of pixel i. Lovasz-Softmax is depicted in Equation 3.11, where m(c) is the vector of pixel errors for class c, and $\overline{\triangle_{J_c}}$ is Jaccard index for class c, and C is all the classes. Given a vector of ground truth labels y^* and a vector of predicted labels $\hat{y}$, the Jaccard Index of class c is defined as in Equation 3.10. The cross-entropy loss primarily focuses on maximizing point accuracy, while the Lovasz-Softmax loss is particularly effective in boosting the recall of rare classes. This enhancement in recall significantly contributes to an improved mean-intersection-over-union (mIoU) score. The formula for mIoU is shown in 5.3. Therefore, the total loss function is a sum of these two components, balancing point accuracy and mIoU optimization. This configuration ensures efficient training and effective learning, contributing to the high performance of the Cylinder3D model in semantic segmentation tasks.

$$CrossEntropyLoss = -\frac{1}{p}\sum_{i=1}^{p} log f_i({y_i}^*) \tag{3.9}$$

$$\overline{\triangle_{J_c}} = \frac{\{y^* = c\} \cap \{\hat{y} = c\}}{\{y^* = c\} \cup \{\hat{y} = c\}} \tag{3.10}$$

$$LovaszSoftmax = -\frac{1}{|C|}\sum_{c \in C} \overline{\triangle_{J_c}}(m(c)) \tag{3.11}$$

3.4 Camera Geometry and Projections

Understanding camera geometry is crucial for the process of capturing and processing images. It contains a range of elements from both 2D and 3D perspectives. At the core of this geometry is the camera pose, which refers to the position and orientation of the camera in a 3D space. This pose is crucial as it determines the viewpoint from which the scene is captured. Alongside the pose, the camera's focal length plays a pivotal role. The focal length, a fundamental part of the camera's intrinsics, affects the field of view and the magnification of the captured image. Intrinsics typically encapsulated in the camera's intrinsic matrix, include other parameters like the lens distortion and the pixel dimensions, which are essential in understanding how the camera lens projects the 3D world onto a 2D image plane. Extrinsic parameters are crucial in camera geometry as they define the camera's position and orientation in the world coordinate system. Essentially, these parameters provide a spatial relationship between the camera and the objects in its view. They are represented by a rotation matrix, R, and a translation vector, t. These parameters allow us to transform a point from the world coordinate system to the camera coordinate system. The transformation from world coordinates to camera coordinates is represented as $\mathbf{p}_{\text{camera}} = R \cdot \mathbf{p}_{\text{world}} + \mathbf{t}$, where R is the rotation matrix and $\mathbf{t}$ is the translation vector. To transform coordinates from camera space to world space, the equation is inverted. The transformation is represented as $\mathbf{p}_{\text{world}} = R^{-1} \cdot (\mathbf{p}_{\text{camera}} - \mathbf{t})$, where R^{-1} is the inverse of the rotation matrix and $\mathbf{t}$ is the translation vector. This formula effectively reverses the initial transformation by first subtracting the translation and then applying the inverse rotation.

The projection of 3D points onto a 2D plane involves both intrinsic and extrinsic parameters. The intrinsic parameters are encapsulated in a matrix, K, which includes the focal length, the skew coefficient, and the principal point. The projection can be represented by the following equation:

$$\mathbf{p}_{image} = K \cdot [R|\mathbf{t}] \cdot \mathbf{p}_{world} \tag{3.12}$$

This equation shows how a point in the world coordinates is first transformed to camera coordinates using the rotation matrix R and translation vector $\mathbf{t}$, and then projected onto the 2D image plane using the intrinsic matrix K.

The reverse process, projecting from 2D back to 3D, is more complex due to the loss of depth information in the projection process. To reconstruct a 3D point from its 2D image representation, additional data or assumptions regarding the scene's depth are necessary. The process involves first transforming the point on the 2D image plane using the inverse of the intrinsic matrix K^{-1} and then converting it back to world coordinates. However, without specific depth information, the exact 3D position remains indeterminate.

3.5 COLMAP

COLMAP [63], [64] is state-of-the-art software for Structure from Motion and Multi-View Stereo. It stands out in the field of computer vision for its robustness and versatility in

3D reconstruction from images. The software automates the construction of 3D models from image sets. Initially, it begins with feature extraction, where the software detects distinct points in each image, creating a set of features that can be tracked across the image set. Following this, it performs feature matching, establishing correspondences between these features across different images. This process is critical for determining the relative positions and orientations of the cameras that captured the images. Next, through the Structure from Motion process, COLMAP estimates the 3D coordinates of the tracked features and the camera parameters, including position and orientation. This results in a sparse 3D reconstruction of the scene.

Following camera pose estimation, COLMAP proceeds with dense reconstruction using Multi-View Stereo algorithms. This phase involves constructing a detailed 3D model by analyzing multiple image viewpoints, resulting in a dense point cloud that captures the scene's intricate geometrical details. In this phase, the software creates a dense point cloud by examining the images from multiple viewpoints, addressing the sparseness of the initial reconstruction, and enhancing the detail of the 3D model. Throughout these stages, COLMAP uses various algorithms to optimize the reconstruction process, ensuring accuracy and efficiency. The final output is a comprehensive 3D model that accurately represents the photographed scene, capturing its geometrical and spatial complexities in high detail.

Fundamentally, COLMAP is an invaluable tool in computer vision, providing an efficient and automated solution for transforming 2D images into detailed 3D models. Its combination of robust feature extraction, precise camera pose estimation, and comprehensive dense reconstruction capabilities solidify its position as a key resource in the field of 3D reconstruction and image processing.

4 Approach

This chapter provides a detailed explanation of the novel contributions proposed in this thesis, along with the specifics of the implementation.

4.1 Overview

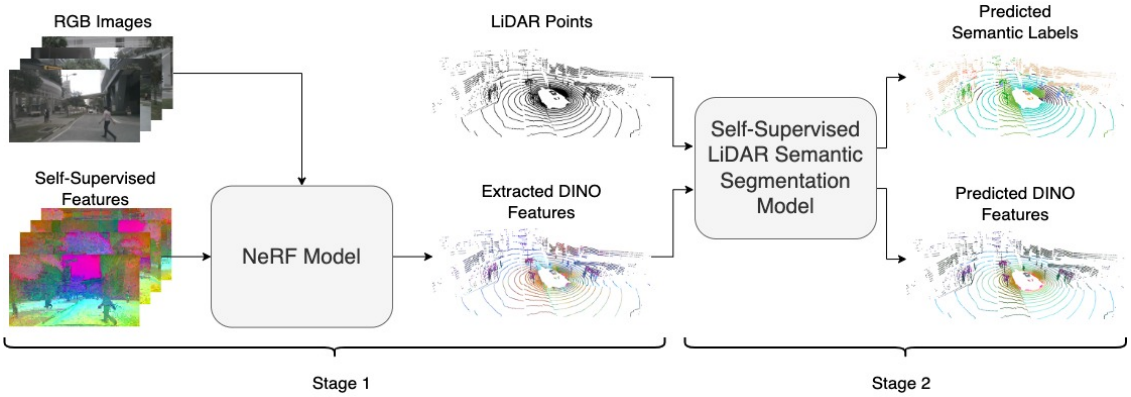

Figure 4.1: Overview of our approach: In Stage 1, scene-wise NeRFs are trained using images and their corresponding features, followed by the extraction of features for each LiDAR scan in the training dataset. Stage 2 involves the training of a generalized Self-Supervised LiDAR Semantic Segmentation Model, using the extracted features.

Our primary objective is to reduce the reliance on labeled data, thereby enhancing the self-supervised 3D LiDAR semantic segmentation task with foundation models. This improvement is achieved through the supervision provided by Neural Radiance Fields (NeRFs) [13], with a specific focus on the training phase. Our approach involves an offline sensor fusion strategy, where techniques containing NeRF, foundation models, and a 3D LiDAR semantic segmentation model are integrated during the training process. This combination is crucial for exploring and understanding the potential of NeRF supervision in improving the accuracy and efficiency of 3D LiDAR semantic segmentation. Significantly, by restricting the multi-modal data processing to the training phase, our method offers notable advantages in terms of speed and reduced complexity during the inference stage, thereby allowing real-time application feasibility.

We propose a novel method for supervising a 3D LiDAR semantic segmentation model using self-supervised DINO [14] features extracted from scene-specific NeRFs. The pipeline of our proposed method is illustrated in Figure 4.1. Our approach leverages

the geometrically aware DINO features provided by NeRFs as self-supervised feature supervision to enhance the performance and accuracy of the 3D LiDAR semantic segmentation task.

Our approach is structured into two main stages, as illustrated in Figure 4.1. In the first stage, we train a NeRF model for each specific scene using images and feature maps derived from the DINO feature extractor. The focus here is on learning a continuous volumetric feature representation while simultaneously reconstructing the 3D environment and extracting relevant DINO features. The second stage uses the features obtained from the trained NeRF model to supervise a generalized model for the 3D LiDAR semantic segmentation task. This is a key step, as it enables our semantic segmentation model to incorporate geometrically informed features, significantly improving its ability to segment objects accurately using both images and 3D LiDAR data even though images are not given as input. Given that our objective is to train a general 3D semantic segmentation model, at this stage, we broaden the model's domain from being limited to a single scene to encompassing the entire training dataset. This expansion into a more generalized approach significantly accelerates the inference step of our process. As a result, our model can independently process LiDAR scans without reliance on NeRFs, allowing for the direct and efficient prediction of semantic labels in the inference step.

4.2 Scene-wise NeRFs

Training scene-wise NeRFs is a critical aspect of our approach, motivated by the need to effectively model spatial environments within volumetric cube constraints. This approach addresses several technical challenges and trade-offs, particularly those related to hardware limitations, computational efficiency, and optimization strategies. Additionally, a significant challenge we face is the extended duration required to train a NeRF model, which impacts the overall efficiency of our process.

In the context of NeRFs, a volumetric cube refers to the 3D space within which the NeRF model learns and renders scenes. This space is defined by its volumetric limits, essentially setting the boundaries for the model's understanding and reconstruction of 3D environments. One of the primary constraints we encounter is the limited size of this volumetric cube that a single NeRF model can effectively learn from. Due to these size limitations, the model's capacity to accurately represent large or complex scenes becomes restricted. Techniques like Block-NeRF [65], a variant of NeRF designed to handle large-scale environments, are crucial for managing this limitation. Block-NeRF divides large scenes into smaller, manageable blocks, each represented by its own NeRF model. These individual models are then combined to create a complete representation of large areas, such as cityscapes. This method overcomes the inherent constraints of the volumetric cube in standard NeRF models.

Without approaches like Block-NeRF, it becomes challenging to represent large scenes, such as those from the nuScenes dataset [6], within a single NeRF model. Our dataset, which includes sequential data from autonomous driving scenarios, inherits this chal-

lenge. In situations where a vehicle is moving along a straight path, capturing the entire environment within a single volumetric cube is inefficient due to the linear progression of the scene. This necessitates the use of advanced techniques like Block-NeRF to effectively model the full extent of these environments, ensuring detailed and accurate scene reconstruction despite the limitations of the volumetric cube.

To address this challenge, we have adopted a strategy of training a separate NeRF model for approximately every 13 consecutive LiDAR scans within scenes. This approach allows us to maintain a high level of detail and accuracy in our environmental modeling. Moreover, in addition to the nature of our dataset, the precision of DINO features per point for LiDAR scans is a crucial requirement for our project. Training scene-specific NeRFs helped to achieve this level of precision. By doing so, each NeRF model is finely tuned to the specific characteristics of its respective scene, ensuring that the DINO features extracted are as accurate and informative as possible. This specificity is critical for the success of our semantic segmentation tasks, as it allows for a more detailed understanding of the 3D environment.

Another important constraint is hardware limitations, as the intensive computational demands of NeRF training require high-performance computing resources. This challenge is further amplified by the necessity for computational efficiency. Optimizing the training process to maximize efficiency in both time and resources is a delicate task. This requires maintaining an equilibrium between speed, quality, and the usage of the computational resources. This constraint sets the maximum number of iterations for the training of each NeRF model. The extended duration required to train a NeRF model poses a significant challenge, as it impacts the overall efficiency and feasibility of our process. This duration constraint not only affects project timelines but also has implications for the iterative development and refinement of models, thereby influencing the quality and applicability of the final outputs. Addressing these constraints is necessary for the successful implementation and advancement of scene-wise NeRFs in modeling spatial environments.

4.2.1 The NeRF model

We chose Language Embedded Radiance Field (LERF) [15] as our NeRF model, a decision influenced by several convincing factors. Firstly, LERF is a method from a recently published paper, ensuring that the model incorporates the latest advancements in the field. This also means that its codebase is up-to-date, an important consideration for maintaining compatibility and leveraging recent improvements. The most important advantage of LERF is its proficiency in learning DINO features effectively, which aligns perfectly with our project's focus on extracting precise DINO features for LiDAR scans. Furthermore, LERF utilizes Nerfacto [59] as its backbone, which is currently the preferred method for training NeRF models due to its architecture which is a collection of current best-performing methods for training NeRFs. Another significant aspect of LERF is its integration with Nerfstudio [59]. Nerfstudio is not only a recent and actively maintained codebase for NeRF models, but it also offers a real-time web viewer.

This viewer is particularly advantageous as it can be operated on Linux servers. By exporting the port, we can monitor the training process in real-time, greatly assisting in debugging and validating our implementation. The active community of Nerfstudio also provides valuable support and insights. Additionally, the public availability of LERF's codebase is a crucial factor, as it allows us to develop our method on top of the existing development. Lastly, the acceptance of LERF into ICCV 2023 emphasizes its relevance and the recognition it has received in the computer vision community, further validating our choice.

The LERF introduces a novel integration of foundational models like DINO and CLIP with NeRF. This integration allows for the processing of open-ended language queries and the learning of image features in a 3D volumetric space. By incorporating foundational model features within a volumetric scene, LERF achieves a more nuanced and context-aware interpretation of 3D spaces, enhancing interaction with various environments. LERF enhances the capabilities of traditional NeRF by learning a dense, multi-scale feature field. This is achieved through volume rendering of DINO and CLIP embeddings along training rays, a technique similar to Mip-NeRF [58]. The focus on preserving the consistency of these embeddings across multiple views allows LERF to achieve a multi-view consistency, which enhances the smoothness and accuracy of the feature field. This is important in capturing complex spatial and textual relationships within 3D scenes.

We have kept Nerfacto [59] as the backbone in our implementation of LERF. Nerfacto is a framework tailored for capturing real static data. It is not an existing published work, but an ensemble of many published methods that authors found work well for real static data. The Nerfacto model combines various effective techniques, including camera pose refinement, proposal sampling, and scene contraction. Nerfacto stands out for its ability to enhance scene reconstruction by densely sampling regions contributing significantly to the render, guided by a density function.

In the LERF's implementation, we kept two distinct heads built upon a shared backbone or a large MLP block. The first head is trained to capture features for DINO and CLIP, while the second focuses on generating standard NeRF outputs like color. This multi-head ensures that feature learning does not interfere with the scene representation process. In the feature head, two separate output MLPs are used for CLIP and DINO. We also kept MSE loss for both CLIP and DINO losses, alongside the RGB loss from Nerfacto, to optimize feature learning with visual scene correctness.

We also followed LERF's rendering process, which mirrors NeRF in terms of color and density but differs in feature embeddings. Language embeddings are rendered along a ray with a scale parameter 's', adapting to the focal length and distance to form a frustum. Unlike NeRF, which outputs color and density based on position and view direction, LERF adds language embedding inputs that are view-independent. This ensures semantic consistency across different views. The scale 's' changes with varying scales in the image, similar to Mip-NeRF's method. The rendering of DINO embeddings is conducted similarly to language embeddings. However, a crucial difference is that

DINO embeddings are not normalized to a unit sphere, and scale parameter 's' is not used which distinguishes it from the language embeddings' rendering process. Further details of volume rendering are depicted in the Fundamentals Section.

The only modification we made to the LERF model is implementing the use of different intrinsics for different cameras, as we have 6 different cameras in the nuScenes dataset. This allows the LERF model to handle variations in camera parameters such as focal length, optical center, and lens distortion, ensuring that the rendered images and extracted features are correctly aligned and accurately represent the scene as captured by each distinct camera. This adaptation is crucial for maintaining the correctness of the 3D reconstructions and feature extractions across the diverse range of camera perspectives present in the dataset.

Notably, Nerfacto models the space within a volumetric cube, which constrains the number of frames to train the NeRF models on the nuScenes dataset. Thus, our approach acknowledges the practical limitations posed by the nature of our dataset while still aiming to achieve the desired performance level.

4.2.2 The Feature Extraction Method

Figure 4.2: Example Screenshots of RGB Renders from Trained NeRF Models

A total of 113 NeRF models were trained, using the training split that we selected from the nuScenes dataset. As will be detailed in the Dataset Section, our focus was

on static scenes. This decision was strategic, allowing for more precise training of our NeRF models. Each NeRF model was trained using images captured from six different camera angles. Additionally, all consecutive RGB frames comprising 13 labeled LiDAR scans were included in the training of each individual NeRF model. Example rendered images from trained NeRF models are demonstrated in Figure 4.2. This comprehensive approach ensured that the models were exposed to a wide range of perspectives and scenarios, further improving the training process. After completing the training process, as detailed in the subsequent paragraphs, we extracted per-point features from each scan within these scenes and saved them as tensors for subsequent use. This approach of saving the extracted tensors significantly improves the overall training speed. The advantage lies in the reduced need to repeatedly query the NeRF for feature vectors; instead, we can efficiently utilize the pre-stored tensors. This not only accelerates the overall training process but also enhances overall computational efficiency by minimizing redundant queries to the NeRF model. This extraction process provided per-point DINO features for the subsequent stages of our method, particularly in supervising our 3D LiDAR semantic segmentation model. An example of extracted features is visualized in Figure 4.3.

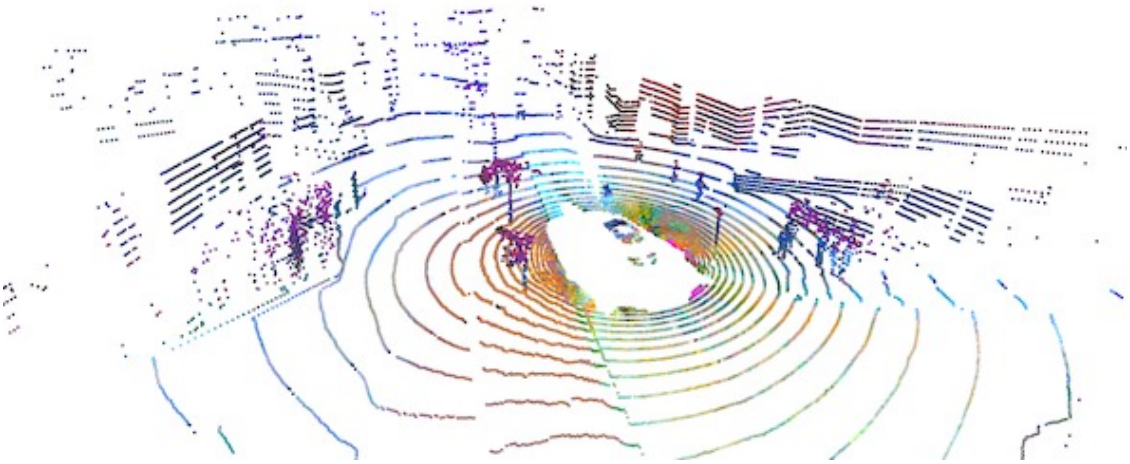

Figure 4.3: 3D Visualization of Extracted DINO Features from Our NeRF Models. PCA applied for feature visualizations.

The training and feature extraction processes in our approach begin with calculating poses and camera intrinsics for each camera and the corresponding images used in training a NeRF model. To achieve this, we utilized COLMAP [63], [64], a robust structure-from-motion, and multi-view stereo software, to recalibrate and compute the poses and camera intrinsics. This step ensures that our data aligns consistently with

the Nerfstudio, facilitating a more efficient and accurate training process for our NeRF models. After applying COLMAP to our training images, we got a transform file that consists of extrinsic and intrinsics for each training camera and the corresponding images. After this step, we train each NeRF with the configurations given in the Experiment Settings Section.

After this point, our goal is to extract DINO features for each LiDAR point in every LiDAR scan of the training dataset. To accomplish this, we project the LiDAR points into the trained NeRF world and later sample the corresponding DINO features for each point within a LiDAR scan with volume rendering techniques. This process of projection, along with the coloring with labels and DINO feature sampling, is visually depicted in Figure 4.4. It provides a clear illustration of how the LiDAR points are integrated into our trained scene and how their features are extracted. This representation helps in understanding the process of mapping each LiDAR point to our NeRF model and extracting the relevant DINO features.

The transformation of LiDAR points to our trained NeRF model's coordinate system is a complex, multi-step process, described mathematically as follows:

1. **LiDAR to Camera Coordinate Transformation:**
   The transformation process begins with the initial LiDAR points, represented in their native LiDAR coordinate system as P_{LiDAR}. The primary goal here is to align these LiDAR points with the camera's coordinate system. Using the nuScenes API, we obtain the transformation matrix $M_{\text{LiDAR}\rightarrow\text{Camera}}$. This matrix is necessary for converting LiDAR coordinates into the camera coordinate system, thereby aligning the LiDAR data with the visual data captured by the camera. The transformation is mathematically represented as:

$$P_{\text{Camera}} = M_{\text{LiDAR}\rightarrow\text{Camera}} \cdot P_{\text{LiDAR}}$$

2. **Camera to COLMAP Camera Coordinate Transformation:**
   The next step involves transforming these camera coordinates into COLMAP's calculated camera coordinate system.
   This requires the transformation matrix $M_{\text{Camera}\rightarrow\text{COLMAP}}$, calculated by COLMAP, and is represented as:

$$P_{\text{COLMAP}} = M_{\text{Camera}\rightarrow\text{COLMAP}} \cdot P_{\text{Camera}}$$

3. **COLMAP Camera to NeRF World Coordinate Transformation:**
   The final transformation step involves converting the COLMAP camera coordinates into the NeRF world coordinate system. This step is crucial for preparing the data for query into the NeRF model, which requires data in its specific NeRF world coordinate system. The NeRF model provides a transformation matrix and scaling factor $M_{\text{COLMAP}\rightarrow\text{NeRF}}$ and s, which we apply to the coordinates to accurately position them within the NeRF world. This transformation allows for

the projection and fitting of these poses within the volumetric cube of the NeRF model. This transformation is expressed as:

$$P_{\text{NeRF}} = s \cdot M_{\text{COLMAP}\to\text{NeRF}} \cdot P_{\text{COLMAP}}$$

By sequentially applying these transformations to original LiDAR points that lie in one training camera's field of view, we map points from their original LiDAR coordinate system to the NeRF world coordinate system. Each step involves specific matrix multiplications that accurately align the point cloud data within the context of our NeRF model's spatial framework. An example result of this projection is depicted in Figure 4.4. After completing the first LiDAR to NeRF world projection for one training camera, we use the coordinates of points in both the LiDAR and NeRF coordinate systems to acquire a mapping transformation matrix $M_{\text{LiDAR}\to\text{NeRF}}$. This matrix enables the direct transformation of points from the LiDAR coordinate system to the NeRF coordinate system. Subsequently, using the $M_{\text{LiDAR}\to\text{NeRF}}$ matrix, we project all the points in the LiDAR scan into our NeRF world, as described by the equation:

$$P_{\text{NeRF}} = s \cdot M_{\text{LiDAR}\to\text{NeRF}} \cdot P_{\text{LiDAR}}$$

After projecting LiDAR points into the NeRF world, we extract features using a technique similar to LERF's training of its feature network for DINO features. For each LiDAR point, we generate a ray $r(t) = o + td$ that originates the training cameras and is directed to the calculated positions of the LiDAR points in our NeRF world and input this ray into our trained NeRF model to accumulate the DINO feature vector. The output, consisting of DINO features, is captured and stored. The data obtained after the extraction process is visually depicted in Figures 4.4 and 4.5. To extract these features, our approach involves volume integration within the LERF to accumulate features. This process is described by the following equation:

$$\phi_{\text{dino}} = \int w(t) F_{\text{dino}}(r(t))\, dt$$

In this equation, ϕ_{dino} represents the accumulated feature vector derived from DINO features along a ray. The integration is performed over the rendering weights $w(t)$, computed as in Equation 3.5, and the feature vector $F_{\text{dino}}(r(t))$ at each point $r(t)$ along the ray. This process differs from ϕ_{lang} done in LERF for CLIP features, as it does not involve normalization to a unit sphere nor the use of a scale parameter 's'. The rendering weights and transmittance function are calculated using the methodology of ϕ_{lang} with adaptations specific to ϕ_{dino}, as outlined in Equations 3.4 and 3.5. The output, which consists of DINO features, is then saved for further supervision in the 3D LiDAR semantic segmentation model.

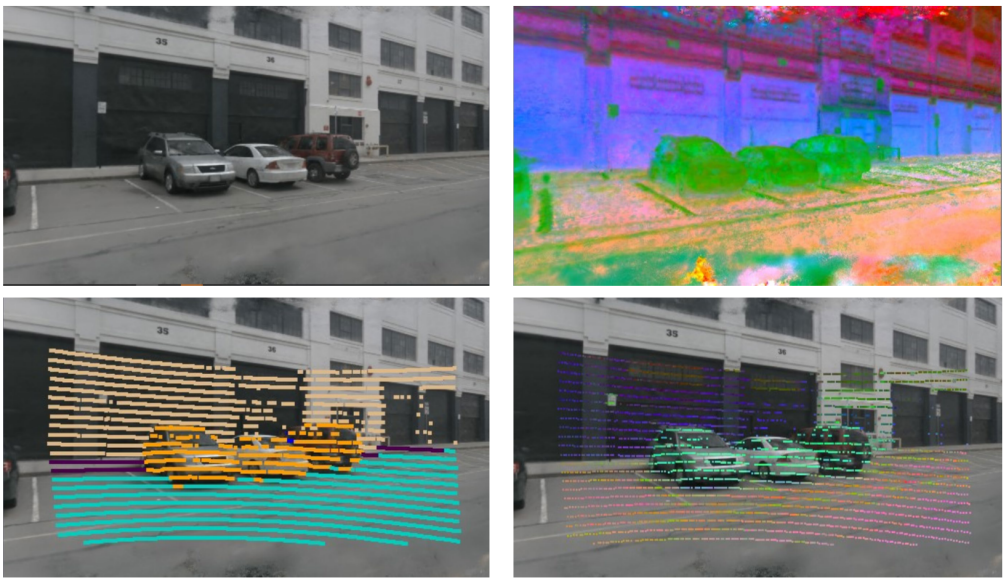

Figure 4.4: Screenshots from Nerfstudio's Viewer: The top left shows the rendered RGB image. The top right displays the rendered DINO features. The bottom left illustrates the projection of labeled points within the 'Front Left Camera' into our NeRF world. The bottom right presents the corresponding DINO features of these projected points. PCA applied for feature visualizations.

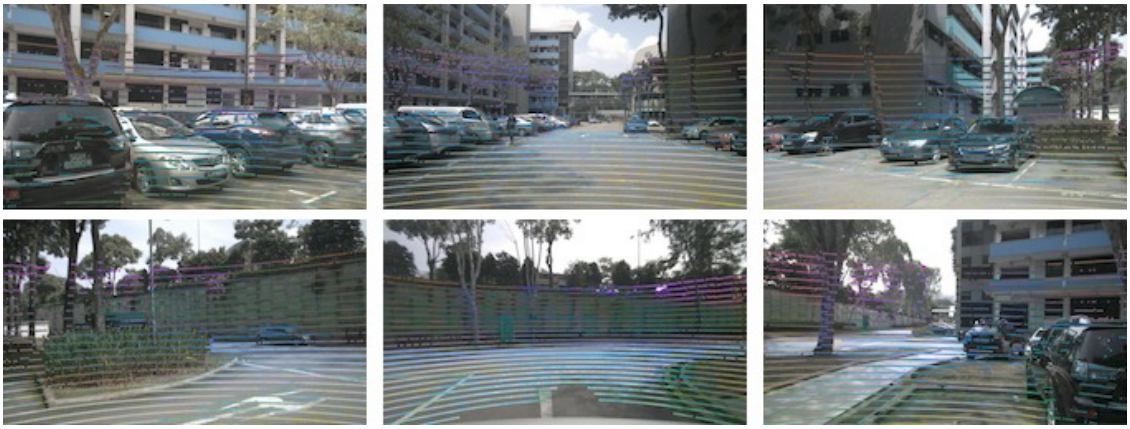

Figure 4.5: Projection of LiDAR Points into NeRF World with Learned DINO Features. PCA applied for feature visualizations.

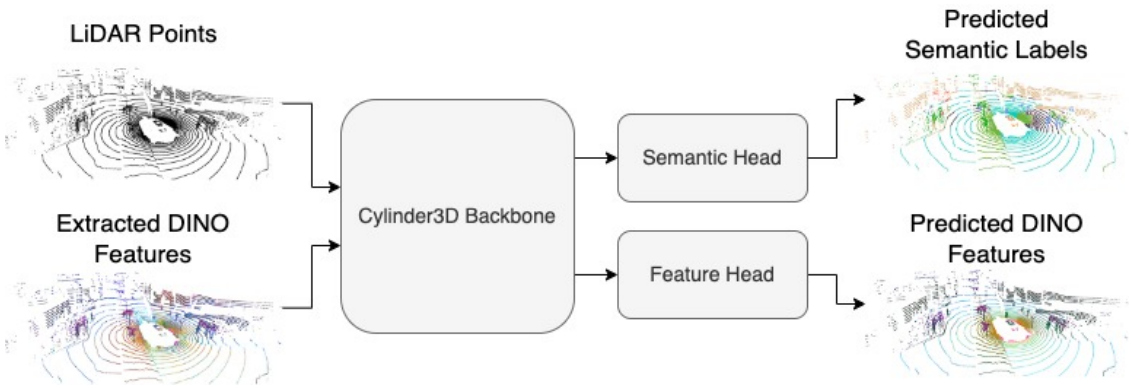

Figure 4.6: Overview of Our 3D Semantic Segmentation Model Pipeline: The pipeline begins with the input of LiDAR points and extracted DINO features from our NeRF models into the Cylinder3D backbone. Following the backbone, the pipeline includes two components: the Semantic Head, responsible for predicting semantic labels, and the Feature Head, focused on predicting DINO features.

4.3 3D Semantic Segmentation Model

4.3.1 The Backbone

The pipeline we have used for utilizing the learned 3D features is depicted in Figure 4.6. In this pipeline, for the backbone of our 3D LiDAR semantic segmentation, we have selected Cylinder3D [16]. Specifically, we utilized the implementation of Cylinder3D as presented in the LaserMix [10] paper. To integrate our approach into this setup, we added a feature head to the model. This addition is designed to effectively use the extracted per-point features, integrating them into the segmentation process. This integration is crucial for improving the accuracy and effectiveness of our 3D LiDAR semantic segmentation, making the most of the detailed features extracted from our unlabeled data. The advantages of this integration will be discussed in the Feature Head Section.

We chose LaserMix's codebase for several reasons. One of the most important reasons is its recent development, which incorporates the latest versions and methodologies. The open-source nature of LaserMix is advantageous, allowing us direct access to the codebase and modify and adapt it to our specific approach. However, while the codebase lacks components for unsupervised training, it does provide a well-implemented backbone for the Cylinder3D model. Additionally, the implementation of a data loader for the nuScenes dataset significantly helped our data handling process. The credibility of LaserMix is also supported by its acceptance and publication in CVPR 2023, a highly esteemed conference in the field. Furthermore, the active engagement of the authors, demonstrated by their responsiveness to issues, provided an additional layer of reliability to the project.

Cylinder3D's architecture offers a novel approach to 3D semantic segmentation of

LiDAR data, transforming point clouds from cartesian to cylindrical coordinates, which addresses the variable point densities in LiDAR data from driving scenarios. Fundamental to the architecture of this model are its innovative coordinate transformation, the use of asymmetric residual blocks, and a specialized context modeling technique called Dimension-Decomposition based Context Modeling (DDCM). As depicted in Fig. 3.6, Cylinder3D integrates a 3D cylinder partition for creating 3D representations and a 3D U-Net-inspired backbone for processing these representations. Cylinder3D tailors its approach for outdoor point clouds, especially with its Asymmetrical Residual Block suitable for cuboid objects like vehicles, and the DDCM module, which effectively processes high-rank context information by decomposing 3D space into separate dimensions for enhanced segmentation accuracy. These asymmetric residual blocks are crucial for learning from previous layers' errors, thus refining learning precision. The overall design, including the 4-layer MLP for cylinder partitioning and the U-Net-based 3D segmentation backbone, significantly elevates Cylinder3D's effectiveness in 3D semantic segmentation.

4.3.2 The Feature Head

In our proposed model, we have adapted the Cylinder3D model as a backbone to not only learn from labeled data but also to incorporate learning from extracted DINO features. This modification, as shown in Figure 4.6, involves a significant architectural change, allowing the model to learn from feature supervision. The integration of a feature head alongside the main Cylinder3D model brings several key benefits.

One of the advantages of this approach is the improved feature extraction capability. By utilizing a shared backbone between the semantic segmentation and feature heads, the model benefits from a more comprehensive feature extraction process. This shared model ensures that the backbone is optimized to learn both semantic understanding and feature learning tasks, enhancing the overall quality of feature extraction. Additionally, consistency in feature learning is another significant benefit. With a shared backbone, the features learned for both semantic understanding and specific feature learning are consistent. This ensures that the features are representative of the same underlying data distribution and characteristics, which is beneficial for the effectiveness of the model to accurately predict semantic classes. Moreover, the shared backbone acts as a form of regularization, helping to prevent overfitting. Since the backbone layers must generalize well across two different tasks, they are likely to learn more robust and generalizable features.

In our model, from the Dimension-Decomposition based Context Modeling (DDCM) module of the Cylinder3D backbone, we acquire 64-dimensional per-point features from voxels. In parallel, in the semantic head, these 64-dimensional voxel features are used for calculating logits for semantic labels. Additionally, we have 256-dimensional cylinder features for each point which are obtained from the 'Cylinder Partition' part of the Cylinder3D architecture, as outlined in The Fundamentals Section. By concatenating these two feature sets, we obtain a comprehensive 320-dimensional feature set for each

point in a LiDAR scan. This approach is essential for guaranteeing distinct per-point features since the features derived from DDCM alone are not sufficiently distinctive within the same cylindrical voxel. This is because DDCM-derived features are sampled from cylindrical voxel grids. The process of sampling voxel features involves collecting sets of features for the points contained within a single grid block. As a result, points that lie in the same block share the same features, which limits the ability to distinguish features on a per-point basis. Another important aspect of this process is avoiding backpropagation of the loss through the cylinder features which are obtained from the beginning of the Cylinder3D backbone, from 'Cylinder Partition', to prevent the model from shortcutting and to ensure the training of all parts of the Cylinder3D backbone.

Following the concatenation of 64-dimensional per-point voxel features from DDCM, and 256-dimensional point features from 'Cylinder Partition', we introduce two MLP layers with a dimension of 384, employing leaky ReLU activation functions and batch normalization. The output dimension of the MLP layers is set to 384, corresponding to the dimension of the DINO features extracted from NeRFs, which are also 384. This selection is further explained and discussed in the Ablation Study Section. The output from this MLP head is then used to calculate the loss between the predicted features and the extracted features from the NeRF model. This comprehensive approach, leveraging the benefits of a shared backbone and feature head, significantly enhances the model's capability in semantic segmentation tasks by integrating per-point DINO features.

To optimize our network, we adopted a strategy similar to the one used in the Cylinder3D model, with some key additions customized to our specific requirements. The optimization process for default Cylinder3D involves a combination of cross-entropy loss and Lovasz-Softmax loss for increasing point accuracy and the mIoU metrics which are shown in Equations 3.9 and 3.11. In our implementation, these two losses are kept as same, ensuring a balanced contribution to the overall training process. In addition to these losses, we introduced an extra component to the loss function: the MSE loss, which is depicted in Equation 3.2. This MSE loss is calculated between the outputs of the feature head and the extracted DINO features from our NeRF models. Since we are doing a regression task, we chose MSE loss due to the faster convergence compared to the cosine similarity loss. The total loss function of our network is composed of the cross-entropy loss, Lovasz-Softmax loss, and weighted feature loss. This configuration, with cross-entropy loss optimizing for accuracy, Lovasz-Softmax loss enhancing the recall of rare classes, and feature loss improving feature representation, is key for efficient training and effective learning. It ensures that the network not only improves semantic segmentation accuracy but also effectively integrates and uses the features extracted from the NeRF models. This approach significantly contributes to the improvement of the Cylinder3D model, particularly in the self-supervised semantic segmentation tasks.

To supervise the learned 3D features from NeRF models, we implemented a training design that efficiently utilizes both labeled and unlabeled data through two parallel data streams, each contributing uniquely to the learning process. In our training loop, with N representing our batch size, we load $N/2$ labeled data points for each pass

through the network. Feature supervision is provided for all N data points. During one epoch, we iterate through the entire training dataset once to train with all the extracted features from the trained NeRF models. However, due to the smaller number of labeled LiDAR scans compared to the total available, we repeatedly loop through the labeled data loader until all unlabeled data have been processed.

The feature head is dedicated to learning DINO features extracted from NeRFs using all samples in a batch, while the semantic head maps the backbone's features to meaningful semantic labels using the labeled data. This dual functionality enables accurate prediction of object classes within scenes, the advantages of this approach are detailed in the previous section. Simultaneously training with both labeled and unlabeled data streams, the model leverages information from both to enhance its learning capabilities. Both data streams are utilized to train the feature head and the shared Cylinder3D backbone. This strategy, which includes integrating features extracted from NeRFs, allows the model to develop a deeper understanding of the 3D environment. Such a comprehensive training approach ensures the model maximizes the benefits from the various data available, significantly improving its performance in semantic segmentation tasks. An example of learned features is visualized in Figure 4.7.

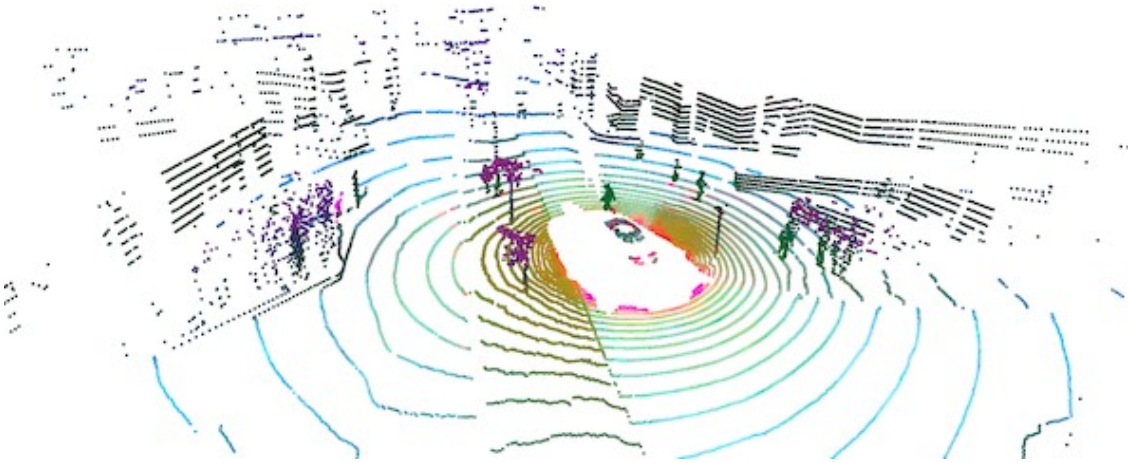

Figure 4.7: 3D Visualization of Features Predicted by the Feature Head in Our 3D Semantic Segmentation Model

5 Evaluation

This chapter presents Experiment Settings, Ablation Study, and Final Model. In the experiment settings, we provided information about the dataset that is used in the following experiments, the baseline model, our model, the camera projection model, and the semi-supervised setup. Then ablation study shows the different experiments done, and their interpretations. In the last part, we provide more results from the selected best-performing model and interpret its results quantitatively and qualitatively.

5.1 Experiment Settings

5.1.1 Dataset

Our primary contribution centers on leveraging features extracted from NeRFs, necessitating high-quality NeRFs both in terms of DINO feature representation and accurate capture of the environment's geometry. A crucial requirement for our dataset is comprehensive RGB information, captured from a wide range of angles relative to the ego vehicle. This comprehensive angular coverage is essential to effectively capture and learn the 3D environment.

In the following paragraphs, we will dive into the specifics of our dataset selection process, detailing the criteria and considerations that guided our choice.

SemanticKITTI [8], [9] and its recent variant, ScribbleKITTI [66], were introduced in 2019 and 2022 respectively. SemanticKITTI offers a considerable volume of data, comprising a total of 23201 annotated LiDAR frames for training and validation. However, a significant limitation of these datasets is their reliance on a single front-facing camera. This setup restricts the ability of our NeRF model to effectively capture the 3D environment, particularly in areas that lie behind or to the sides of the vehicle, which is crucial for extracting meaningful features for the LiDAR scans. Furthermore, while ScribbleKITTI builds upon SemanticKITTI by providing annotations in the form of line scribbles instead of dense annotations, it inherits the same camera setup and therefore does not meet our requirements for comprehensive environmental capture from multiple angles.

The Waymo Open Dataset [7], introduced in 2020, presents another option for our project. It contains a notable volume of data, with 1000 scenes for training and validation, each providing 20 seconds of footage and annotated at 10 Hz, amounting to a total of 200000 annotated LiDAR frames. A notable improvement over the SemanticKITTI and ScribbleKITTI datasets is the usage of five cameras, covering the front, front-left, front-right, side-left, and side-right perspectives as shown in Figure 5.1. While this

setup offers a more comprehensive capture of the environment compared to a single front-facing camera, it still falls short in one critical aspect – the lack of rear-facing cameras. Consequently, while decent NeRF models can be trained using this five-camera configuration, the representation of the environment behind the vehicle remains insufficient. This limitation particularly impacts the quality of extracted DINO features for LiDAR scans from the rear, as they cannot be effectively trained with the camera setup that Waymo provides. Furthermore, a significant challenge in using the Waymo Open Dataset is the absence of an established benchmark. This absence makes it difficult to evaluate and compare the performance of our work in the context of 3D LiDAR semantic segmentation.

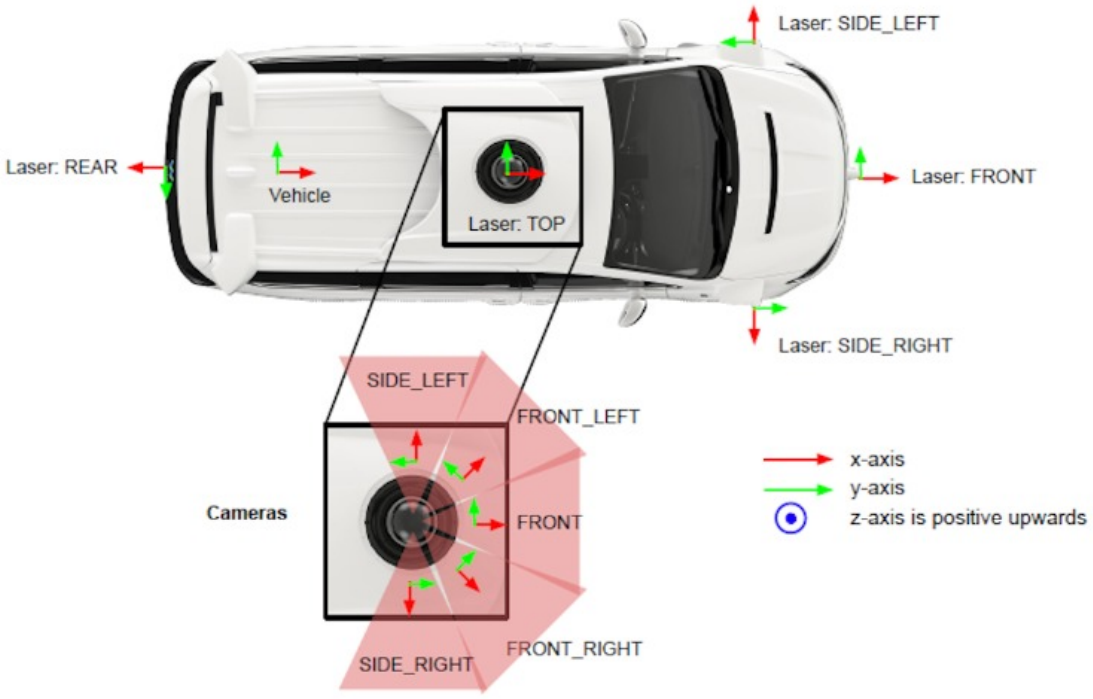

Figure 5.1: Overview of the Camera Setup in the Waymo Dataset [7]

The nuScenes dataset [6], released in 2020, appears as a particularly suitable choice for our project. It contains 850 scenes for training and validation, with annotations provided at 2Hz, resulting in a total of 35149 annotated LiDAR frames. A distinctive feature of nuScenes is its six cameras setup, strategically positioned to cover 360 degrees around the vehicle as visualized in Figure 5.2: front, front-left, front-right, back, back-left, and back-right. This extensive camera setup enables the creation of NeRF models that can effectively capture and reconstruct the entire surrounding environment of the vehicle, with RGB images providing views in every direction. The ability to train NeRF models with such a comprehensive environmental range is a crucial factor in our decision to proceed with the nuScenes dataset. Its 360-degree capture capability aligns perfectly

with our project's requirements, ensuring that our NeRF models are well-trained by diverse visual inputs from all around the vehicle.

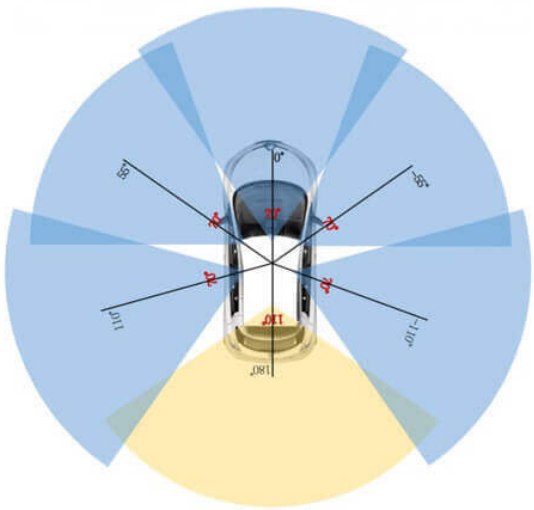

Figure 5.2: Overview of the Camera Setup in the NuScenes Dataset [6]

Further, the nuScenes dataset includes 1000 driving scenes, each 20 seconds long and recorded in challenging urban landscapes of Boston and Singapore. The nuScenes dataset provides annotations for 23 object classes with 3D bounding boxes at a 2Hz frequency across the entire dataset. Furthermore, the nuScenes-lidarseg extension is the required part to use this dataset on our project. It offers extensive LiDAR point annotations, with each point in a keyframe labeled with one of 32 possible semantic categories. This results in 1.4 billion annotated points across 40000 pointclouds and 1000 scenes. An example annotation is visualized in Figure 5.3.

In our study, we selected 1456 scans for training, and the 6,019 scans designated for validation split were kept during this phase. These training scans, chosen from 84 diverse scenes, encompass a wide range of environmental conditions, ensuring a comprehensive training process. During the self-supervised training phase, all 1456 scans were used consistently. For label supervision, we utilized up to 728 labeled scans, which is 50% of the total, while all scans were used for unlabeled feature supervision, allowing our models to effectively learn from both types of data. This approach enhanced learning from both labeled and unlabeled data. The dataset was tailored for static scenes, reducing from 700 to 84 scenes, due to the suitability of the NeRF models for static environments and the challenges posed by dynamic objects, rain, and low light conditions. Detailed information about the usage of labeled scans is further explained in the Semi-Supervised Setup Section.

We customized the training dataset to align with our methodological requirements, focusing on mostly static scenes. This decision was influenced by several challenges encountered in our initial dataset including 700 training scenes, leading us to reduce the number to 84. The primary reason for this reduction is the inherent nature of

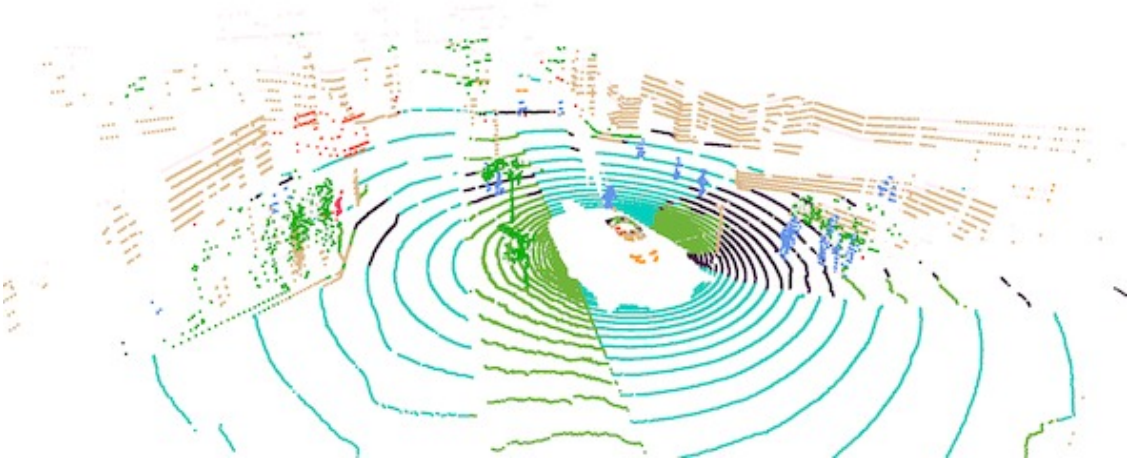

Figure 5.3: Annotated LiDAR Scan: Example from nuScenes Dataset

NeRF models, which are best suited for static environments. They require static scenes to accurately capture and model light and color, and dynamic objects in the scene can significantly disrupt the training process without advanced masking techniques. Additionally, we excluded rainy scenes to avoid the visual noise caused by raindrops, which could degrade the quality of our training data. Lastly, the heavy reliance of NeRFs on light information made it impractical to include night scenes in our training dataset, as they could negatively affect model performance.

5.1.2 Baseline Settings

Our baseline method is a 'supervised-only' model, meaning it utilizes only labeled data for training and validation. This approach demonstrates the model's effectiveness and capabilities without relying on feature supervision, providing a clear standard for performance evaluation. We adopted LaserMix's implementation of the Cylinder3D model, which they also used as a 'supervised-only' baseline. This implementation operates on point cloud data, converting it into three-dimensional cylindrical voxels. This novel approach diverges from traditional voxel representation, offering specific advantages for processing LiDAR data in 3D environments. The details of this model are explained in the Fundamentals and Approach Section.

We used the Cylinder3D model with its original configurations, including 56.26 million parameters and a voxel resolution of [480, 360, 32]. Given a predefined voxel resolution, points in the cylinder coordinate can be partitioned into the corresponding voxel cells. The semantic labels are split into partitioned cylinder voxels, where all points

within the same voxel are assigned a unified label via majority voting. For cylindrical representation, training losses are calculated on the voxel predictions of size [k, 480, 360, 32], where k denotes the number of semantic classes.

Additionally, we set the bounds for voxelization at a maximum of [50, 180, 2] and a minimum of [0, -180, -4]. The dimensionality of the input point features was fixed at 9. We initialized our cylindrical model with a size of 16, to begin with, and set the training to run for 50 epochs. A combination of cross-entropy loss and Lovasz-Softmax loss [62] is utilized. Cross-entropy loss and Lovasz-Softmax are depicted in Equations 3.9 and 3.11. The optimization of the model was done by using Stochastic Gradient Descent (SGD) with a momentum of 0.9, a weight decay of 0.0001, and an initial learning rate of 0.25. To adjust the learning rate, we used a scheduler based on the 'onecycle' policy, which is designed to converge the training faster by cyclically adjusting the learning rate.

To enhance the model's ability to generalize and prevent overfitting, we used several data augmentation techniques. The flip augmentation was applied in a randomized manner where the data might not be flipped or flipped along the x-axis, y-axis, or both. Scaling augmentations were also random, with the scale factor varying between a decrease of 5% and an increase of 5%. Rotation was introduced by randomly rotating the data through any angle between -180 and +180 degrees. Lastly, jitter augmentation was used to introduce a small, normally distributed random noise to the data points. These augmentations collectively aimed to create a robust model capable of handling various transformations and perturbations in the input data. Other than the number of epochs, the rest of the configurations are kept the same as LaserMix's configurations.

There are a couple of differences between our baseline and LaserMix's 'supervised-only' baseline. As mentioned in the previous subsection, we only selected static scenes from the training dataset while LaserMix used all the available data from nuScenes. Additionally, we trained our models with 50 epochs while they trained their baseline model with different numbers of epochs for each split. Despite this change in the training dataset, we maintained methodological consistency with LaserMix, such as using the same validation split and training configs.

5.1.3 NeRF Settings

For our LERF model's backbone, we used most of the default configurations of the Nerfacto model, as was also the case for the LERF model. The Nerfacto model consists of two main MLPs, one dedicated to density and the other to RGB color processing. The hidden layer dimension for both the density and the RGB MLPs is set to 64 by default. This dimension is a crucial factor in the model's capability to accurately process and represent the density and color information of a scene. In terms of its hashmap structure, the model is configured with 16 levels. The base resolution of the hashgrid is set at 16, while the maximum resolution for the hashmap in the base MLP is established at 2048. The size of the hash table is 2^{19}. These resolutions are key to achieving detailed and precise scene reconstructions. Additionally, the default loss function for the RGB

component is the MSE loss as shown in Equation 3.2 where N is the number of data points, x is the predicted value, and y is the ground truth value. This choice is indicative of the model's focus on color representation, ensuring that the output closely mirrors the real-world visual data.

In the implementation of LERF, several modifications were on the details of the Nerfacto network. Specifically, the number of samples in the proposal sampling process was reduced from 48 to 24, a change that significantly accelerates the training process. For CLIP feature learning, the OpenClip ViTB/16 model [25], trained on the LAION-2B dataset [67] is used. This model employs an image pyramid with scales varying from a minimum of 0.05 to a maximum of 0.5, distributed across seven steps. Regarding DINO, the dino-vits8 model, trained on the Imagenet-1k dataset [26] and optimized for 8x8 image patches, is used. The hashgrid used for training language features in LERF is larger than the RGB hashgrid used in the Nerfacto model. It consists of 32 layers, with resolutions ranging from 16 to 512, and features a hash table size of 2^{21} and a feature dimension of 8. The MLP for CLIP incorporates three hidden layers with a width of 256, in the final 512-dimensional CLIP output. For DINO feature extraction, the MLP has a single hidden layer with a dimension of 256 and in the final 384-dimensional DINO output. The Adam optimizer was used for both the proposal networks and fields, with a weight decay of 10^{-9}. An exponential learning rate scheduler was used, reducing the rate from 10^{-2} to 10^{-3} over the first 5000 training steps. The models were trained for a total of 30000 steps. In terms of loss weighting, a lambda value of 0.1 was used for the CLIP loss and 1.0 for the DINO loss. The MSE loss is used for CLIP and DINO losses. The equation for MSE loss is depicted in 3.2.

$$\text{Feature}_{loss} = 0.1 * \text{CLIP}_{loss} + 1.0 * \text{DINO}_{loss} \tag{5.1}$$

5.1.4 3D LiDAR Semantic Segmentation Model Settings

We used the Cylinder3D model as a backbone, with its original configurations, including 56.26 million parameters and a voxel resolution of [480, 360, 32]. Training losses, cross-entropy loss, and Lovasz-Softmax loss are calculated on the voxel predictions of size [k, 480, 360, 32], where k denotes the number of semantic classes. Cross-entropy loss and Lovasz-Softmax loss are shown in Equations 3.9 and 3.11. As mentioned before, feature loss, MSE is calculated on the output of the feature head which is [N, 384], where N denotes the number of points. Feature loss, MSE depicted in Equation 3.2. Additionally, we set the bounds for voxelization at a maximum of [50, 180, 2] and a minimum of [0, -180, -4]. The dimensionality of the input point features was fixed at 9. We initialized our cylindrical model with a size of 16 and set the training to run for 50 epochs. The optimization of the model was done by using SGD with a momentum of 0.9, a weight decay of 0.0001, and an initial learning rate of 0.25. To adjust the learning rate, we used a scheduler based on the 'onecycle' policy [68], which is designed to converge the training faster by cyclically adjusting the learning rate.

To enhance the model's ability to generalize and prevent overfitting, we used several

data augmentation techniques. The flip augmentation was applied in a randomized manner where the data might not be flipped or flipped along the x-axis, y-axis, or both. Scaling augmentations were also random, with the scale factor varying between a decrease of 5% and an increase of 5%. Rotation was introduced by randomly rotating the data through any angle between -180 and +180 degrees. Lastly, jitter augmentation was used to introduce a small, normally distributed random noise to the data points. These augmentations collectively aimed to create a robust model capable of handling various transformations and perturbations in the input data.

5.1.5 The Camera Projection Model

We implemented and trained a camera projection model with our dataset. It is essential for several reasons, especially when evaluating the performance of 3D semantic segmentation models that rely on sensor fusion. Firstly, having a baseline, or 'vanilla', model provides a standard against which we can measure any enhancements or modifications we apply to our model. This comparison is crucial for validating the effectiveness of our approach. Furthermore, incorporating a straightforward, commonly accepted methodology for unsupervised feature supervision into our evaluation framework allows for a clear understanding of our model's capabilities. By comparing our model's performance with another that employs feature supervision, we can see the effectiveness and potential improvements of our approach.

The camera projection model and our model offer distinct approaches to feature extraction, each with its unique advantages. The camera projection model is simple and direct. By projecting LiDAR points onto camera images, it can immediately extract DINO features from the RGB data for those points that are within the camera's field of view. Since the projection technique is the same as used in our model, its steps can be found in the Approach Section. This method is efficient because it bypasses the need for complex training processes, allowing it to operate without the time-intensive task of training NeRFs for each scene. In contrast, our model trains NeRFs with both RGB images and their associated DINO features. This enables it to reconstruct a three-dimensional environment and extract features for each LiDAR point, including those not directly visible to the cameras. Therefore, while our model requires a significant investment in training NeRFs for each scene, it benefits from the ability to sample features across the entire range of the LiDAR scan. Moreover, the camera projection model limits its feature loss calculation to points that are captured by the camera's view, our model extends this calculation to every point captured by the LiDAR, offering a more comprehensive evaluation of feature loss.

The model architecture is kept as same as our model. The camera projection model's approach might be preferable for applications that require speed and simplicity, whereas our model is suited for scenarios where a more thorough scene understanding is beneficial.

5.1.6 Semi-Supervised Setup

In our approach, we developed a semi-supervised learning setup to effectively leverage both labeled and unlabeled data. This approach was crucial for overcoming the challenges caused by the limited availability of densely labeled LiDAR datasets in the autonomous driving domain. Our methodology randomly selects 1%, 10%, 20%, and 50% splits from the available static scenes, using these subsets as our "labeled" data. We chose these varying percentages to analyze the impact of unlabeled supervision on the model's performance and evaluation metrics across different volumes of labeled data. While the labeled data was restricted to these splits, the "unlabeled" data — used for extracting and learning representative features — used all available static scenes. This approach allowed for the capture of comprehensive and rich feature representations, enhancing the model's ability to understand the data, even within the unlabeled partition.

This semi-supervised setup enabled us to maximize the information extracted from our limited labeled data while still capitalizing on the vast amount of information present in the unlabeled data. The incorporation of DINO features, known for their effectiveness in capturing fine-grained details and high-level representations, further enhanced our model's performance, enabling it to generalize well across various levels of labeled data availability.

To evaluate our model, we used Intersection-over-Union (IoU), Jaccard index, per semantic class. Given a vector of ground truth labels y^* and a vector of predicted labels $\hat{y}$, the IoU of class c is defined as in Equation 5.2. Further, we used mIoU for all classes as shown in Equation 5.3 where C is the number of classes, and semantic label accuracy as shown in Equation 5.4 as our key metrics. IoU for each semantic class offered a detailed view of the model's performance across various categories. Meanwhile, mIoU measured the model's overall segmentation ability across all classes. Label accuracy was also crucial, reflecting the model's ability to correctly identify and assign labels, an important part of semantic segmentation tasks. Additionally, we calculated these metrics for both voxels and points. We obtained per-point metrics through trilinear interpolation, helping us understand the model's performance per point in the 3D space.

$$IoU_c = \frac{Areaof Overlap}{Areaof Union} = \frac{\{y^* = c\} \cap \{\hat{y} = c\}}{\{y^* = c\} \cup \{\hat{y} = c\}} \tag{5.2}$$

$$mIoU = \frac{\sum_{c \in C} IoU_c}{C} \tag{5.3}$$

$$Accuracy = \frac{TruePositive + TrueNegative}{AllSamples} \tag{5.4}$$

We conducted our experiments in a Linux environment, using the PyTorch framework on an NVIDIA Tesla V100 SXM2 GPU with 16 GB of RAM. Each 3D LiDAR semantic segmentation model took approximately 6 hours to train 50 epochs and each NeRF model took approximately 3 hours to train.

5.2 Ablation study

To determine the final model design, we conducted a series of experiments aimed at addressing specific challenges and optimizing the model's overall performance:

Addressing Shape Mismatch: Our initial focus was on resolving the shape mismatch observed in feature regression. The experiments compared the effectiveness of directly regressing features with adding a feature head, specifically analyzing the impact of different output dimensionalities (64 and 384). This comparison was crucial in determining whether a more complex feature head would provide significant benefits over a simpler direct regression approach.

Evaluating the Benefit of Per-Point Information: To address the issue of different points having identical features but different labels, we conducted two experiments: the concatenation of per-point features and the summation of per-point features. This experiment was designed to assess whether providing the model with explicit per-point information would enhance its ability to distinguish points with the same features but different labels, thereby improving the overall accuracy of the segmentation.

Assessing the Impact of Feature Weight: Finally, we experimented with different feature weights to understand their effect on the model's performance. We compared the results using feature weights of 0.1, 0.5, 1.0, 2.0, 3.0, and 6.0. This experiment was essential to determine the optimal balance in feature weighting.

Below, we will describe each experiment in detail to tackle the previously mentioned issues.

Direct Regression of DINO Features: It initially appeared to be a viable method, using 64-dimensional voxel features to regress DINO features, which were reduced to 64 dimensions using PCA. However, this resulted in information loss due to using 64-dimensional feature targets. Moreover, the constraint of assigning uniform features within the same cylindrical voxel limited per-point feature learning. The preliminary results showed poor feature learning, confirming our concerns.

Addition of a Feature Head: To enhance feature regression, we introduced a feature head containing two MLPs. We experimented with varying output dimensions, and our initial findings revealed that higher dimensionality correlated with better feature learning and better mIoU. Preliminary results indicated an improvement in feature learning with increased dimensionality, suggesting that the model could effectively learn more complex, high-dimensional features.

Concatenation of Per-Point Features: By detaching per-point features early in the network and concatenating them with voxel-derived features, followed by the addition of two MLP layers, we observed a significant improvement in the model's feature-learning capability. This method's effectiveness is likely due to the use of enriched information available in the model by combining both voxel-level and point-level features. Initial results showed that this method is promising.

Summation of Per-Point Features: This technique is almost the same as the 'concatenation of per-point features' but the only difference is instead of concatenation, we

added voxel-level and point-level features together. As from previous works, we expect that concatenation will work better than summation and preliminary results confirmed our expectations.

Adjusting Feature Weights: Our initial experiments with varying feature weights showed that decreasing the feature weight increases the performance of the model in terms of evaluation metrics.

Each of these experiments was important in choosing the final model design. They were aimed not only at addressing specific issues we identified but also at directly comparing different approaches under the same conditions. The insights gained from these comparative analyses allowed us to make informed decisions about the most effective strategies and configurations for our final model.

In Table 5.1, we compare the approaches mentioned above. Each experiment was conducted using a 10% split of the data and was trained for 50 epochs. Although the addition of a 384-dimensional feature head achieved the highest mIoU, we also evaluated their performance in feature learning. The concatenation of per-point features yielded slightly better performance. Theoretically, the idea behind concatenating per-point features convinced us to proceed with this method.

Experiment Name	mIoU	Accuracy
Direct Regression	28.6	78.3
Sum. of Point Feats.	38.8	85.2
Concat. of Point Feats.	40.1	85.6
Feature Head 384 Dim.	40.4	85.5
Feature Head 64 Dim.	31.4	80.0

Table 5.1: mIoU and Accuracy Metrics for the Ablation Study.

In Figure 5.4, we examine the impact of using different feature weights during training. These experiments were performed using a 10% data split and, due to the lengthy training times, we limited this comparison to 25 epochs.

Finally, we selected a 0.5 feature weighting and point concatenation as our final model configuration. This decision was based on a comprehensive evaluation of the model's performance with various feature weighting schemes. The 0.5 feature weighting, when combined with point concatenation, provided a balanced approach that leveraged the strengths of both feature learning and the robustness of concatenated per-point features. This configuration was not only theoretically sound but also analytically validated through our testing, showing promising results that support our confidence in its potential for accurate and efficient semantic segmentation.

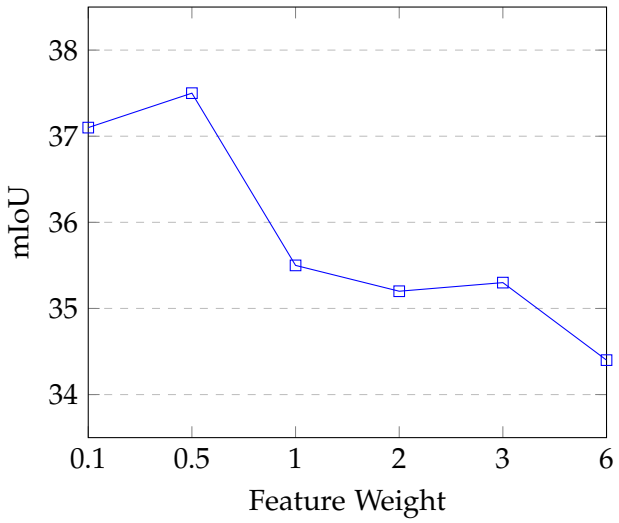

Figure 5.4: The Relationship Between Feature Weight and mIoU.

5.3 Final model

5.3.1 Interpreting Results

Split	Model	mIoU	acc	barr	bicy	bus	car	const	moto	ped	cone	trail	truck	driv	othe	walk	terr	manm	veg
%1	Vanilla	23.3	77.5	0.4	0.0	0.0	61.7	0.0	**3.9**	1.4	**0.1**	8.5	**5.1**	83.6	6.5	22.0	**52.2**	64.8	**62.7**
	Our Model	23.4	**77.9**	0.7	0.0	0.0	**64.2**	0.0	2.1	2.1	**0.1**	**9.6**	4.1	**84.2**	6.2	**24.5**	49.7	65.9	61.7
	Camera Proj.	**23.5**	77.8	**0.9**	0.0	0.0	60.9	0.0	2.0	**2.3**	**0.1**	8.7	3.9	**84.2**	**10.0**	22.8	51.3	**66.4**	62.6
%10	Vanilla	39.3	85.5	24.0	0.3	**0.8**	**78.5**	2.3	26.7	**52.2**	13.7	22.5	44.0	90.2	10.4	47.1	63.1	76.4	77.1
	Our Model	40.0	85.7	24.3	0.3	0.7	77.2	2.6	31.2	51.3	**14.5**	21.9	45.0	90.7	15.3	48.8	**63.3**	76.3	77.1
	Camera Proj.	**40.8**	**86.0**	**25.6**	**0.4**	0.6	77.7	**2.8**	**32.2**	52.1	14.3	**25.1**	**48.3**	**91.3**	**16.3**	**49.2**	63.0	**76.7**	**77.5**
%20	Vanilla	41.6	86.7	23.4	2.0	**1.6**	78.9	4.0	28.8	**55.2**	**23.5**	**25.9**	47.2	**91.9**	8.9	54.0	64.9	**78.2**	78.4
	Our Model	42.7	86.6	25.2	2.8	0.8	78.1	5.4	**36.3**	54.8	20.5	25.2	**49.3**	91.6	19.6	53.3	64.9	78.0	78.3
	Camera Proj.	**43.1**	**86.9**	**25.9**	**2.9**	1.0	**79.8**	**5.8**	36.2	54.8	21.9	22.5	48.8	**91.9**	**21.9**	**54.1**	**65.4**	78.1	**78.5**
%50	Vanilla	45.3	87.4	31.2	1.7	**14.0**	80.1	6.1	32.1	**57.8**	29.9	22.5	47.8	92.3	27.8	57.5	66.6	78.9	79.0
	Our Model	45.7	87.8	**33.5**	2.4	4.8	78.5	**8.5**	**43.0**	57.5	19.2	22.0	48.4	92.7	**36.0**	58.9	67.7	79.3	**79.2**
	Camera Proj.	**46.4**	**88.1**	31.5	**2.7**	4.5	**80.7**	**8.5**	37.7	57.2	**31.9**	**23.5**	**51.9**	**93.0**	34.7	**59.6**	**67.8**	**79.4**	79.1

Table 5.2: Evaluation of Model Performance Across Different Dataset Splits: Abbreviations are used for class names due to space limitations. Models with the best performance in each split are highlighted in bold. Detailed configurations of these models can be found in the Experiment Settings Section.

In Table 5.2, we present a comparative analysis of all three models across the various dataset splits previously introduced. Each model was trained for 50 epochs 3 times to overcome randomness caused by weight initialization. Our final model and the camera projection model were trained with an additional 0.5 weighted feature loss. Apart from this distinction, the configurations for these models were aligned with those of the vanilla model, as detailed in the Baseline Section. The first column of the table is the dataset splits, while the second and third columns provide metrics on the mIoU and

pixel accuracy, respectively. The subsequent columns detail the IoU for each specific class. Due to constraints in page width, class names have been abbreviated. The full names of these classes are barrier, bicycle, bus, car, construction vehicle, motorcycle, pedestrian, traffic cone, trailer, truck, drivable surface, flat & other, sidewalk, terrain, manmade, and vegetation. In our analysis, the best-performing models for each dataset split are highlighted in bold for easy identification.

Our analysis reveals that both our model and the camera projection model outperformed the vanilla model in terms of mIoU and accuracy. Interestingly, the camera projection model showed better performance in mIoU and accuracy across different splits of the dataset compared to our model. These outcomes were unexpected, as we hypothesized that learning from DINO features across all points in the LiDAR scans, as opposed to only those within the camera's view, would be more advantageous. This assumption was based on the premise that incorporating information from all points in our NeRF models should enhance performance in evaluation metrics. We did not observe a significant improvement in evaluation metrics, particularly in smaller dataset splits. This outcome was surprising as we expected a more noticeable improvement due to the ratio between used labeled and unlabeled data. However, the increase in mIoU was found to be almost uniform across each split.

In the following paragraphs, we will dive into the results of each model. After this, we will discuss potential reasons for these outcomes, exploring various factors that might have influenced the performance of the models.

The camera projection model demonstrates consistently high performance across all data splits, with the highest mIoU scores noticeable in all splits. Additionally, this model generally achieves the highest, or nearly the highest, accuracy levels in all these splits. Notably, its performance is noteworthy in certain classes such as 'bicycle', 'drivable surface', and 'sidewalk', where it often achieves the top IoU scores in all data splits. Moreover, the model shows strong results in the 'car', 'cone', 'truck', and 'trail' classes, especially notable in the larger data splits. This overall performance emphasizes the camera projection model's effectiveness in various aspects of class prediction and dataset sizes.

Our model shows several improvements over the vanilla model. Its mIoU and accuracy scores typically fall between the vanilla and camera projection models. Notably, in the 'car' and 'sidewalk' classes within the 1% data split, our model outperforms the other models, showcasing its proficiency in these specific categories. Furthermore, in the larger 50% data split, it achieves the highest IoU for the 'barrier', 'motorcycle', and 'flat & other' classes. This varied performance across different classes and data splits highlights the strengths of our model in certain areas.

The vanilla model, while not the best in overall performance, demonstrates competitive results in specific classes. It is particularly effective in predicting 'bus' and 'pedestrian' classes across all data splits. In these categories, the vanilla model often achieves the highest IoU scores, highlighting its strength in identifying these classes. This consistent performance in 'bus' and 'pedestrian' classes indicates that the vanilla model has specific

areas of proficiency.

5.3.2 Potential Reasons for Results

To understand the performance of our model, it is essential to consider several key factors. The following reasons offer insight into these results.

The most possible reason for the observed performance variations among the models could be linked to the limitations of NeRFs in learning RGB and DINO features for moving objects, as evidenced in our findings. As illustrated in Figure 5.5, there is a notable difference in the ability to capture features for moving objects between directly extracted DINO features from RGB images and those obtained through our NeRFs. The static nature of RGB images allows for more effective feature extraction of moving objects, which is not the case with our NeRFs. In Table 5.2, we can observe the effect of this limitation on the performance of our model, particularly in dynamic classes such as 'bicycle' and 'pedestrian', where it underperforms. In contrast, for static classes like 'flat & other', 'barrier', and 'sidewalk', our model demonstrates better performance, aligning with expectations. This factor is a crucial consideration in understanding the strengths and limitations of our model in various class predictions. A potential solution for this challenge is proposed in the Conclusion and Future Work Section.

Another potential factor affecting the performance of our model could be the use of COLMAP for pose estimation. The process of calculating poses with COLMAP may have resulted in inaccuracies in pose estimation. These inaccuracies, in turn, could lead to the learning of insufficient RGB and DINO features. This aspect is particularly critical because the accuracy of pose estimation directly impacts the quality of the data used for training the model. If the poses are not accurately calculated, the model may not effectively learn the correct features, leading to non-optimal performance, especially in classes where precise spatial orientation and positioning are crucial. Therefore, some defects in pose estimation using COLMAP could have played a role in the observed performance of our model.

A further reason for the varying performance of our model could be linked to the optimization of some NeRF scenes. The volume or complexity of certain scenes might have led to non-optimal optimization, therefore affecting the learning of sufficient DINO features. This issue is evident in the some of rendered RGB images and DINO features from NeRFs, as shown in Figure 5.6. This figure reveals defects in the renders, particularly noticeable in classes like 'motorcycles' and 'roads'. Additionally, Figures 5.7 and 5.8 provide further insight, showing that the camera projection model tends to learn features more effectively than our NeRF-based model. This comparison suggests that the method of feature extraction and learning in the camera projection Model is more robust in capturing and representing the necessary features, particularly in complex scenes. This aspect emphasizes the potential limitations of NeRFs in certain scenarios and their impact on the model's overall performance.

Another likely reason for the limited improvement in our model's performance metrics

is the formatting of our training dataset, which was focused on mostly static scenes. This restriction likely restricted the model's ability to learn from and adapt to more dynamic or crowded scenes. Training on static scenarios can degrade the model's generalization capabilities, especially when later evaluated on diverse scenes with varying levels of activity.

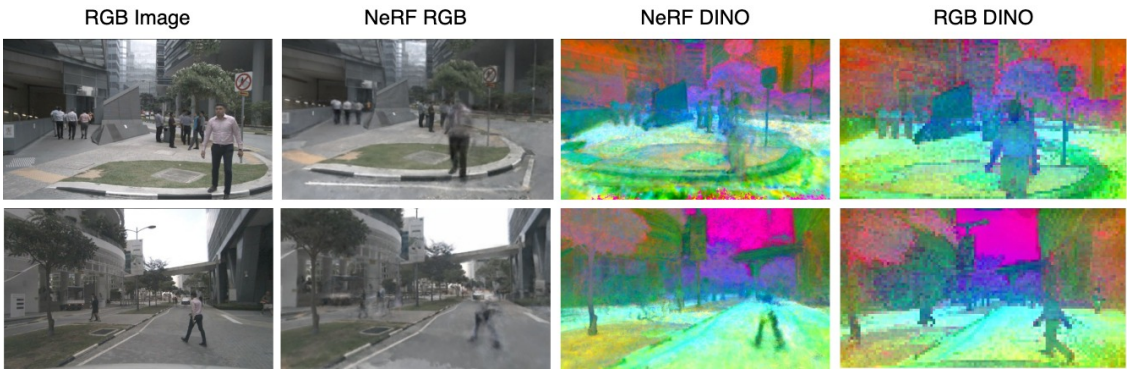

Figure 5.5: Example of Moving Objects Within the Scene. PCA applied for feature visualizations.

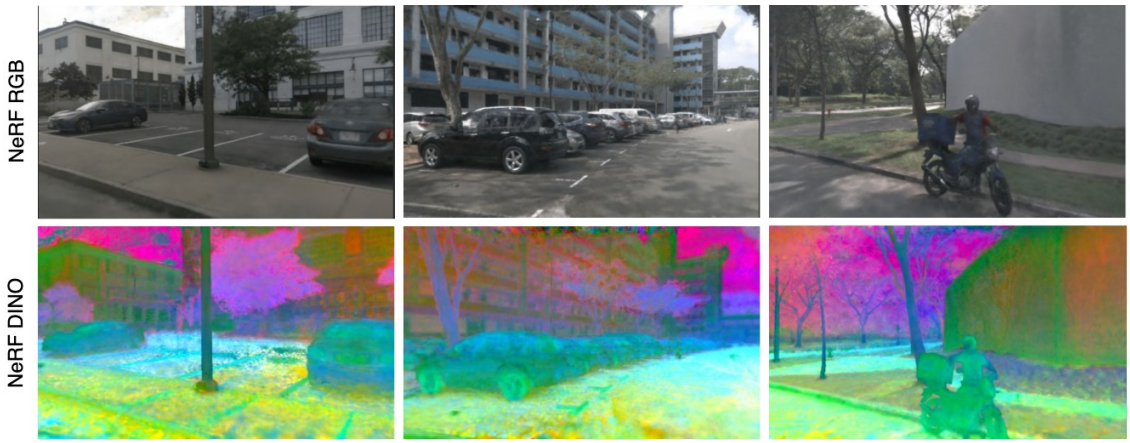

Figure 5.6: Examples of RGB Images from Trained NeRF Models

In Figures 5.7 and 5.8, our goal is to visualize the feature supervision given to the models and how they learn these features. Both our model and the camera projection model were effective in capturing the input features, but the camera projection model showed slightly higher success in learning the provided features. For both models, there is a clear correlation between the learned features and the objects present in the scene. Classes such as 'cars', 'walls', and 'roads' are distinctly distinguishable from other objects, indicating the models' ability to differentiate between various features. This ability is further highlighted in Figures 5.9 and 5.10 which are example scenes from the validation dataset. Despite this, the cues and relationships learned from the training data are still evident, demonstrating the models' capacity to generalize in learning

features.

Also, we compare the output DDCM features of the Cylinder3D model which is directly related to predicting semantic segmentation labels, Figures 5.11 and 5.12 demonstrate improvements obtained by both our model and the camera projection model. Feature supervision leads to more uniform and distinct features for each object type in the scene, indicating improved segmentation accuracy.

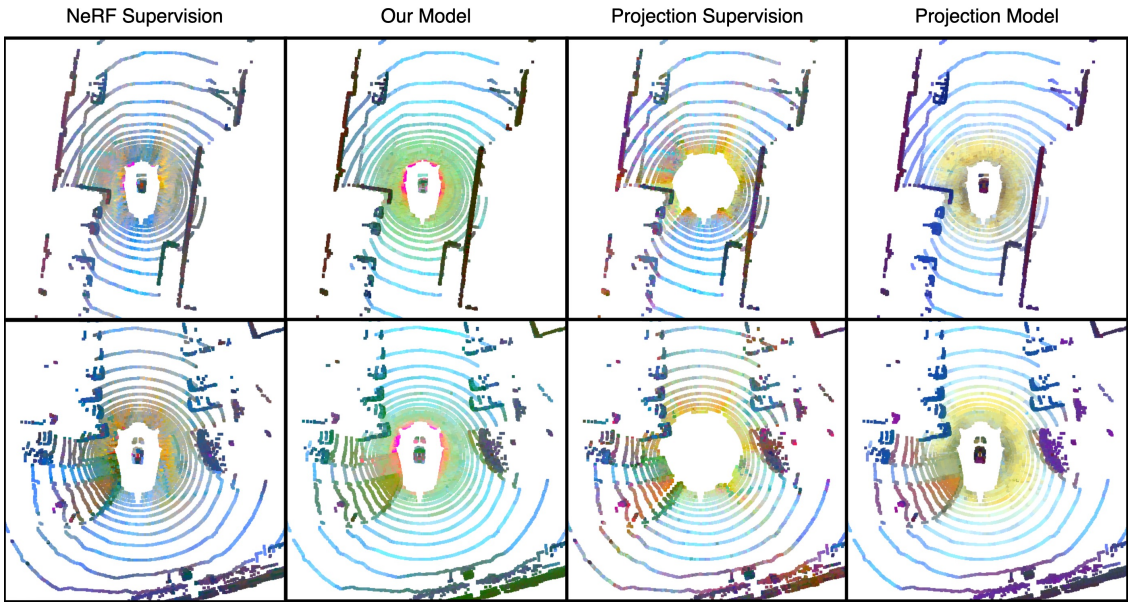

Figure 5.7: Example Scans from the Training Dataset: The models are trained with a 50% split of the dataset. PCA visualizations shown in these images showcase the supervision data for both models, along with the outputs generated by the network's feature head.

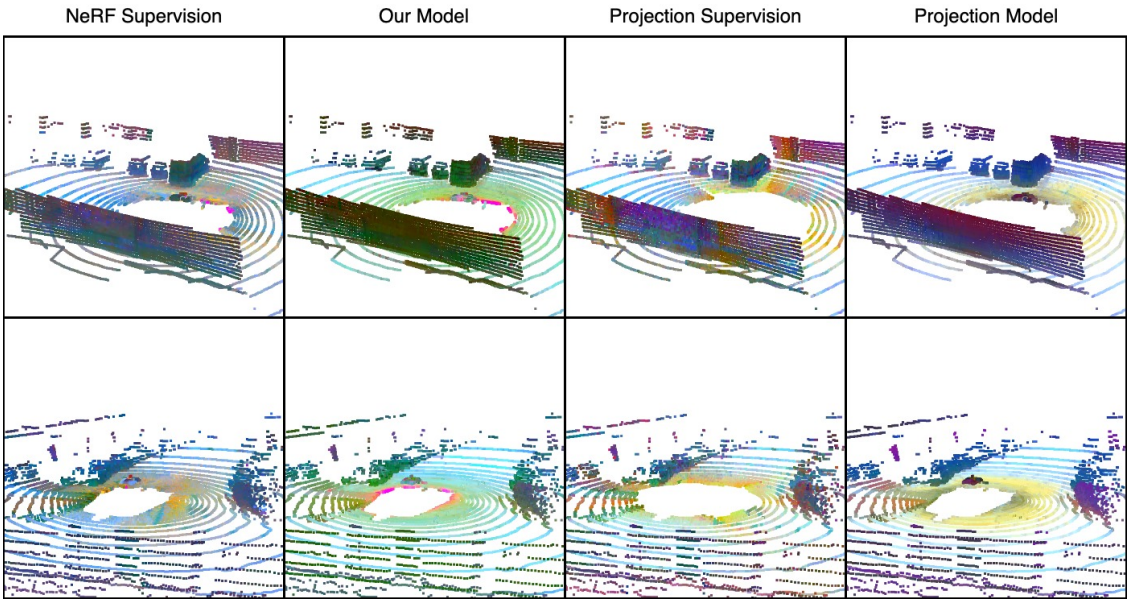

Figure 5.8: Example Scans from the Training Dataset: The models are trained with a 50% split of the dataset. PCA visualizations shown in these images showcase the supervision data for both models, along with the outputs generated by the network's feature head.

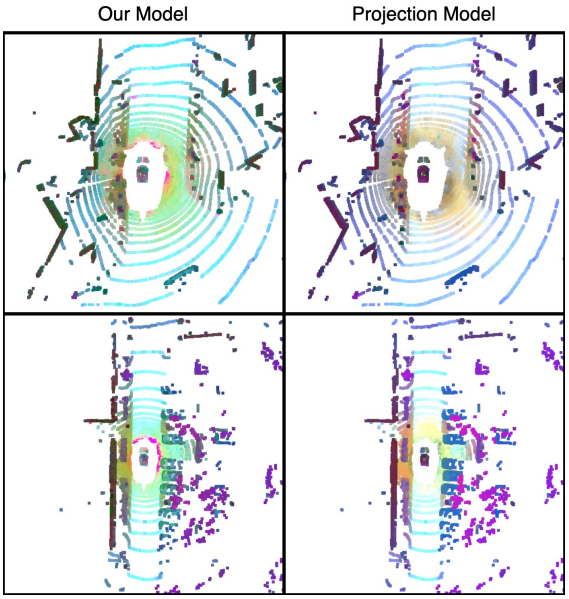

Figure 5.9: Example Scans from the Validation Dataset: The models are trained with a 50% split of the dataset. PCA visualizations shown in these images showcase the outputs generated by the network's feature head.

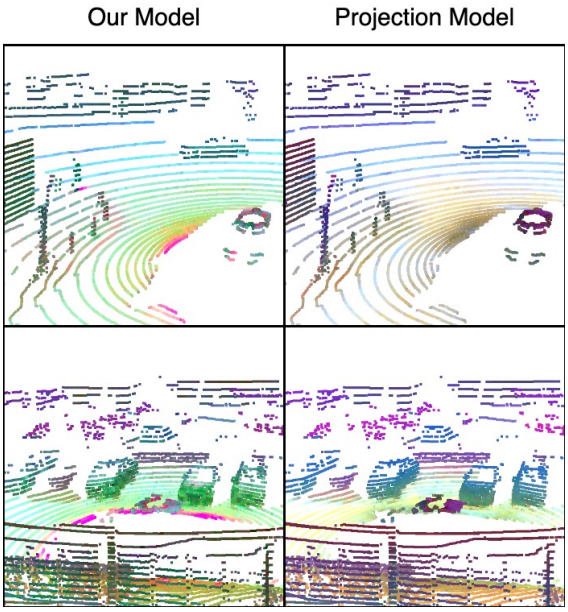

Figure 5.10: Example Scans from the Validation Dataset: The models are trained with a 50% split of the dataset. PCA visualizations shown in these images showcase the outputs generated by the network's feature head.

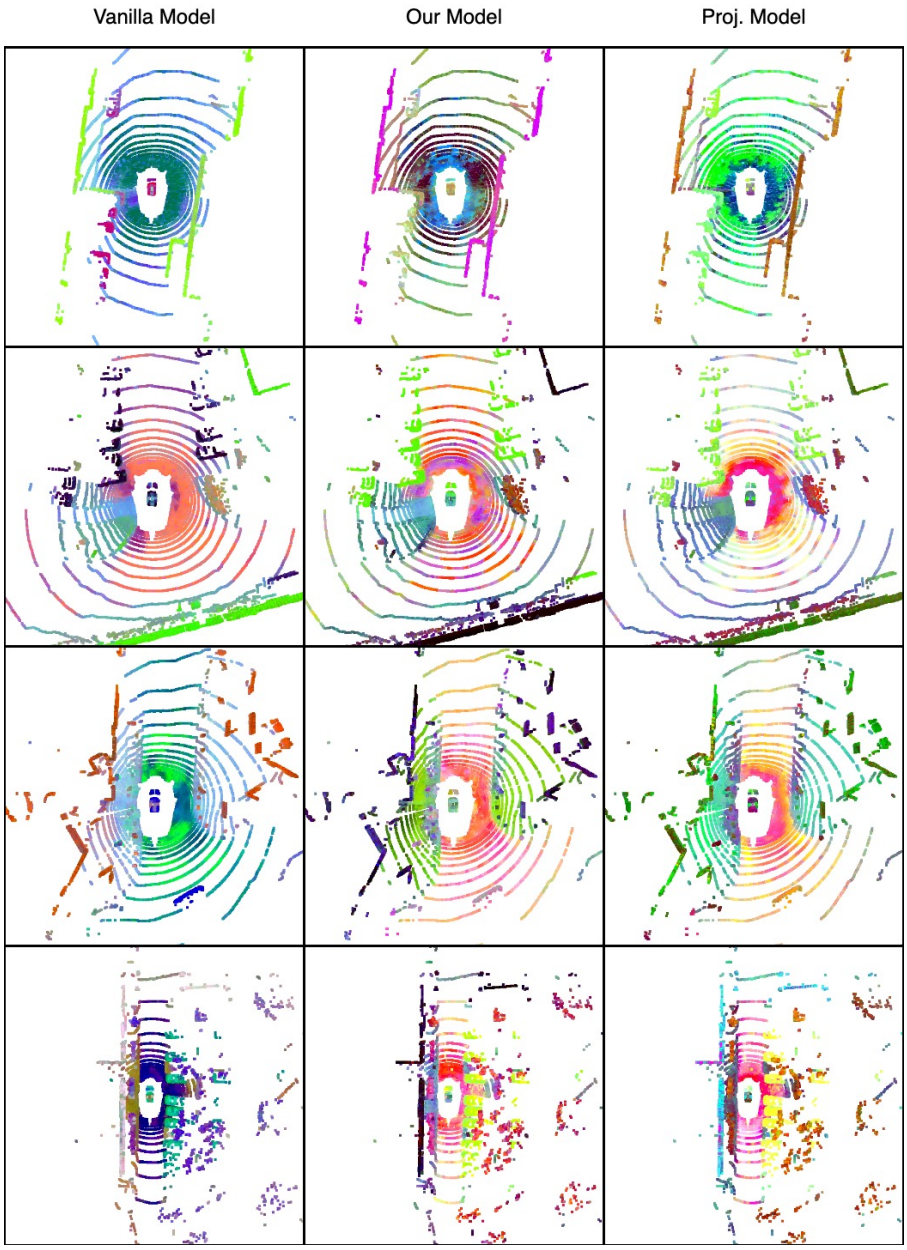

Figure 5.11: The images illustrate PCA visualizations of the output from the Cylinder3D model, used for calculating logits in semantic segmentation. The first two rows feature examples from the training dataset, while the final two rows are from the validation dataset. Models are trained with a 50% split of the dataset.

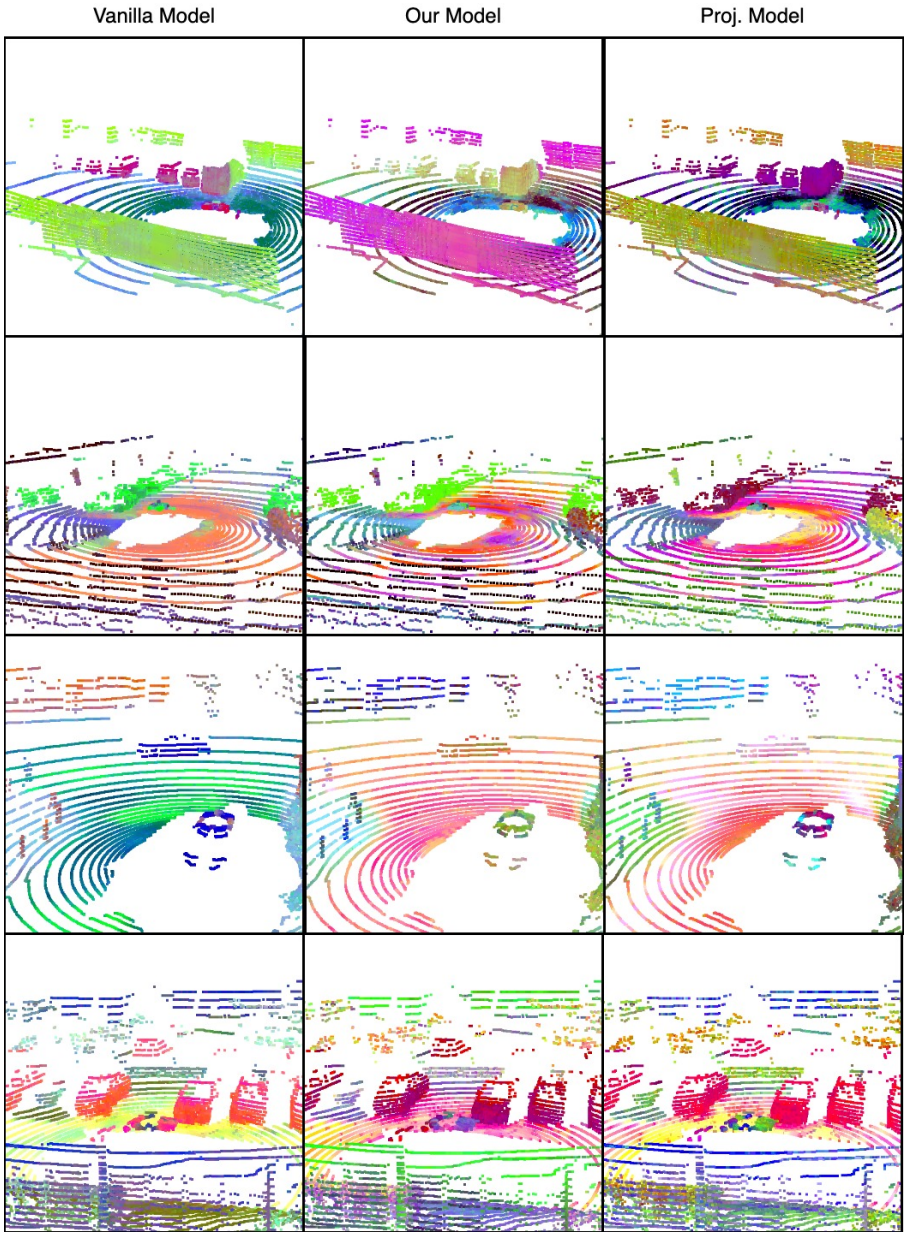

Figure 5.12: The images illustrate PCA visualizations of the output from the Cylinder3D model, used for calculating logits in semantic segmentation. The first two rows feature examples from the training dataset, while the final two rows are from the validation dataset. Models are trained with a 50% split of the dataset.

6 Conclusion and Future Work

6.1 Conclusion

Semantic segmentation is a challenging task, especially in the context of robotics and autonomous driving. The task requires high-precision classification at the pixel level, which becomes even more complex when extended to three dimensions. In 3D semantic segmentation, the sparsity of 3D data adds an extra layer of difficulty, making accurate segmentation a challenging task. Traditionally, most 3D semantic segmentation methods depend on supervised learning, which is heavily dependent on the availability of labeled data. However, labeling 3D data is a labor-intensive and time-consuming process. In contrast, acquiring unlabeled data is relatively straightforward and less resource-intensive.

In this thesis, we addressed the significant challenge of reducing reliance on extensively labeled datasets in the training of 3D LiDAR semantic segmentation models. Our research specifically focused on integrating novel approaches such as NeRF and foundation models with the 3D semantic segmentation model to overcome this challenge. Our proposed solution, a novel self-supervised 3D semantic segmentation approach, directly addresses these issues. Our model leverages the surplus of unlabeled data by training scene-specific NeRFs with images and foundation models. This is achieved by extracting point-wise features from NeRFs, which allows our model to supervise the 3D semantic segmentation process effectively. This innovative approach not only avoids the need for extensive labeled data but also harnesses the potential of available unlabeled data.

The dataset selection for our project was a critical process, particularly due to the requirements of learning scene-wise NeRFs and the usage of a NeRF model optimized for static scenes. Working within the autonomous driving domain, our focus was on datasets that offer both camera and labeled LiDAR data. After a detailed review, we chose the nuScenes dataset for its comprehensive six-camera setup, which captures a 360-degree view around the ego vehicle. This dataset provided us with the essential RGB and LiDAR data for each scene, aligning perfectly with the requirements of our project.

Using the LERF model, we trained 113 NeRF scenes and extracted DINO features from 1456 LiDAR scans. These extracted features served as unlabeled supervision for training a general self-supervised 3D semantic segmentation model, which was a Cylinder3D model with an additional feature head. To evaluate the performance of our model, we conducted tests using the validation subset of the nuScenes dataset. For a comprehensive performance comparison, we also trained the Cylinder3D model

solely with labeled data and developed a projection-based model. This comprehensive approach allowed us to gain a deep understanding of the strengths and limitations of our model in the context of common methodologies.

In the Evaluation Section, we outlined our experimental setup and conducted an ablation study. We tested with various splits of labeled data to assess the impact of labeled-unlabeled data ratios on training. Our results showed that our proposed method outperformed the baseline, yet it was surpassed by the projection method. To better understand these outcomes, we compared the learned features of both models through visualizations. The observed performance variations in the models could be due to several potential reasons. Firstly, the limitations of NeRFs in learning RGB and DINO features for moving objects, as indicated by the difference in feature capture ability between directly extracted DINO features from RGB images and those obtained through NeRFs. Secondly, the use of COLMAP for pose estimation might have resulted in errors, impacting the learning of accurate RGB and DINO features. Moreover, the optimization of some NeRF scenes could have been non-optimal, affecting the learning of sufficient DINO features, especially in complex scenes. Finally, the training dataset's focus on mostly static scenes likely limited the model's ability to adapt to dynamic or crowded environments, affecting its generalization capabilities.

Our research answers the proposed research question by demonstrating how the integration of NeRF and foundation models can effectively minimize the necessity of extensively labeled data in training state-of-the-art 3D LiDAR semantic segmentation models, thereby contributing a novel perspective to the field of autonomous driving technology.

6.2 Future Work

In this section, we outline potential future works for this thesis. These include the exploration of dynamic scenes, which could significantly enhance the use of the whole nuScenes dataset. Another aspect is experimenting with various 2D self-supervised feature extractors, potentially offering different representations. Additionally, investigating different architectures for volumetric space representation could lead to improvements in model performance. Lastly, instead of using per-point DINO features, utilizing per-cylindrical voxel features might offer another approach to feature extraction.

6.2.1 Incorporating Dynamic Scenes

Our work focuses on almost static scenes with minimum dynamic objects, but there is significant potential in expanding this to include dynamic scenes. Several existing techniques offer promising methods for this expansion. For instance, in projects such as Block-NeRF [65] and Panoptic NeRF [41], dynamic scenes have been successfully incorporated by employing masking strategies for moving objects using 3D bboxes. Block-NeRF approaches this by ignoring moving objects during training, a method

that may not align with our objectives. On the other hand, Panoptic NeRF adopts a more fitting approach for our needs, training separate MLP heads for moving objects and the static background, using masking techniques. This method could potentially allow our model to train on a broader range of scenes. However, it also introduces its own set of challenges, such as the need for a self-supervised mask extractor to extract per-dynamic object masks. While these techniques offer techniques to include dynamic scenes, they have not been explored within the scope of our thesis and present an interesting direction for future research.

6.2.2 Different Self-Supervised Feature Extractors

In the field of computer vision, with advancements in hardware technology, researchers are continuously developing new techniques for self-supervised feature extraction tasks. These models present opportunities for feature extraction. Exploring whether these new models can outperform the performance of the currently used model in our work is a promising area for future research. Improved feature extraction capabilities could directly enhance the quality of features derived from NeRFs. This approach has the potential to enhance the 3D semantic segmentation model by enabling it to learn high-quality features, therefore improving the overall performance of the segmentation process.

6.2.3 Different Architecture for Volumetric Space

Our approach centers the forward-moving ego vehicle on the z-axis within a cubic space. However, this configuration may not be the most efficient, particularly considering that our dataset mostly contains forward-moving scenes of 20-second durations. A possible development might be the development of a volumetric space tailored to forward-moving datasets. For instance, a rectangular volumetric space could potentially learn the entire scene within a single NeRF model while maintaining the same volumetric space. Such an architecture could offer a more efficient representation of the type of data we are working with, potentially enhancing the overall effectiveness of the model. Block NeRF addresses this challenge by training multiple overlapping NeRFs to represent large-scale environments. It decomposes a large scene into smaller segments, each represented by an individually trained NeRF, and then renders the entire scene by seamlessly integrating these overlapping NeRFs.

6.2.4 Using Per-Cylindrical Voxel Features

Our 3D model samples points from cylindrical voxels, concatenating them with corresponding point features, and then computing a per-point feature loss for the DINO feature head. An alternative strategy could involve calculating the center coordinates of each cylindrical voxel. For each center, we would then query the respective NeRF model to obtain DINO features. This process would shift the focus to calculating a

per-voxel loss for the DINO feature head, rather than a per-point loss. Since the mIoU and accuracy losses in our model are computed on a per-voxel basis, this approach aligns with these losses. By using a per-voxel strategy for DINO feature learning, there is potential to enhance the overall performance of the model.

Abbreviations

TUM Technical University of Munich

NeRF Neural Radiance Fields

CNN Convolutional Neural Network

ViT Vision Transformer

MLP Multilayer Perceptron

DDCM Dimension-Decomposition based Context Modeling

mIoU mean-intersection-over-union

SGD Stochastic Gradient Descent

IoU Intersection-over-Union

MSE Mean Squared Error

PCA Principal Component Analysis

SSC semantic scene sompletion

LiDAR Light Detection and Ranging

DINO DIstillation with NO labels

CLIP Contrastive Language-Image Pretraining

LERF Language Embedded Radiance Fields

List of Figures

List of Tables

Bibliography

[1] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun, "Towards fully autonomous driving: Systems and algorithms," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, 2011, pp. 163–168. DOI: 10.1109/IVS.2011.5940562.

[2] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020. DOI: 10.1109/ACCESS.2020.2983149.

[3] E. Ahmed, A. Saint, A. Shabayek, K. Cherenkova, R. Das, G. Gusev, D. Aouada, and B. Ottersten, *Deep learning advances on different 3d data representations: A survey*, Aug. 2018.

[4] S. Hao, Y. Zhou, and Y. Guo, "A brief survey on semantic segmentation with deep learning," *Neurocomputing*, vol. 406, pp. 302–321, 2020, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.11.118.

[5] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019. DOI: 10.1109/TNNLS.2018.2876865.

[6] 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 *CVPR*, 2020.

[7] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[8] 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 International Conf. on Computer Vision (ICCV)*, 2019.

[9] 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, pp. 3354–3361.

[10] L. Kong, J. Ren, L. Pan, and Z. Liu, "Lasermix for semi-supervised lidar semantic segmentation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 21 705–21 715.

[11] X. Yan, J. Gao, C. Zheng, C. Zheng, R. Zhang, S. Cui, and Z. Li, "2dpass: 2d priors assisted semantic segmentation on lidar point clouds," in *European Conference on Computer Vision*, Springer, 2022, pp. 677–695.

[12] K. Genova, X. Yin, A. Kundu, C. Pantofaru, F. Cole, A. Sud, B. Brewington, B. Shucker, and T. Funkhouser, "Learning 3d semantic segmentation with only 2d image supervision," in *2021 International Conference on 3D Vision (3DV)*, 2021, pp. 361–372. DOI: 10.1109/3DV53792.2021.00046.

[13] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *ECCV*, 2020.

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

[15] J. Kerr, C. M. Kim, K. Goldberg, A. Kanazawa, and M. Tancik, "Lerf: Language embedded radiance fields," in *International Conference on Computer Vision (ICCV)*, 2023.

[16] 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," *arXiv preprint arXiv:2011.10033*, 2020.

[17] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Gläser, F. Timm, W. Wiesbeck, and K. Dietmayer, "Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1341–1360, 2021. DOI: 10.1109/TITS.2020.2972974.

[18] M. Siam, M. Gamal, M. Abdel-Razek, S. Yogamani, M. Jagersand, and H. Zhang, "A comparative study of real-time semantic segmentation for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2018.

[19] H. Wang, Y. Chen, Y. Cai, L. Chen, Y. Li, M. A. Sotelo, and Z. Li, "Sfnet-n: An improved sfnet algorithm for semantic segmentation of low-light autonomous driving road scenes," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 11, pp. 21 405–21 417, 2022. DOI: 10.1109/TITS.2022.3177615.

[20] A. Milioto and C. Stachniss, "Bonnet: An open-source training and deployment framework for semantic segmentation in robotics using cnns," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 7094–7100. DOI: 10.1109/ICRA.2019.8793510.

[21] I. Alonso, L. Riazuelo, and A. C. Murillo, "Mininet: An efficient semantic segmentation convnet for real-time robotic applications," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1340–1347, 2020. DOI: 10.1109/TRO.2020.2974099.

[22] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., Cham: Springer International Publishing, 2015, pp. 234–241, ISBN: 978-3-319-24574-4.

[23] M. Z. Khan, M. K. Gajendran, Y. Lee, and M. A. Khan, "Deep neural architectures for medical image semantic segmentation: Review," *IEEE Access*, vol. 9, pp. 83 002–83 024, 2021. DOI: 10.1109/ACCESS.2021.3086530.

[24] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2015, pp. 3431–3440. DOI: 10.1109/CVPR.2015.7298965.

[25] 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," *ICLR*, 2021.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[27] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," *arXiv:2304.02643*, 2023.

[28] Y. Li, L. Ma, Z. Zhong, F. Liu, M. A. Chapman, D. Cao, and J. Li, "Deep learning for lidar point clouds in autonomous driving: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3412–3432, 2021. DOI: 10.1109/TNNLS.2020.3015992.

[29] X. Lai, Y. Chen, F. Lu, J. Liu, and J. Jia, "Spherical transformer for lidar-based 3d recognition," in *CVPR*, 2023.

[30] Y. Wang, T. Shi, P. Yun, L. Tai, and M. Liu, "Pointseg: Real-time semantic segmentation based on 3d lidar point cloud," *ArXiv*, vol. abs/1807.06288, 2018.

[31] A. Ando, S. Gidaris, A. Bursuc, G. Puy, A. Boulch, and R. Marlet, "Rangevit: Towards vision transformers for 3d semantic segmentation in autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2023, pp. 5240–5250.

[32] A. Jhaldiyal and N. Chaudhary, "Semantic segmentation of 3d lidar data using deep learning: A review of projection-based methods," *Applied Intelligence*, vol. 53, no. 6, pp. 6844–6855, 2023, ISSN: 1573-7497. DOI: 10.1007/s10489-022-03930-5.

[33] 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 *European Conference on Computer Vision*, 2020.

[34] Y. Hou, X. Zhu, Y. Ma, C. C. Loy, and Y. Li, "Point-to-voxel knowledge distillation for lidar semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2022, pp. 8479–8488.

[35] K. Park, U. Sinha, J. T. Barron, S. Bouaziz, D. B. Goldman, S. M. Seitz, and R. Martin-Brualla, "Nerfies: Deformable neural radiance fields," *ICCV*, 2021.

[36] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-NeRF: Neural Radiance Fields for Dynamic Scenes," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

[37] L. Yen-Chen, P. Florence, J. T. Barron, A. Rodriguez, P. Isola, and T.-Y. Lin, "Inerf: Inverting neural radiance fields for pose estimation," Dec. 2020.

[38] S. Zhi, T. Laidlow, S. Leutenegger, and A. Davison, "In-place scene labelling and understanding with implicit scene representation," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[39] S. Liu, X. Zhang, Z. Zhang, R. Zhang, J.-Y. Zhu, and B. Russell, "Editing conditional radiance fields," in *Proceedings of the International Conference on Computer Vision (ICCV)*, 2021.

[40] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Transactions on Graphics*, vol. 41, no. 4, pp. 1–15, Jul. 2022. DOI: 10.1145/3528223.3530127.

[41] A. Kundu, K. Genova, X. Yin, A. Fathi, C. Pantofaru, L. Guibas, A. Tagliasacchi, F. Dellaert, and T. Funkhouser, "Panoptic Neural Fields: A Semantic Object-Aware Neural Scene Representation," in *CVPR*, 2022.

[42] G. Hinton, J. Dean, and O. Vinyals, "Distilling the knowledge in a neural network," in *NeurIPS*, Mar. 2014, pp. 1–9.

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

[44] S. Kobayashi, E. Matsumoto, and V. Sitzmann, "Decomposing nerf for editing via feature field distillation," in *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[45] V. Tschernezki, I. Laina, D. Larlus, and A. Vedaldi, "Neural feature fusion fields: 3D distillation of self-supervised 2D image representations," in *Proceedings of the International Conference on 3D Vision (3DV)*, 2022.

[46] K. Blomqvist, L. Ott, J. J. Chung, and R. Siegwart, *Baking in the feature: Accelerating volumetric segmentation by rendering feature maps*, 2022. arXiv: 2209.12744 [cs.CV].

[47] J. Ye, N. Wang, and X. Wang, "Featurenerf: Learning generalizable nerfs by distilling pre-trained vision foundation models," *arXiv preprint arXiv:2303.12786*, 2023.

[48] Z. Fan, P. Wang, Y. Jiang, X. Gong, D. Xu, and Z. Wang, *Nerf-sos: Any-view self-supervised object segmentation on complex scenes*, 2022. DOI: 10.48550/ARXIV.2209.08776.

[49] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," in *ICML*, 2021.

[50] A. Hayler, F. Wimbauer, D. Muhle, C. Rupprecht, and D. Cremers, "S4c: Self-supervised semantic scene completion with neural fields," *arXiv preprint arXiv:2310.07522*, 2023.

[51] A.-Q. Cao and R. de Charette, "Scenerf: Self-supervised monocular 3d scene reconstruction with radiance fields," in *ICCV*, 2023.

[52] F. Wimbauer, N. Yang, C. Rupprecht, and D. Cremers, "Behind the scenes: Density fields for single view reconstruction," *arXiv preprint arXiv:2301.07668*, 2023.

[53] A. Yu, V. Ye, M. Tancik, and A. Kanazawa, "pixelNeRF: Neural radiance fields from one or few images," in *CVPR*, 2021.

[54] A. Trevithick and B. Yang, "Grf: Learning a general radiance field for 3d scene representation and rendering," in *arXiv:2010.04595*, 2020.

[55] S. Vora, N. Radwan, K. Greff, H. Meyer, K. Genova, M. S. M. Sajjadi, E. Pot, A. Tagliasacchi, and D. Duckworth, *Nesf: Neural semantic fields for generalizable semantic segmentation of 3d scenes*, 2021. arXiv: 2111.13260 [cs.CV].

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

[57] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791.

[58] J. T. Barron, B. Mildenhall, M. Tancik, P. Hedman, R. Martin-Brualla, and P. P. Srinivasan, "Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields," *ICCV*, 2021.

[59] M. Tancik, E. Weber, E. Ng, R. Li, B. Yi, J. Kerr, T. Wang, A. Kristoffersen, J. Austin, K. Salahi, A. Ahuja, D. McAllister, and A. Kanazawa, "Nerfstudio: A modular framework for neural radiance field development," in *ACM SIGGRAPH 2023 Conference Proceedings*, ser. SIGGRAPH '23, 2023.

[60] T. Müller, *tiny-cuda-nn*, version 1.7, Apr. 2021.

[61] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *arXiv preprint arXiv:1612.00593*, 2016.

[62] M. Berman, A. Rannen Triki, and M. B. Blaschko, "The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4413–4421.

[63] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[64] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixelwise view selection for unstructured multi-view stereo," in *European Conference on Computer Vision (ECCV)*, 2016.

[65] M. Tancik, V. Casser, X. Yan, S. Pradhan, B. Mildenhall, P. Srinivasan, J. T. Barron, and H. Kretzschmar, "Block-NeRF: Scalable large scene neural view synthesis," *arXiv*, 2022.

[66] O. Unal, D. Dai, and L. Van Gool, "Scribble-supervised lidar semantic segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2022, pp. 2697–2707.

[67] C. Schuhmann, R. Beaumont, R. Vencu, C. Gordon, R. Wightman, M. Cherti, T. Coombes, A. Katta, C. Mullis, M. Wortsman, P. Schramowski, S. Kundurthy, K. Crowson, L. Schmidt, R. Kaczmarczyk, and J. Jitsev, *Laion-5b: An open large-scale dataset for training next generation image-text models*, 2022. arXiv: 2210.08402 [cs.CV].

[68] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017, pp. 464–472. DOI: 10.1109/WACV.2017.58.