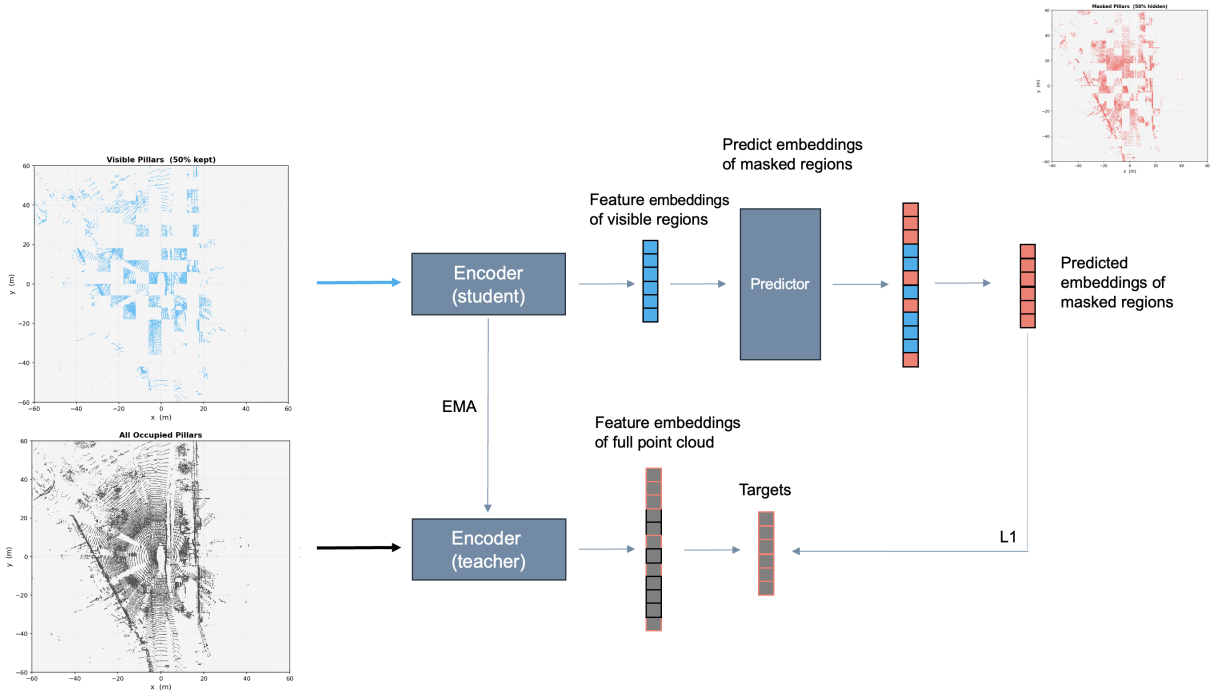
CHALMERS
UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG



Self-supervised Representation Learning for LiDAR Point Clouds

A Design Science Study of a Self-Supervised Model for Perception in Autonomous Driving

Master's Thesis in Computer science and engineering

Mariam Kronberg & Ylva Eriksson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's Thesis 2026

Self-supervised Representation Learning for LiDAR Point Clouds

A Design Science Study of a Self-Supervised Model for Perception in Autonomous Driving

Mariam Kronberg & Ylva Eriksson

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Self-supervised Representation Learning for LiDAR Point Clouds
Mariam Kronberg & Ylva Eriksson

Industrial Supervisors: Pauline Nässlander, Willem Verbeke & Luca Caltagirone, Zenseact
Academic Supervisor: Ali Nouri, Department of Computer Science and Engineering
Examiner: Gregory Gay, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Mariam Kronberg & Ylva Eriksson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Autonomous driving systems are large, complex software systems where accurate environmental perception is foundational to safe navigation. Perception systems often rely heavily on supervised deep learning models trained on large volumes of manually annotated datasets, resulting in performance heavily tied to the quality of the annotations. Self-supervised learning (SSL) offers a promising alternative by deriving supervisory signals directly from raw, unlabeled data, yet its application to LiDAR point clouds of autonomous driving data remains largely underexplored. We investigate whether a JEPA-based architecture, adapted to operate on LiDAR data, can learn high-quality representations without manual labels, and what that implies for the autonomous driving system during its evolution. We examine the meaning for software quality as well as the impact on the engineering process of building and updating the system. Through an iterative design process, we find that our SSL model substantially outperforms a fully supervised baseline in low label-budget regimes, and that fine-tuning the pre-trained backbone recovers nearly identical detection performance under the full label budget. Our results suggest that SSL pre-training is a viable architectural strategy for reducing annotation dependency and improving maintainability through backbone reuse, though these benefits come with meaningful upfront engineering complexity that should be weighed in practice.

Keywords: Self-supervised learning, LiDAR, Autonomous driving, Perception, Representation learning, Label efficiency, Design Science Research, Maintainability, Scalability, Robustness.

Acknowledgements

We would like to express our sincere gratitude to our supervisors Pauline Nässlander, Willem Verbeke and Luca Caltagirone at Zenseact for their support, guidance, and expert feedback throughout this thesis. The process has been both enriching and enjoyable, and we deeply appreciate your dedication.

A special thanks to Pauline Nässlander, who has been a constant source of support and guidance throughout the entire process, always taking the time to engage with our work and offer thoughtful feedback.

Lastly we want to thank our academic supervisor Ali Nouri and examiner Gregory Gay for their guidance and encouragement throughout the project.

Mariam Kronberg & Ylva Eriksson, Gothenburg, June 2026

Contents

List of Acronyms

AD Autonomous Driving. 1, 3, 4, 7, 16, 17, 22, 25, 28, 44, 84, 89
AP Average Precision. 40

BEV Bird Eye's View. 16, 19, 20, 22
BYOL Boostrap Your Own Latent. 18, 19

CCE Categorical Cross-Entropy. 25
CNN Convolutional Neural Network. 15, 16

DDP Distributed Data Parallel. 34
DL Deep Learning. 2
DSR Design Science Research. 2, 4, 29, 30, 33, 43, 48, 53, 79

EMA Exponential Moving Average. 18, 27, 55

FN False Negative. 40
FP False Positive. 40

GPU Graphical Processing Unit. 15
GT Ground Truth. 26, 37–40

I-JEPA Image-based Joint Embedding Predictive Architecture. 19, 20, 26–28, 47, 63, 89
IoU Intersection over Union. 39, 40

JEA Joint-Embedding Architecture. 13, 14
JEPA Joint-Embedding Predictive Architecture. 20, 46, 89

LiDAR Light Detection and Ranging. 1–5, 7, 10, 12, 14–16, 19–22, 24, 26–28, 32–34, 36, 37, 46, 57, 79, 82, 83, 89

MAE Mean Absolute Error. 40, 41
MAE Masked Autoencoder. 19
mAP Mean Average Precision. 40, 47, 49–51, 54–57, 59, 62, 65, 67, 70, 72, 73, 78–80, 85
MHA Multi-Head Attention. 24
ML Machine Learning. 9–11, 17, 20, 30, 86
MLP Multi-Layer Perceptron. 9, 21, 24, 45, 73

NLP Natural Language Processing. 2, 15, 23, 24

NMS Non-Maximum Suppression. 39, 54

OD Object Detection. 24, 49, 50
ODD operational design domain. 17

PCA Principal Component Analysis. 45, 46, 55, 73
PFE Pillar Feature Encoder. 21, 28, 49, 54

RE Requirements Engineering. 17
RGB Red, Green and Blue. 46

SCCE Sparse Categorical Cross-Entropy. 25
SoC Separation of Concern. 11, 83
SSL Self-supervised Learning. 2–4, 7, 11–14, 16, 17, 19, 20, 29–34, 48, 49, 55, 71, 74–79, 81–83, 85, 89

TP True Positive. 40

ViT Vision Transformer. xv, 15, 19, 23, 24, 26
VRAM Video Random Access Memory. 41, 49, 54, 77

ZOD Zenseact Open Dataset. 36

List of Figures

List of Tables

1

Introduction

Autonomous Driving (AD) has been a highly active research area for decades, and it has seen great technical progress over the last years. A central motivation for the development of self-driving capabilities in vehicles is the potential to prevent and reduce traffic accidents [5], by improving transportation safety and efficiency [1].

AD systems are large, complex software systems. A common way that these systems are composed is by grouping components into sensing, perception, planning, and control. Sensing retrieves raw data from sensors such as cameras, Light Detection and Ranging (LiDAR), and radar that scan the surrounding environment [5]. This data is passed to the perception system, which interprets the scene by performing tasks like object detection, tracking, and semantic segmentation [71], producing structured representations of the world. These feed into the planning component, where the system predicts the behavior of other agents, plans trajectories, and makes decisions that are finally translated into physical actions by the control component.

Of these components, perception is the foundation for all higher-level functionality. Its output directly determines the quality of planning and decision making, making it critical for the safety and robustness of the overall system [11]. Engineering a perception component that is accurate and scalable is therefore not merely a model engineering problem, it affects the entire software architecture.

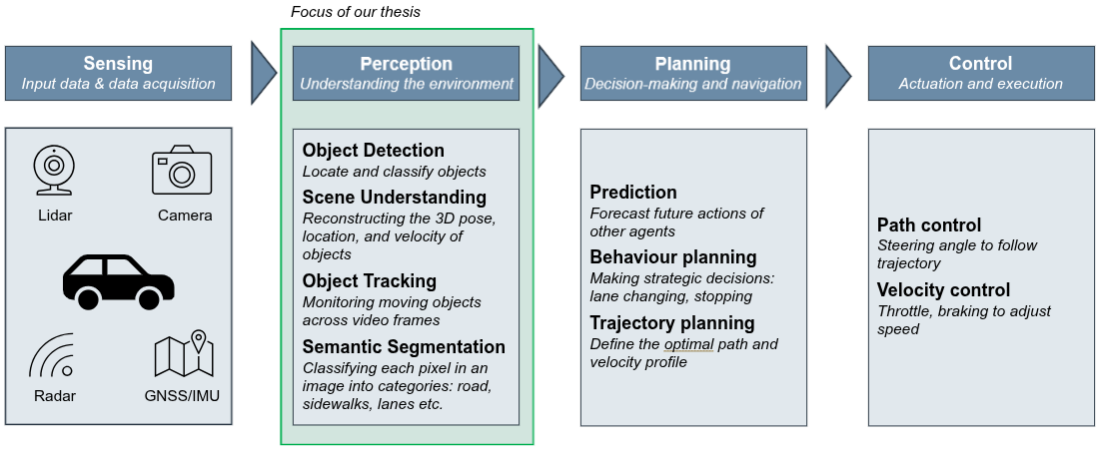


Figure 1.1: Conceptual overview of the AD-pipeline

Many current systems are built on Deep Learning (DL) models, which are complex networks that learn patterns in the environment by training on large-scale datasets. The main approach in current perception systems is supervised learning, where the models are trained on human-labeled datasets. Supervised models are constrained both by their dependency on quality labeled data as well as resource-intensive training and tuning [63]. Tight coupling between model performance and labeled data complicates both updating and scaling the models, as changes in requirements or domain shifts may require re-annotation and retraining [63].

Self-supervised Learning (SSL) offers a promising alternative to supervised learning by deriving supervisory signals directly from the raw data, reducing the need for manual labels. SSL has achieved remarkable results in Computer Vision and Natural Language Processing (NLP) [9], providing a possibility of leveraging large unlabeled datasets.

Fully supervised approaches require separately engineered models that each is tied to a specific downstream task. In contrast, the SSL-approach instead learns general semantic structures, producing representations that are task-agnostic[15]. That enables reusing the same model as underlying knowledge and train with just a small amount of labeled samples to adapt to a wide range of tasks, which simplifies scaling of the system. This decoupling also has direct implications on maintainability as the scope of changes required when requirements evolve is significantly reduced [80].

Different sensor data have different strengths and limitations to their suitability as training input. While camera-based perception has advanced significantly with deep learning and SSL specifically, its lack of explicit depth information limits accurate spatial understanding required for robust navigation. LiDAR provides rich spatial information that enables more precise distance estimation and scene understanding. However, processing LiDAR point clouds is challenging due to their sparsity, irregular distribution, and unstructured nature. These characteristics complicate feature extraction and model training, introducing different challenges for 3D detection than for 2D image perception [2]. Furthermore, annotating LiDAR data is particularly time consuming and difficult, not only due to the complexity of annotation taxonomy but also due to the sparsity and difficulty in interpreting LiDAR point clouds [75].

This thesis investigates SSL for LiDAR-based perception as a software design alternative to traditional supervised approaches. The study adopts a Design Science Research (DSR)-approach to implement a novel SSL model trained exclusively on LiDAR data from autonomous driving scenes. Through iterative development and evaluation, we generate insights about different architectural design choices and their effect on performance and compute efficiency.

Our SSL-model is compared to a supervised baseline model trained fully on annotated samples, to assess the potential to reduce reliance on annotated data while increasing performance. To compare the performance, 3D object detection is used as a downstream task for assessment. Through this comparison, we investigate how

leveraging SSL as a pretraining strategy can improve scalability, maintainability and robustness in the overall system.

1.1 Problem Description

Perception plays a significant role in autonomous driving systems, as many higher-level tasks rely on its output. Although LiDAR-based perception methods have improved significantly in recent years, most state-of-the-art approaches still rely heavily on supervised approaches, creating a tension between achieving model performance sufficient for deployment, and the practical limits of human annotation.

Producing high-quality labeled data is especially difficult for AD due to the high complexity of driving scenes, including large variation in objects and geographic diversity [9, 83], and annotation is costly and often inconsistent. The same scenario might be labeled differently across annotators or geographies, degrading the reliability of the training data [78]. As requirements evolve, models must be re-annotated and retrained from scratch, and in systems where multiple models share the same data, a single requirement change can cascade across the entire pipeline.

Scaling these systems requires addressing two challenges. Firstly, as datasets grow, managing storage, allocating resources, and maintaining training efficiency all become increasingly costly. Secondly, model performance is tightly coupled to the quantity and quality of labeled data, and in supervised settings this coupling is explicit and difficult to break [63].

The task-specific structure of supervised models introduces further fragility. Each model is independently maintained and tightly coupled to its own annotation schema, leading to significantly increased engineering overhead with every new perception capability added to the system. These structural properties make the pipeline difficult to scale and costly to maintain over time.

Even models trained on large well-annotated datasets can degrade significantly when exposed to conditions underrepresented in training, such as geographic domain shifts, making robustness another persistent concern [10]. An important requirement in AD is that the system must perform reliably even under rare conditions, and finding models that generalize well and are robust to changes both during training and in action is a concern.

SSL offers an architectural alternative that addresses both the annotation dependency and problems with task-specific architectures. By learning from raw unlabeled data, it removes the direct dependency on label-defined tasks and enables a shared model that can be reused [80], which generally reduces the number of annotations needed and improves generalization. However, realizing this in practice for LiDAR-based perception is a challenge. Existing SSL methods have demonstrated promising results, but their application to outdoor point cloud data remains limited, as the unstructured and incomplete nature of LiDAR makes it difficult to design an SSL task that lets the model learn meaningful representations [75, 79]. How to navigate

these challenges and their engineering trade-offs for the broader perception system remains a relevant area for exploration.

Taken together, these points highlight practical limitations of fully supervised learning for LiDAR-based perception in autonomous driving, which are annotation-intensive, costly to adapt as requirements evolve, and may struggle to generalize beyond the conditions seen during training. This motivates the exploration of alternative learning paradigms that reduce dependence on large quantities of high-quality annotations, while remaining adaptable and robust across diverse operating conditions.

1.2 Purpose of the Study

The purpose of this study is to design and evaluate an SSL-model trained on LiDAR sensor data.

We investigate a novel combination of techniques in SSL that has not yet been applied to LiDAR. The study follows the DSR-paradigm, which allows us to iteratively develop, evaluate and redesign the model. Through this we generate design knowledge about how architectural choices such as model task design, capacity and architecture affect downstream perception performance. Our aim is to bridge a gap in the knowledge about how to implement AD-specific SSL for LiDAR and investigate how SSL-based LiDAR perception affects the software engineering processes of the AD system.

To further specify the focus of the study, the following research questions are formulated:

RQ1 - How do software engineering design decisions in a self-supervised LiDAR perception model influence the quality of learned representations in terms of downstream perception performance? And how can these effects be interpreted in terms of robustness, maintainability, and scalability of the perception component?

With this research question we aim to investigate the implications of the software design choices and how they affect performance.

RQ2 - How do these engineering decisions influence training efficiency (compute cost, memory footprint etc.), and which components of the SSL pipeline constitute the main computational bottlenecks?

This is aimed to evaluate quantitative aspects that affects the process of developing the component, and focuses on the non-human resources needed.

RQ3 - How will an SSL-pretrained object detection LiDAR model compare to a fully supervised baseline in object detection accuracy when trained on a limited label budget?

With this RQ we aim to investigate to what extent we can reduce label dependency by self-supervised pretraining. The goal is to achieve better performance per used annotated sample.

1.3 Significance of the Study

This study takes an interdisciplinary approach aiming to bridge the gap between software engineering and deep learning in the domain of autonomous driving. A self-supervised learning model for LiDAR-based perception is designed and developed, representing a novel combination of architectural and engineering design choices not previously explored in this context. In doing so, the study aims to generate design knowledge about how these engineering choices influence performance. By treating self-supervised learning as an architectural strategy, we also map out the connections to different engineering trade-offs in the system related to data-dependency, training complexity and maintenance. The findings are intended to be relevant both to researchers exploring alternative learning paradigms for LiDAR-based perception, and to practitioners seeking more adaptable and sustainable solutions for real-world autonomous driving systems.

1.3.1 Thesis Outline

The table below presents the structure of the thesis:

Chapter	Content
Chapter 2 Background	Presents relevant concepts that the thesis is based on
Chapter 3 Related Work	Introduces work adjacent to the focus of this thesis and presents the research gap in which this work sits
Chapter 4 Approach	Gives an overview and thorough description of the components used to build the software artifact, and describes how the components are composed in the software artifact.
Chapter 5 Methods	Introduces the methodology, iterative development process, and design of the experiments.
Chapter 6 Results	Presents insights from the design process as well as the results of the experiments.
Chapter 7 Discussion	Discussion of results and their implications for the perception system.
Chapter 8 Conclusion	Presents conclusions of the thesis.

2

Background

This chapter provides an overview of relevant topics that this thesis builds upon, starting with an introduction to perception systems and the software engineering challenges of supervised learning, and how SSL is proposed as a strategy. Following that is an introduction to SSL and how to process LiDAR-data.

2.1 AD Perception Systems

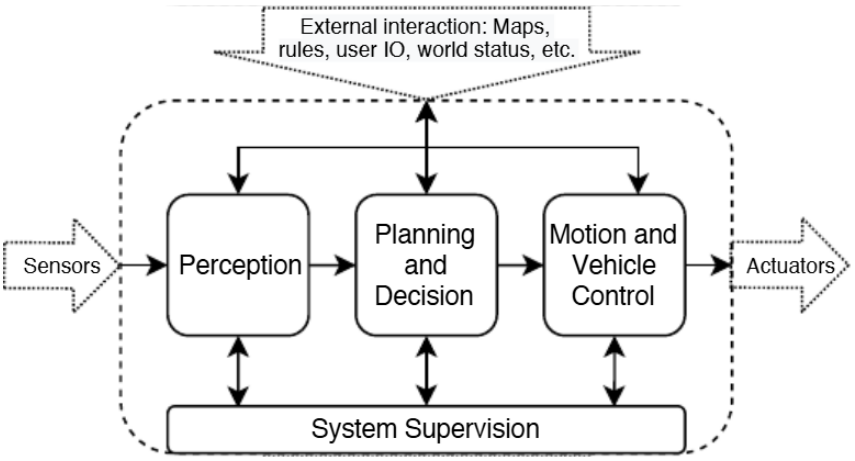


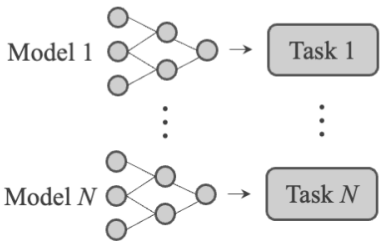
Figure 2.1: Functional view of an autonomous driving system. [71]

An overview of a common AD-system is illustrated in Figure 2.1. The perception component can be described as the visual cortex of the vehicle that is responsible for converting raw sensory input into meaningful representations of the world. This can be done through performing intermediate tasks, that acts as a base for the downstream planning and control components, with the end goal of executing appropriate actions for the vehicle [71]. Examples of common perception tasks are:

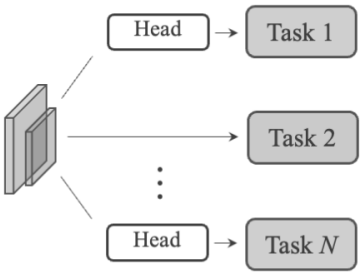
- Object Detection locates and predicts properties like size and rotation of surrounding entities.
- Semantic Segmentation divides the terrain into regions such as lanes, buildings or free space.

- Classification identifies the different types of objects such as pedestrians, vehicles, traffic signs and animals.
- Depth Completion focuses on estimating the distance of the pixels or points to the viewer.

Traditionally, separate, supervised models have been trained for each task as shown in Fig.2.2(a). Since each of these models require their own data pipeline and training infrastructure, the result is a complex system with many independently maintained components, each tightly coupled to its own annotation schema [73]. Updating one model to reflect new requirements does not automatically update others, creating a risk of inconsistencies across the pipeline. Every new perception capability requires building and maintaining a new model from scratch, multiplying annotation cost and engineering overhead by the number of tasks.



(a) Standalone Models



(b) Multi-task Framework

Figure 2.2: Illustration of task-specific models versus a shared backbone architecture [31].

Current research is therefore shifting towards all-purpose models that learn general representations, commonly referred to as Foundation Models [60, 8, 41, 13]. Typically they are trained using Self-Supervised or Unsupervised Learning, that leverages non-labeled data to learn general knowledge about the world. This information can then be used to adapt the model to a wide range of downstream tasks, while reusing the same backbone model. A method to do this is to train lightweight task-specific modules on top of the backbone, as illustrated in Figure 2.2(b).

The purpose of the task heads is to map the output of the backbone model to a

label-specific task, such as locating and classifying objects or tracking other agents. The main intelligence is in the backbone, while the task head acts as a lightweight converter, mapping the backbone's output to label-specific targets. For example, if object detection is the task, then an Multi-Layer Perceptron (MLP) can be used as a task head, which is just a few linear layers with a non-linear activation, mapping the output from the backbone to specific label-based targets like size and orientation of annotated object bounding boxes.

2.2 Software Engineering Challenges

As the perception stack becomes increasingly dominated by these large models, new categories of software engineering challenges become apparent that go beyond traditional software development. This area has gained increased awareness in the last few years following recent rapid evolution of large Machine Learning (ML)-enabled systems [65, 63, 61, 55, 64]. Recent studies emphasize that they introduce inherent scalability and maintainability challenges that are difficult to manage using only conventional software engineering practices [63, 64].

2.2.1 Software Quality Metrics

We start by giving a brief overview of how we refer to various software quality metrics throughout the thesis, and how they are relevant in the light of ML-enabled systems.

Scalability refers to the software systems' ability to maintain performance under increasing workloads, i.e., that the system should not experience lower efficiency or effectiveness when the workload increases. However, when talking about ML systems, scalability rather refers to the ML systems training efficiency when the dataset grows including managing storage requirements and allocating computational resources, without degrading in performance. Furthermore, scalability in ML is highly connected to the quality and quantity of training data as larger models require larger datasets [63].

Maintainability refers to the effort required to modify, extend, or update the perception component over time. It is a crucial aspect of any software system as it must be able to evolve to stay relevant. Maintainability is typically achieved by designing artifacts that are modular, testable, and easy to understand. *Modularity* is a sub-characteristic of maintainability, which is the degree to which the system is composed of components that can be changed without affecting other components[33]. In the context of ML there are additional challenges beyond code that affect maintainability as the models need to handle the evolving nature of data as well. Therefore, maintainability in ML-based systems also highlights the ability to effectively manage changes in the data requirements, and updating and extending the system in response to it [63, 64].

Robustness in general terms refers to a systems ability to maintain stable and

reliable performance across different conditions or challenges, i.e., demonstrate resilience when uncertainties or unexpected changes occur [10]. In software systems, it means for example being able to consistently handle edge cases. In the context of ML-models, a broad definition is that a robust system can effectively handle unforeseen conditions during its evolution [6], meaning it maintains stable predictive performance despite varying conditions, domain shifts or variations in the input data. In the broader system context, robustness of the ML-model affects the robustness of the whole system, as failures of the model can produce unpredictable behavior that propagates through to downstream software components, making it a system-level concern [77, 10]. Beyond handling variations in input data, robustness of ML-systems also include the models capacity to maintain stable performance under variations in the training algorithm or hyper-parameters [21].

2.2.2 The Problem of Data Dependency

The most apparent consequence of ML based systems in comparison to regular software systems is that data becomes a significant software dependency. Unlike conventional software, where behavior is determined by code, ML-models' behavior is largely determined by the data it was trained on [30], and the system becomes brittle in response to changes or even improvements to this data, as it can have unpredictable consequences for dependent components in the system [61]. Data-related technical debt in ML-systems can be harder to detect than code-level dependencies, since static analysis is not sufficient to identify these dependencies [61].

2.2.2.1 Annotation Engineering Bottleneck

Annotation dependency is a particularly challenging kind of data dependency, that is generally viewed as a major engineering challenge and developer bottleneck [50, 51, 30, 25, 68].

Firstly, defining the data requirements, i.e. what should be labeled and how, is not a trivial task. Within the context of AD-systems, complexities arise due to driving scenarios being extremely diverse depending on the geographical location. City, highway, or nature settings differ between countries, and weather conditions, object types and variety in the scene can be widely diverse both by location and time. Classes and subclasses need to be defined and prioritized, but no matter what level of detail is chosen, ambiguities, inconsistencies and rare scenarios will arise [58, 51]. Adding to the challenge is domain shifts, where for example weather conditions differ from the training data [20, 40], or data distribution shifts where new kinds of objects are introduced in traffic scenes, and the car needs to be able to process and react to all these variations in a safe way.

Due to this complexity, annotation requirements are difficult to define. Once they are defined, producing consistent, good quality labels is extremely slow and costly [7, 45, 68]. Especially LiDAR-point clouds are notoriously difficult to annotate due to their sparse nature and lack of color information [75].

Furthermore, annotation taxonomies are prone to changes [25, 78]. Apart from the process of having to re-annotate, there is another engineering problem that arises due to this, which is that models that are dependent on these annotations must be re-trained to incorporate the changes. In a system where multiple models depend on the same dataset, a small change can trigger a need for re-training across the entire pipeline. This coupling between model behavior and labels limits the ability to incrementally evolve the system.

2.2.3 pretraining as SE Strategy

Self-supervised pretraining pipelines are often quite complex, and come with their own challenges that require good software engineering practices [54], but the architectural properties of shared backbone designs make SSL an interesting strategy for managing long-term engineering costs of perception systems.

2.2.3.1 Architectural Benefits

There is an architectural motivation behind leveraging pretraining strategies like SSL to train a general-purpose backbone. By utilizing a design that separates general-purpose feature learning from task-specific inference, we take a key software engineering principle into account, which is Separation of Concern (SoC). SoC is the idea that individual components should have well defined responsibilities and be interchangeable without affecting the rest of the system [73]. This is difficult to uphold in fully supervised ML systems, where feature learning and task inference are entangled within a single model, and changing the task requires retraining the model from scratch.

This utilization of SSL can also be defined as increased model modularity [66]. An architectural boundary is introduced between the backbone and label-dependent task heads, separating them to distinct sub-entities. The two can be developed, updated and replaced independently.

Since the intelligence in the backbone is general and not connected to a label-specific task, this promotes potential reusability of the backbone, where adding a new perception capability can be as simple as adding a new task head, not completely rebuilding the pipeline [60].

A consequence of this architectural pattern is that limitations of the pre-trained backbone can propagate throughout the system. However, improvements to the backbone could similarly benefit all downstream tasks simultaneously. Updating or improving the core representation layer therefore has system-wide effect, rather than requiring maintenance for each task-specific model. Knowledge updates and troubleshooting can be more targeted, if different kind of knowledge is separated into different modules.

2.2.3.2 Label Dependency Reduction

Although this approach still requires annotated samples to adapt to the downstream tasks, a further advantage with the pretraining paradigm is that this step usually requires less labeled data than fully supervised approaches, and can outperform supervised approaches, especially when only small amounts of labels are available [80]. It may also improve robustness by generalizing better across domains, since it is not as optimized towards a specific downstream objective [77]. This has a direct positive impact on the maintainability of the system through avoiding built-in dependency from human labels in the backbone, and reducing the overall need for annotation processes and maintenance connected to label dependency.

2.2.4 Model Training Efficiency

Training efficiency is a well established concern in deep learning. Computational cost like training time and memory consumption is known to scale with larger models and larger dataset sizes [35], which means that the training process becomes slower and more time- and resource-consuming as models and datasets are scaled, affecting costs and hindering efficient development and maintenance [43].

From a software engineering perspective, the efficiency concern is not the cost of a single training run, but the cumulative training and retraining cost over the lifetime of the system. Minimizing training dataset size and model complexity therefore serves both a resource efficiency goal and an engineering goal as it reduces the time and effort associated with modifying and updating the system over time. We therefore find it relevant to take into consideration when evaluating the effects of the architectural shift that we propose in this study.

2.3 Background on SSL and LiDAR

In this section we aim to give an overview of how SSL works in general, and commonly associated challenges. Then we present the attributes of point cloud data, in particular LiDAR, and how a neural network can learn from it in practice.

2.3.1 Self-Supervised Learning

In SSL, instead of relying on annotated samples, the supervisory signal is derived from the raw unlabeled data by solving a "pretext task". These pretext tasks exploit the inherent structure of the data to learn useful features, for instance by reconstructing masked regions or enforcing consistency between augmented views of the same scene.

The primary objective of this is to construct a general representation of the world. Rather than learning task-specific features, the model learns semantic embeddings that captures universal concepts like depth, object boundaries, and spatial coherence. The learned representations can then be transferred to downstream tasks

such as object detection or semantic segmentation, as shown in Figure 2.3. This enables the modular software stack where the representations form a general backbone component that can be leveraged for more than a single task [60, 80].

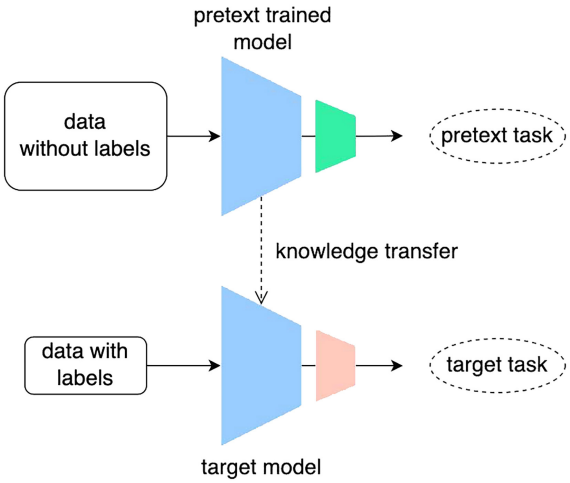


Figure 2.3: A visual abstraction of how learned information is transferred from a self-supervised model to a downstream task [68]

.

There are two leading learning approaches in the field of SSL: Joint-Embedding Architecture (JEA)s and reconstruction-based.

Reconstruction, also known as generative, is a paradigm where the goal is to predict and reconstruct parts of the input that have been masked or corrupted. This approach trains the model by penalizing errors between the reconstructed and original input. This means the model is rewarded for accurately reproducing the raw input, rather than for understanding its semantic content. As a result, the model is driven to capture whatever varies most in the data, since that is where reconstruction errors are largest. This is not inherently problematic, but whether the most variable features are also the most semantically meaningful depends on the modality. In language, this alignment happens naturally as tokens are compact entities that already encode meaning, so predicting a missing word requires the model to reason about context and semantics rather than low-level patterns. In visual domains however, the highest variance often lies in low-level details such as texture and lighting, which are statistically dominant but carry little semantic meaning. Consequently, pixel-level reconstruction objectives tend to work in favor of reproducing surface-level patterns rather than learning the higher-order structure that builds scene understanding [70].

JEA approaches operate in embedding space rather than the raw input space. The pretext task is to produce similar embeddings for different augmented views of the same input [70]. The main challenge with these approaches is representation collapse, which occurs when the representations produced converge to a constant value regardless of the input [4]. **Contrastive methods** are a branch of JEAs that en-

ables the model to learn semantic features and mitigating the risk of representation collapse by using negative (dissimilar) pairs of inputs that are explicitly repelled from the positive (similar) pairs, while positive pairs are pulled together. By requiring the model to distinguish between inputs, the method avoids collapse since producing constant representations would result in high loss for all negative pairs [34] However, these methods are constrained by the sampling negatives as it's difficult to ensure that they are neither too hard nor too easy to distinguish [59].

Non-contrastive methods are another branch of JEAs that learns representations and prevents collapse without using negative samples but instead introduces architectural constraints such as self-distillation where a model learns from another, also known as teacher-student framework. The student model is trained to replicate the outputs or internal behaviors of the teacher. The information transferred from the teacher can take several forms, such as predicted probability distributions or intermediate feature representations. Regardless of the specific form, the objective is to minimize the difference between the student and teacher representations [60]. Other common techniques to prevent representational collapse is stop gradient, exponential moving average and asymmetric architectures [59].

While these methods are introduced to avoid the issue of representation collapse, SSL-methods still experience **Dimensional Collapse**. Dimensional collapse is a state in which the embeddings occupy only a subspace of the total dimensional space. This failure mode has been linked to decreased performance on the downstream task, as the representations can become too uninformative to capture the necessary features to solve the task [59]. Furthermore, dimensional collapse can allow the model to satisfy the pretraining objective without learning meaningful features. This is because low dimensional representations are easier to predict, but carry less information about the underlying structure [39].

2.3.2 LiDAR Data and Processing Architectures

2.3.2.1 Attributes of LiDAR Data

LiDAR (Light Detection and Ranging) is a sensing technology that measures distance by emitting laser pulses and recording the time until they return. This produces a point cloud, a sparse, unordered set of 3D coordinates representing the geometry of the surrounding environment. Each point also carries additional information such as intensity, which measures the magnitude of the reflected signal, providing complementary surface information alongside the distance from the vehicle. LiDAR sensors can cover ranges of hundreds of meters, producing hundreds of thousands of points per scan [56].

The different sensors of the vehicle have different strengths and weaknesses. LiDAR has shown strong performance in tasks like object detection due to its precise depth estimation, 3D spatial structure and robustness to varying weather and light conditions, and is considered to play an important role in robust 3D-prediction tasks [60].

While LiDAR is considered well-suited for perception tasks, its raw output is challenging to process compared to 2D images or indoor 3D point clouds. The main difficulty is connected to the unstructured nature of the point cloud and that, unlike images, it does not contain information everywhere in the scene. This creates a mismatch with many dense, grid-like architectures often used in deep learning since the large empty spaces need to be handled somehow. Furthermore, this unstructured nature of the data complicates efficient utilization of Graphical Processing Unit (GPU) architectures, which are inherently optimized for the clean parallelism and contiguous memory access of dense 2D grids [74].

Furthermore, outdoor LiDAR from driving scenes can cover hundreds of meters, but the density of the points can be very low far from the sensor, which complicates interpretation of distant objects. The large spatial scale also poses additional challenges regarding memory and computational efficiency, since the number of points can be very large [76, 67].

An inherent difference between images and point clouds is that a pixel contains information about its colors mapped to a fixed position, while a LiDAR points's primary information is its position. This needs to be taken into account when trying to construct a model that should learn something else than purely predicting trivial aspects like distance from the sensor.

2.3.2.2 Common Applicable Network Architectures

To process the structural information inherent in LiDAR data, two dominant architectural paradigms are typically employed.

Convolutional Neural Network (CNN): Common neural network architecture for processing data with fixed, grid-like structures like images. CNNs consist of stacked layers that apply small learned filters across the input, where each filter produces a feature map. By stacking multiple such layers, CNNs build up increasingly abstract representations [37]. While CNN-based networks have shown great success in object detection and recognition, they focus largely on local features, limiting their ability to capture long-range dependencies across the input [24].

Transformer: Transformers were originally introduced in NLP and have since been adapted to computer vision through the introduction of the ViT. The transformer architecture uses a attention mechanism that allows the model to learn relationships between all parts of the input, regardless of their distance from one another. This directly addresses the CNNs limitation of local receptive fields, making transformers better suited for tasks requiring a broader understanding of the visual context. In ViT, images are divided into fixed-size patches that are linearly embedded and processed as a sequence, allowing the attention mechanism to model dependencies across the entire image [19].

2.3.2.3 Deriving Features from LiDAR Data

There are many different approaches for processing the point cloud into features suitable for input to a neural network. Previous works have proposed a variety of methods for 3D feature extraction to address some of the challenges with LiDAR, including point-based, voxel- and pillar-based methods [67].

Point-based methods are straightforward in their approach. They process the set of points directly without restructuring the data into an intermediate representation. The model learns representations by first processing each point individually and independently, then aggregating information from neighboring points, and finally combining these into a global representation [52, 53, 62]. However, training these kind of architectures on point clouds containing hundreds of thousands of points is very computationally heavy [82], and therefore not manageable for models with sufficient expressivity to solve the complex tasks that are of interest in AD.

Voxel-based methods divide the 3D space into voxels, a regular grid of cuboids. Each voxel processes information about the points that fall inside it, typically features like mean coordinates of points, point density and intensity statistics, forming neural feature vectors [82, 17]. While this approach provides strong representations, the computational complexity scales cubically with the resolution of the voxel grid.

Pillar-based methods are alternatives to voxel-based approaches that offer a structural optimization by dividing the 3D space into vertical columns instead of cuboids, the scene can then be represented as a 2D image in the Bird Eye's View (BEV) plane. This approach reduces the spatial scaling cost from cubic to quadratic. This drastically lowers the compute time and memory cost while largely preserving the spatial relationships that can be used in network architectures like Transformers or CNNs [38].

2.3.3 The Challenge of Applying SSL to LiDAR

Together, this outlines the core challenges that arise when applying SSL to outdoor LiDAR perception. From the SSL side, the central difficulty is designing a pretext task that drives the model toward semantic representations rather than superficial patterns, and avoiding representational collapse while doing so. From a data perspective, the sparse and unstructured nature, computational demands, and dominance of positional information in outdoor LiDAR point clouds all affect which architectures and learning objectives are practically applicable[75].

3

Related Work

This chapter reviews related work across four areas: software engineering for perception system development, SSL in 2D vision, SSL in 3D, and a concluding discussion of the research gap this work addresses. The breadth of the covered literature reflects the interdisciplinary nature of the thesis, which spans both the engineering processes surrounding perception system development and the underlying machine learning methods that enable it.

3.1 SE for Perception

Recent research investigates the Requirements Engineering (RE) aspect of perception in AD-systems. Habibullah et al. [25] present an interview study on requirements engineering for automotive perception systems. Their work finds that practitioners encounter challenges related to operational design domain (ODD), scenarios and edge cases, requirements breakdown, traceability and requirements specifications. Heyn et al. [30] build on this but focusing on RE challenges for data and annotations specifically, as well as topics regarding actors' ecosystems and business models' ability to handle data-intensive development of ML-based perception. One of their main findings is that while requirements specifications seem to be of major significance in automotive sourcing and certification processes, there's no standardized approach for specifying data or data annotations. This gap can be explained by the lack of consistent measurements of data variation, unclear data collection processes, vague definitions of annotation quality and lack of transparency in the annotation processes.

Saeeda et al. [58] further investigates requirements specifically for data annotations, how they are defined and used in practice, and challenges associated with their quality and impact on AI-enabled perception systems. Through interviews with 19 practitioners, they identify the major challenges to be ambiguity in annotation guidelines, edge case complexity, requirements evolving over time, inconsistencies in annotation requirements and resource constraints that force quality trade-offs throughout the annotation pipeline. The paper also indicates how annotation requirement quality has negative downstream effects on annotated data quality and ultimately on perception system performance. The authors recommend embedding structured quality assurance mechanisms such as multi-stage reviews, iterative feed-

back loops, and cross-functional collaboration into annotation workflows, as well as aligning annotation requirements with regulatory frameworks such as ISO 26262 and the EU AI Act. Complementing this, [57] present a taxonomy of 18 recurring data annotation errors in ML-based perception system development, organized along three data quality dimensions: completeness, accuracy, and consistency. Derived from a multi-organizational case study across the automotive supply chain, the taxonomy identifies how errors such as missing feedback loops, insufficient annotation guidance, and inter-annotator disagreement propagate through the development pipeline and degrade perception model reliability. Industry experts validated the taxonomy as a practical diagnostic tool, comparing it to FMEA (Failure Mode and Effects Analysis), and highlighted its utility for supplier quality reviews and root-cause analysis.

Together, these studies highlight where the software engineering efforts have been directed in the context of perception systems.

3.2 SSL in 2D vision

Before exploring SSL in the context of 3D LiDAR point clouds, it is useful to examine how these methods first emerged in 2D vision. The field has evolved around three main paradigms: contrastive, non-contrastive, and generative, each offering a distinct approach to learning representations without labeled data. The following section presents key methods, namely BYOL, SimSiam, MAE, and I-JEPA.

Boostrap Your Own Latent (BYOL), introduced by Grill et al. [23], is a non-contrastive image representation learning approach built on two neural networks that interact and learn from each other, an online network initialized with a set of weights and a target network with a different set of weights. The online network processes an augmented view of an image and is trained to predict the representation of another differently augmented view of the same image produced by the target network. Stop gradient is applied to the target network to prevent gradient flow during back propagation, instead the parameters of the target network are updated as a slow-moving average of the online network parameters, producing a slowly evolving target representation. In contrast to many earlier contrastive learning methods such as SimCLR [14] and MoCo [27], BYOL does not rely on negative pairs to prevent collapse, as the architectural asymmetry between the online and target networks, enforced by the stop gradient and slow Exponential Moving Average (EMA), prevents the model from producing meaningless representations. Instead, the model learns useful representations by encouraging consistency between representations of different augmented views of the same image [23]. SimSiam adapts the BYOL paradigm but collapses the online and target network into one single encoder that processes both views sequentially creating two data paths, and a predictor is applied to one of the encoded outputs. Stop gradient is applied to one of the paths to prevent representation collapse, and the encoder parameters get updated by gradients flowing through the active path [16].

In parallel, a separate line of work frames SSL as a reconstruction problem. Masked Autoencoder (MAE) learns representations through masked image reconstruction. A masked autoencoder reconstructs the missing parts of an image from the latent representations of the image's visible patches. The model consists of an encoder-decoder architecture, where the encoder is a ViT applied only to the visible patches of the image. The decoder receives the encoded visible patches together with mask tokens (shared learned vectors indicating the locations of missing patches) and processes them through a series of Transformer blocks to reconstruct the original image in pixel space. The decoder is used only during pre-training for the reconstruction task, while the encoder is used to produce image representations [26].

A key limitation of pixel-space reconstruction approaches like MAE is that they optimize for low-level visual detail rather than semantic content. Image-based Joint Embedding Predictive Architecture (I-JEPA) [4] addresses this by shifting the prediction target from pixel space to representation space. It uses a similar setup as BYOL with an online and a target network. However, the learning objective is formulated as predicting representations of masked regions from the surrounding image context rather than enforcing consistency between two augmented views of the same image. The architecture consists of three components: a context encoder, a target encoder, and a predictor, all implemented using ViT. The context encoder processes the visible (unmasked) regions of the image and produces patch-level representations of the context. The target encoder processes the full image and produces representations for all image patches, from which representations corresponding to selected target blocks are extracted [4]. By avoiding pixel-space reconstruction, I-JEPA encourages the model to capture more abstract, semantic structures.

3.3 SSL for 3D Point Clouds

Most SSL methods for LiDAR point clouds are based on frameworks originally developed for 2D vision, adapted to handle the irregular and sparse nature of 3D data. Mirroring the evolution in 2D, early 3D SSL methods framed the pretext task as masked reconstruction in input space, while more recent work has shifted toward prediction in representation space.

Occupancy-MAE [44] adapts the masked auto-encoding idea to voxelized LiDAR data. The input point cloud is first converted into a voxel grid, where the 3D space is divided into volumetric cells. During pretraining, a subset of voxels is masked, and the remaining voxels are processed by an encoder using sparse 3D convolutions. While related approaches such as Point-MAE [48] and Voxel-MAE [29] apply masked reconstruction directly to point coordinates or voxel features, Occupancy-MAE instead predicts voxel occupancy, i.e., whether a voxel contains points or not and reconstructs the geometric occupancy of the general 3D world. Occupancy-MAE employs a range-aware masking strategy that applies different masking ratios depending on the distance from the sensor to better account for the varying density of LiDAR point clouds [44]. BEV-MAE [81] takes a different approach to the same reconstruction objective, projecting the point cloud onto a BEV grid and apply-

ing masked modeling directly in BEV feature space. The model reconstructs points from masked BEV representations and additionally predicts point density. Together, these methods demonstrate that masked reconstruction can be effectively transferred to 3D, but like their 2D counterparts, they optimize for input-level reconstruction rather than semantic representations.

AD-L-JEPA [83] addresses this limitation by bringing the Joint-Embedding Predictive Architecture (JEPA) framework to LiDAR data. Rather than reconstructing masked regions in input space, the model predicts the representations in embedding space, following the same principle as I-JEPA in 2D. The architecture follows the I-JEPA setup with a context encoder, a target encoder, and a lightweight predictor network. However, instead of a Vision Transformer backbone, the encoder is implemented using sparse 3D convolutions to process point cloud data. The resulting 3D features are reshaped into BEV embeddings, from which a lightweight convolutional predictor estimates the target embeddings for masked regions. By operating in representation space, AD-L-JEPA inherits the same semantic focus that motivates I-JEPA in 2D, making it the most directly relevant prior work to the method developed in this thesis.

3.4 Research gap

While prior SE research has identified challenges and recommended solutions in specifying and managing data and annotation requirements for ML-based perception systems, less attention has been paid to how the choice of learning strategy can itself help to mitigate these challenges. Existing SSL methods for LiDAR data, such as Occupancy-MAE, BEV-MAE, and AD-L-JEPA, demonstrate that strong representations can be learned without labeled data, but are primarily evaluated as technical contributions without consideration of their role in addressing these engineering challenges. To our knowledge, no prior work has adopted this perspective. Furthermore, while I-JEPA has shown strong results for 2D vision using ViT-based encoders, its adaptation to LiDAR point clouds remains relatively unexplored. AD-L-JEPA tests one approach but with different encoder and predictor architectures (CNNs). This work addresses both gaps by designing and evaluating an I-JEPA-inspired SSL model with ViT-based encoders adapted for LiDAR data, examining architectural choices and the broader software-system implications of adapting this approach.

4

Approach

The chapter walks through the supervised pipeline architecture and its component in turn, covering the pillar feature encoding and the transformer architectures that together create the backbone, the objet detection head and loss function, followed by the I-JEPA-based SSL setup, and how it is adapted to LiDAR.

4.1 Supervised Pipeline Architecture

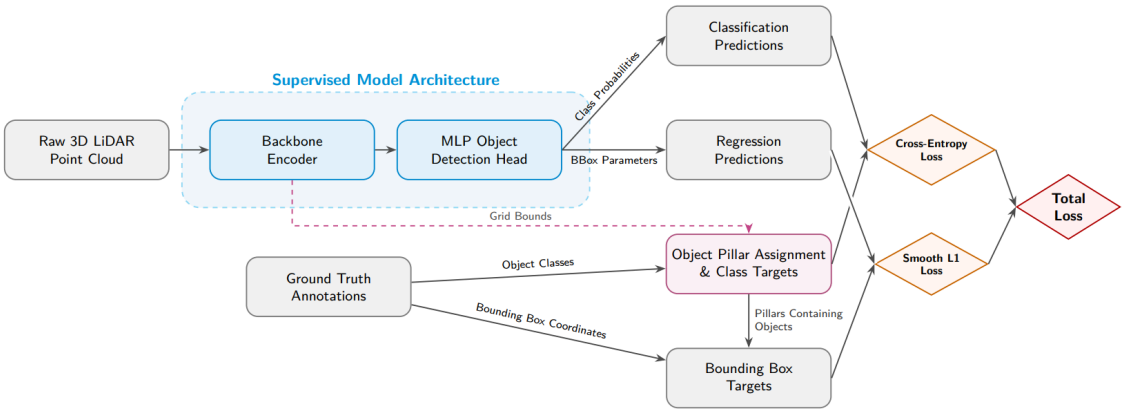


Figure 4.1: Overview of the Supervised Model Training Pipeline

Figure 4.1 illustrates the architecture of the training pipeline for the supervised model. The blue box shows the supervised model architecture, consisting of the backbone encoder and MLP object detection head, which takes raw 3D LiDAR point clouds as input and outputs class probabilities and bounding box parameters. These are then compared against targets derived from the ground truth annotations to compute the total training loss.

4.1.1 Backbone Encoder Architecture

Our encoder consists of two separate components, the Pillar Feature Encoder (PFE) and a transformer network, and will be used for both the supervised and the self-supervised model.

4.1.1.1 PointPillar Feature Encoder

The Feature Encoder is the component responsible for processing the LiDAR data into structured representations that can be used as input for the neural network. One method is to leverage the PointPillar strategy proposed in [38]. It builds on the voxel-based approach where a grid containing the LiDAR points is divided into cells, but utilizes a single dimension in the vertical direction z, resulting in vertical pillars. This is an appropriate approach as the point cloud in AD scenes usually varies more in the BEV plane than in z, which means that information can be largely retained while projecting the 3D points onto a 2D plane.

The process of transforming the raw LiDAR-point cloud starts by segmenting the grid into pillars, where the points inside each pillar are aggregated into a fixed-length feature vector of length C.

The input cloud can be defined as a tensor $\{p_1, \ldots, p_n\}$ where each point has coordinates (x, y, z) and intensity (i). The encoding of each pillar is done by calculating the distance of each point within the pillar to its center coordinates in the xy-plane, as well as the arithmetic mean of all the points. Then the points are augmented into a D-dimensional vector ($D = 9$) consisting of the intensity, absolute location, offset to center $(\Delta x_c, \Delta y_c)$ and distance to the arithmetic mean $(\Delta x_p, \Delta y_p, \Delta z_p)$.

$$p_{aug} = [x, y, z, i, \Delta x_c, \Delta y_c, \Delta x_p, \Delta y_p, \Delta z_p] \tag{4.1}$$

For all non-empty pillars (P) the points are stacked to create a tensor of shape (D, P, N). N is the maximum number of points per pillar, and if any pillar exceeds this, points will be randomly sampled and dropped, and if there are fewer than N points, the tensor will be zero-padded.

After that, a point-wise linear layer is applied followed by a normalization layer and a non-linear activation function, which results in a tensor of shape (C, P, N), where C is the encoded feature dimension. Then a max-pooling operation is applied to aggregate the point-wise channels into a single feature vector per pillar with the shape (C, P). To retain the information of where the pillar is located, the features are scattered back to the original center coordinates and the result is a pseudo-image of shape (C, W, H), where W and H are the width and height of the grid.

The final size of the output (the number of encoded pillars) will therefore depend on the resolution of the defined grid, as well as the chosen output feature dimension C.

All points or annotations outside the defined grid are not encoded, and therefore not included in either training or evaluation. No minimum or maximum number of points per pillar is chosen, which means that all points that fall within the grid bounds were encoded into pillar features to retain as much information as possible.

The size and resolution of the grid determine the amount of encoded pillars. This resolution is identified as an engineering trade-off, where a finer grid can increase

the potential detection capabilities for small objects and create more informative representations, but also increases the input sequence length to the transformer, affecting memory consumption and compute cost.

In this work a sparse approach where empty pillars are not included as input is used. Since most cells in the grid will be empty, this reduces the size of the input to the downstream transformer network.

4.1.1.2 Transformer

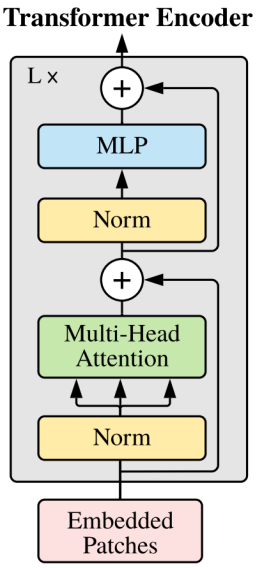


Figure 4.2: The ViT architecture from [19]. The input embeddings are processed through a normalization layer, a sequence of multi-head attention blocks followed by a second normalization layer and an MLP.

The transformer is a network architecture that uses an attention mechanism to model global dependencies within data. It was originally introduced for NLP, where it has been shown to be highly effective. Attention replaces the local receptive fields of traditional convolutions and allows the input elements to communicate with each other by assigning different levels if importance to one another. Mathematically it works by projecting each input into three vectors: Query(Q), Key(K) and Value(V). As seen in equation 4.2, the attention is a weighted sum of the values, where the weights are calculated by calculating the similarity via the dot product between the Queries and the Keys.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.2}$$

There are two common categories of attention, Self-Attention and Cross-Attention.

In Self-attention, the keys, queries and values all come from the same input, allowing the input to map dependencies to itself, such as correlated words in a sentence in NLP applications. In cross attention, the queries come from a different input than the keys and values, and dependencies are modeled between them, for example to translate from one language to another.

In practice, the attention mechanism runs in parallel with different learned projections called "attention heads". Dividing the projections has been shown to enable each projection to specialize towards representing different aspects of the input, such as geometric shapes or orientation. This mechanism is called Multi-Head Attention (MHA).

Originally constructed for using word-based tokens as inputs, the ViT successfully adapted the transformer architecture to 2D images by aggregating pixels into patches, which are then flattened and used as input tokens as shown in figure 4.2 [19]. A positional encoding is added to each patch to retain spatial information. The Norm-layer normalizes the input by Layer Normalization to stabilize the training, followed by the MHA-blocks and a MLP. The MLP consists of a few linear layers with non-linear activations. An important feature is the residual connections (the plus-sign) around the MHA-blocks and the MLP, which lets the model to preserve information across layers. This allows the network to focus on learning smaller updates instead of relearning entire feature maps.

For LiDAR processing, a similar logic can be applied but replaces the image patches with spatial structures, as defined in Section 4.1.1.1. In this project, the "tokens" for the transformer are the feature vectors (C, P) generated by the PointPillar encoder. In the same way that the ViT converts the image to a sequence of patches, the occupied pillars can be treated as a sequence of input tokens.

The final output is a variable number of feature vectors with D dimensions, depending on the number of encoded pillars N, forming an output vector of shape (N, D).

4.1.2 Object Detection Head

The object detection head takes the output of the transformer and converts it to final predictions. The Object Detection (OD)-head has shared internal layers with two parallel branches, one for classification and one for regression. The classification layer predicts whether a pillar contains an object, and which type of object it is, for each occupied pillar. The regression layer predicts the size, orientation and offset to the object center of the predicted object.

4.1.3 Object Detection Loss Formulation

In the context of object detection, the total loss (L_{total}) is typically a weighted sum of the classification loss (L_{cls}), measuring the error between the predicted class and the true class, and the regression loss (L_{reg}), which corresponds to the error of the

different properties of the object such as size, rotation and localization:

$$L_{total} = \lambda_{cls} L_{cls} + \lambda_{reg} L_{reg} \tag{4.3}$$

where λ represents hyperparameter weights used to balance the importance of each task.

4.1.3.1 Cross-entropy Loss

There is a wide range of loss functions that can be applied to the classification task, and which one to choose depends on the number of classes, the structure of the data and the model architecture [69]. Categorical Cross-Entropy (CCE) loss is commonly used when there are multiple different classes to predict, which is well suited for the various objects in the context of AD. When class labels are available as integer indices, the mathematically equivalent Sparse Categorical Cross-Entropy (SCCE) can be used, where the labels are encoded as integers.

Following the formulation in [69], if the class labels are given as integers $y_i \in \{1, 2, .., C\}$, and $\hat{p}_{i,j}$ is the predicted probability that prediction candidate i belongs to class j:

$$\hat{p}_{i,j} = \frac{\exp(z_{i,j})}{\sum_{k=1}^{C} \exp(z_{i,k})} \tag{4.4}$$

$z_{i,j}$ is the raw class logits output by the model at candidate i and class j. Then the individual loss is:

$$L_{query}(y_i, \hat{\mathbf{p}}_i) = -log(\hat{p}_{i,y_i}) \tag{4.5}$$

Where $\hat{p}_{i,y_i}$ is the probability that the model assigned the correct class to candidate i. The overall loss over n predictions will then be:

$$L_{\text{SCCE}}(y_i, \hat{\mathbf{p}}_i) = -\frac{1}{n} \sum_{i=1}^{n} \log(\hat{p}_{i,y_i}) \tag{4.6}$$

4.1.3.2 L1 Loss

L1 loss calculates the absolute difference between the target y and the predicted value $\hat{y}$:

$$L_1(y, \hat{y}) = |y - \hat{y}| \tag{4.7}$$

Smooth L1 loss is an extension of this, commonly used as the regression loss component in the object detection loss function. It was originally introduced in the Fast

R-CNN framework as a strategy to enable smooth training as the loss approaches zero. For the Ground Truth (GT) y and the predicted $\hat{y}$ value, the function is defined as:

$$L_{\text{smoothL1}}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & \text{if } |y - \hat{y}| < \beta, \\ |y - \hat{y}| - \frac{1}{2}\beta, & \text{otherwise.} \end{cases} \tag{4.8}$$

This two-part structure ensures that minor absolute errors (below β) are penalized quadratically, while larger absolute errors are penalized linearly, mitigating the influence of outliers [69].

4.2 Self-supervised Pipeline Architecture

4.2.1 I-JEPA - architecture

Our approach is largely based on the I-JEPA [4] architecture, but adapted for LiDAR point clouds instead of images. Building on the theoretical overview introduced in section 3.2, this section describes the architecture in detail. Briefly, I-JEPA avoids pixel-space reconstruction by instead predicting representations of masked image regions from surrounding context, using a teacher-student framework. The following describes each component and the training procedure.

The core learning objective of the I-JEPA-model is to predict the representations of various target blocks given a number of context blocks.

The model consists of a context-encoder, a target-encoder and a predictor that are all Vision Transformer (ViT)-based architectures. The context-encoder is also referred to as the *student*, and the target-encoder as the *teacher*. The teacher receives the full input and predicts the corresponding embeddings, and the student only a partial version of the same input, and with the help of the predictor is given the task to predict the missing context embeddings corresponding to the teachers output.

The target-encoder (teacher) takes an image converted into a sequence of N non-overlapping patches and returns corresponding patch-level representations of those patches. Then M blocks, possibly overlapping, are randomly sampled from the target representations with random aspect ratios and random scale, both within fixed ranges, to act as target blocks for the context-encoder (student).

The context-encoder input is a randomly sampled block from the same image with a random scale ranging between (0.85, 1.0) and unit aspect ratio. Any overlapping regions between the target blocks and context blocks is removed (masked) from the context block before going into the context-encoder to prevent a trivial prediction task. The output from the context encoder is the corresponding patch-level representations of the masked context block. Given these representations the predictor's task is to predict the M masked target block representations. To do this, the pre-

dictor takes the output of the context encoder as well as a mask token, i.e. a shared learnable vector including a positional embedding, for every patch that is to be predicted. This is repeated once per target block, each time with mask tokens encoding a different target block's location, to obtain predictions for each.

Finally, the loss calculated is the average L1 distance between the predicted patch-level representations, and the actual patch-level representations from the target-encoder output.

Both the predictor and context-encoder parameters are updated with gradient-based optimization while the target-encoders parameters are updated through an exponential moving average (EMA) of the context-encoder parameters. This way the target-encoder slowly follows the evolution of the student, and thereby producing increasingly better target representations.

4.2.2 SSL Model Architecture

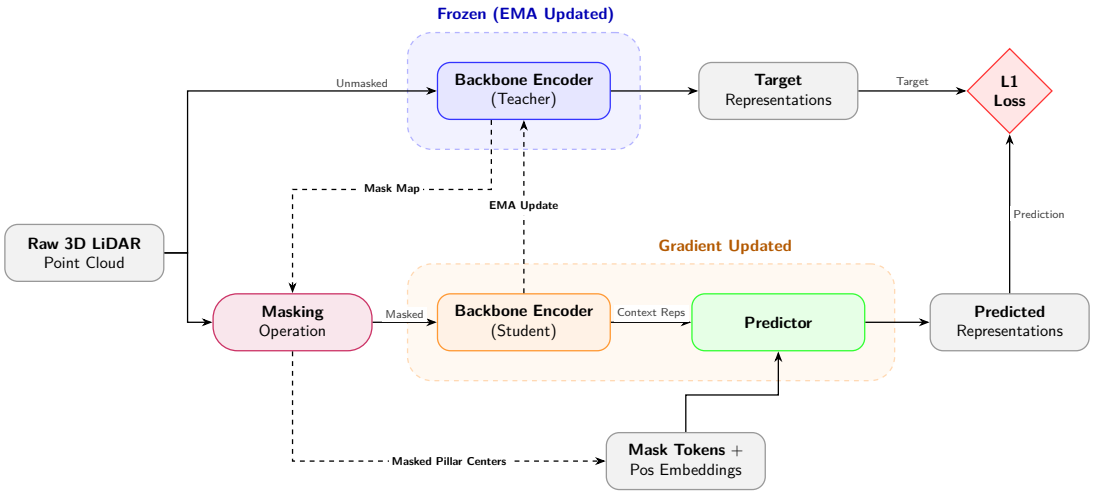


Figure 4.3: Overview of our Teacher-Student Pipeline

For the self-supervised pretraining setup, the exact same data pre-processing, point-cloud configurations and backbone from the baseline model are applied. Our setup adapts the Joint-Embedding Predictive Architecture (I-JEPA) [4] described in Section 4.2.1 for 3D LiDAR data. Where the original I-JEPA operates on 2D image patches, our artifact operates on encoded point-pillars. The pillar input in practice has the same format as the patch embeddings, with a center coordinate and a corresponding embedding. The difference lies in the fact that the point cloud is sparse, and therefore the length of the input vector varies, unlike image-based input. Since the I-JEPA utilizes the transformer network architecture for all model components, the adaptation to point-pillar input was straightforward as the transformer architecture can handle varying input sizes.

Following the theoretical framework, the artifact consists of three main components: a target-encoder (the teacher), a context-encoder (the student), and a predictor. The student-encoder reuses the combined PFE and transformer encoder backbone architecture established in Section 4.1.1. The teacher-encoder is initialized as a copy of the student. The predictor is also a transformer, but smaller in capacity, which is placed exclusively on top of the student encoder.

4.2.2.1 Pretext Task

In our LiDAR-based implementation, the pretext task translates to predicting the semantic embeddings of masked pillars using only the geometric and spatial context of the surrounding visible pillars.

While the architectural implementation of the ssl model is the same as I-JEPA, the task is slightly different. In I-JEPA, the training data consists of images with mostly one prominent object, while in AD, the scene usually contains multiple smaller objects with varying distances and levels of occlusion. Therefore, the masking strategies that are explored differ from the I-JEPA-approach.

5

Methods

5.1 Research Strategy

The study follows the Design Science Research (DSR) process, which involves iterative development and evaluation of a software artifact: the SSL-model [49, 12]. The goal is to achieve a successful pretraining strategy to construct a model that produces useful representations which can be transferred to perform downstream tasks, and thereby qualify as an alternative to supervised models in the perception system. The feasibility of our solution is measured by comparing final performance with a supervised counterpart acting as a baseline.

Figure 5.1: Overview of our DSR-process.

DSR is a problem-solving paradigm focused on building artifacts and generating design knowledge about why they work effectively in context [49, 12], which in our case is the perception system. A visual overview of our process can be seen in Figure

5.1. It is commonly divided into six steps, but as the model is only evaluated in an isolated setting and not tested in practice, we have chosen to aggregate the steps "demonstration" and "evaluation" as they overlap in our case, narrowing it to five steps, as listed below:

1. **Problem identification and motivation**. Method: Literature review. Output: Definition of SE challenges in perception, and analysis of architectural complexities of applying SSL to LiDAR data.

2. **Definition of solution objectives**. Method: Translating the problem identification into measurable performance metrics. Output: Establishment of desired outcomes and dependent variables in the study.

3. **Design and development**. Method: Iterative design across three cycles: establishing the supervised baseline (Cycle 1), building the SSL pipeline (Cycle 2), and refining the model (Cycle 3). Output: Insights from the exploratory design process and two developed artifacts, one SSL model and one supervised model.

4. **Demonstration and evaluation**. Method: Laboratory experiments, including both models, where the SSL model is the artifact to be evaluated. Output: Final performance evaluation through comparisons of the final SSL model versus the supervised model.

5. **Analysis and Communication**. Method: Aggregation of insights from the design process, experiments, and literature. Output: Design insights and analysis of impact on the perception system, synthesized in the sections Results and discussion.

5.1.1 Research Pipeline and Evaluation Strategy

There are various approaches for evaluating the artifact within different adaptations of the DSR methodology [72]. Since our study has a strong technical focus and does not involve a system in deployment, the evaluation stages provide quantitative performance metrics that serve as an proxy for system applicability.

5.1.1.1 Approach

Since this study was conducted in an offline laboratory setting, directly measuring software quality metrics like maintainability, scalability, and robustness was not feasible without a deployed system. Instead, our evaluation produced quantitative ML metrics for downstream detection performance, training efficiency, and label efficiency across varying label budgets, which formed the empirical basis from which we reasoned about software quality implications, supported by relevant literature. The following describes how each research question was answered:

Answering RQ1: This question was addressed in three parts:

1. The design of the training pipeline was varied to examine which design choices impact the quality of the representations, and how through qualitative visual diagnostics in combination with detection performance changes.

2. The observed effects were analyzed to reason about whether the design variables theoretically or empirically affect robustness, maintainability or scalability on an *internal* pipeline-level.

3. The implications were assessed at a *system*-level, by reasoning about the architectural impact of adopting SSL as a paradigm in the perception component, grounded in the achieved performance results and relevant literature.

Answering RQ2: We addressed the second research question by monitoring the effect on resource consumption of different design choices during development, to understand computational trade-offs in the pipeline architecture. Training cost is also measured in the final summative evaluation, providing a direct comparison of the SSL and the supervised paradigms in terms of training time and memory consumption.

Answering RQ3: Finally RQ3 was addressed by comparing performance of the two training strategies when given different label budgets. The resulting performance-label curve serves as an empirical base for comparing the label efficiency and reasoning about effects on annotation effort.

5.1.1.2 Formative and Summative Evaluation

Our evaluation strategy was divided into two common phases: *formative evaluation* to guide the development process, and *summative evaluation* that assesses the final artifact [72]. This separation was useful to allow for efficient iterative development while maintaining rigor for final comparisons. The formative evaluation phase corresponded to the iterative design cycles of the baseline and the SSL model. During this phase, the primary goal was to navigate the complex design space, test architectural setups, and establish working model configurations. This will guide us in how different design choices impacted the quality of the learned representations.

Formative Evaluation:

- **Approach:** We utilized a heuristic, trial-and-error approach.

- **Variable Handling:** The variables related to the training pipeline design (as defined in Tables 5.1 and 5.6) were varied and extended continuously based on internal diagnostics.

- **Rigor & Reproducibility:** Because the focus here was on rapid iteration and design knowledge generation, variables were not strictly isolated in every single run. The results serve as indicators guiding our engineering choices rather than definitive scientific evidence.

Once the SSL development phase was concluded, the artifacts entered the demonstration phase where summative experiments were conducted to evaluate and showcase the final performance, through a number of controlled training runs.

Summative Evaluation:

- **Approach:** A controlled laboratory experimenal setup.
- **Variable Handling:** The architectural and pipeline design choices determined in the design phase were locked, and the primary independent variable manipulated in this phase was the *label budget* as well as training strategy (fully supervised vs SSL).
- **Rigor & Reproducibility:** Absolute determinism is practically unachievable in distributed deep learning training due to hardware-level stochasticity [84]. To enable a fair comparison, dataset splits were constant and random seeds fixed where it was possible in the software stack to mitigate run-to-run variance as much as possible [42]. With that in mind, the performance metrics gathered provided a reliable basis for our final conclusions.

5.2 Problem Identification

The problem identification phase involved identifying challenges related to the maintainability and scalability of machine learning-based perception systems, particularly those that require large amounts of labeled data for training. This was conducted through an exploratory literature review, which was used to explore and define challenges and possibilities related to adapting SSL-based models in the perception software system. The findings are synthesized in Section 2.2. Furthermore, the study was also targeted towards investigating and motivating the design of the artifact, identifying challenges related to developing LiDAR-based SSL.

Google Scholar was used as a primary database to search for articles published in IEEE Xplore, SpringerLink, and ACM. Some pre-prints that were not yet formally peer-reviewed were still included and deemed credible based on citations and author reputation. Two broad search themes were applied. The first targeted software engineering concerns, combining keywords such as *"annotation"*, *"data dependency"*, *"machine learning"*, *"deep learning"*, *"pretraining"*, *"maintainability"*, *"scalability"*, *"software engineering"*, and *"technical debt"*. The second targeted research relevant for model development, combining keywords such as *"self-supervised learning"*, *"supervised learning"*, *"autonomous driving"*, *"object detection"*, *"representation learning"*, *"foundation models"*, *"LiDAR"*, and *"point cloud"*. Furthermore, a snowball approach was applied where references were explored to identify further relevant work. Deep learning-related searches prioritized recent work, while foundational contributions were included regardless of publication date.

5.3 Objectives of the Solution

The second DSR activity revolved around translating the identified problem into concrete objectives for the software artifact.

The main objective was to develop a modular self-supervised LiDAR perception model that reduced the need for manual annotations while retaining or improving performance as much as possible. This was to be achieved while minimizing computational cost in terms of training time and memory consumption. A modular approach was chosen where each component of the pipeline was separated to enable future substitution of the dataset, encoder, backbone, and task head.

To evaluate to what extent this objective was reached, the following measurable goals were identified.

Downstream Object Detection Performance
Goal: The model should perform as well as possible when applied to a downstream task. The fundamental requirement of the perception backbone is to learn transferable representations of the environment. Because object detection performance indicates the usefulness of these representations, the primary goal is to maximize downstream performance when a task-specific head is appended to the pre-trained architecture.

Computational and Training Efficiency *Goal: The model should be as resource efficient as possible.* The training pipeline must be computationally viable and resource-efficient. The objective is to optimize and minimize both memory utilization and training time. Maximizing compute efficiency limits the infrastructure costs associated with initial model training and continuous retraining during model maintenance.

Label Efficiency *Goal: The SSL model should manage to achieve competitive performance on limited label budgets.*
To mitigate the data dependency risk and manual annotation bottlenecks inherent in supervised learning, the model should maximize the utilization of the available labels.

5.4 Design and Development Process

Following the iterative nature of the DSR process, the design and development process was divided into different cycles, each of which was iterated to achieve a desired outcome. To be able to verify our progress and improve the design, objectives were formed for each cycle with corresponding internal evaluation steps, guiding the development towards the desired end result. Since the search space of potential variations and experiments is vast in the SSL-domain, a heuristic approach was applied, meaning that we selected variables, and varied them guided by indications from internal evaluation steps rather than trying to optimize every single configuration.

Design choices were treated as variables throughout the development, and referred to various architectural design decisions that had to be made and balanced in order to solve the problem at hand.

5.4.1 Experimental Setup

Each experiment was conducted on 4 NVIDIA A100 GPUs (40 GB VRAM) and 128 CPUs (480 GB RAM). The software environment was built on Python (3.14.4) and PyTorch (2.8).

To achieve an efficient training process for multiple GPUs, we used PyTorch's Distributed Data Parallel (DDP) with the NCCL backend.

5.4.2 Design Cycle 1 - Baseline Model and Infrastructure

The first iteration of our design process focused on establishing the core software infrastructure. This involved implementing the data processing pipeline, the object detection head, and the backbone architecture. The objective of Cycle 1 was to build a stable, modular, and resource efficient artifact, since the same architectural components were used in the SSL framework.

5.4.2.1 Design Challenges

The primary design choices when developing the supervised model architecture and the overall infrastructure revolved around how the unstructured 3D LiDAR data would be mapped into a structured format and filtered. These were fundamental architectural decisions that we had to balance to ensure a robust pipeline and a baseline model that was both computationally viable and capable of learning. Table 5.1 summarizes the challenges we identified, and the following sections describe how we decided to handle them.

Table 5.1: Core Architectural Design Choices for the Perception Pipeline

Design Choice	Description	Quality attribute	Rationale
		Architecture	
Grid Size & Resolution	The physical range and cell dimensions of the encoded pointpillars.	Scalability, Robustness	A finer grid can help detect small objects, which is important for robustness but increases memory use and processing time, impacting scalability.
Task formulation	Choosing the mathematical objective functions.	Robustness	Ensure the model does not ignore one task accuracy in favor of another, or collapse.
Post-Processing	Rules and thresholds for filtering out predictions.	Robustness, Maintainability	The filtering strategy and thresholds affect robustness, via balancing false positives or missed true predictions. More sophisticated filtering strategies may also increase implementation complexity and complicate traceability of predictions, affecting maintainability.
		Model Capacity	
Model Capacity	Embedding dimensions and the number of transformer layers.	Scalability	We want to be able to scale the model size up and down without triggering Out-of-Memory (OOM) crashes during distributed training.
Training Hyperparameters	Learning rate, weight decay, gradient clipping, and warm-up epochs.	Robustness	Confirm that the training loop is stable, the loss reliably decreases, and gradients do not explode over time, indicating robustness of the training pipeline.

Note: The design choices were identified heuristically, guided by prior literature and empirical observation, as the design space of such architectures is practically unbounded.

5.4.2.2 Data Processing

The dataset selected for training and evaluation of the models was the Zenseact Open Dataset (ZOD). The dataset contains LiDAR point clouds, camera images, radar data, and a diverse set of annotations. This dataset was chosen because it contains dense point clouds and high geographic diversity. The average point-cloud density is 254k points per frame, which can be compared to similar datasets like Waymo Open with 177k points per frame. This makes it suitable for LiDAR-based training, and the large geographical area covered enables testing for robustness in varying environmental conditions (705 km^2 vs. 76 km^2 (Waymo)) [3].

Table 5.2: Overview of the Zenseact Open Dataset (ZOD). Full dataset available at `https://zod.zenseact.com`.

Property	**Value**
Total Frames (camera-LiDAR pairs)	100,000
Training Frames	*89,972*
Evaluation Frames	*10,023*
Annotation Types	Static and Dynamic Object, Lanes, Traffic Signs, Ego-Road, Road Condition
3D Annotation Range	Up to 245 m

ZOD contains both driving sequences and individual frames. A key-frame is defined as a snapshot midway through a sequence of 20 seconds, and for each key-frame, ground truth annotations are provided. The LiDAR points exist for a full 360 degree view, but annotations are limited to the front 120 degree view to align with the camera perspective. Therefore, when training on labels, we limit the input to only include points that fall into this angle, as shown in Figure 5.2.

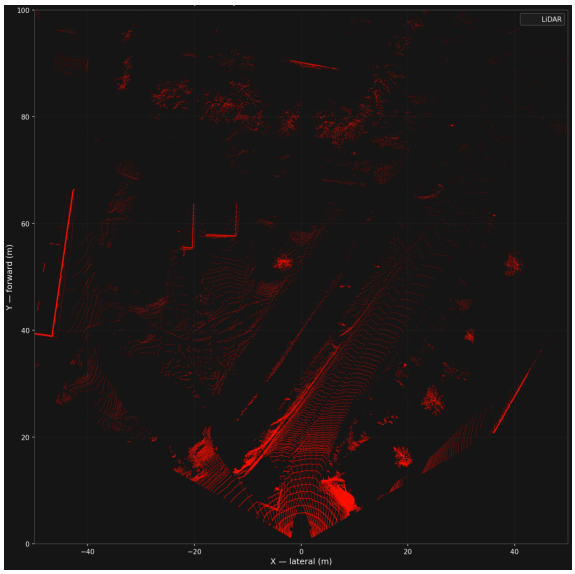


Figure 5.2: Field of view (FOV) angle of the point cloud that contains annotations (limited to [-50,50] in x and [0,100] in y for visibility).

The dataset for training and evaluation was constructed by extracting the following data: Key-frame LiDAR point clouds with corresponding point coordinates, intensity and timestamp, as well as object-level annotations filtered by class and their corresponding bounding-boxes consisting of size, center coordinates and rotation.

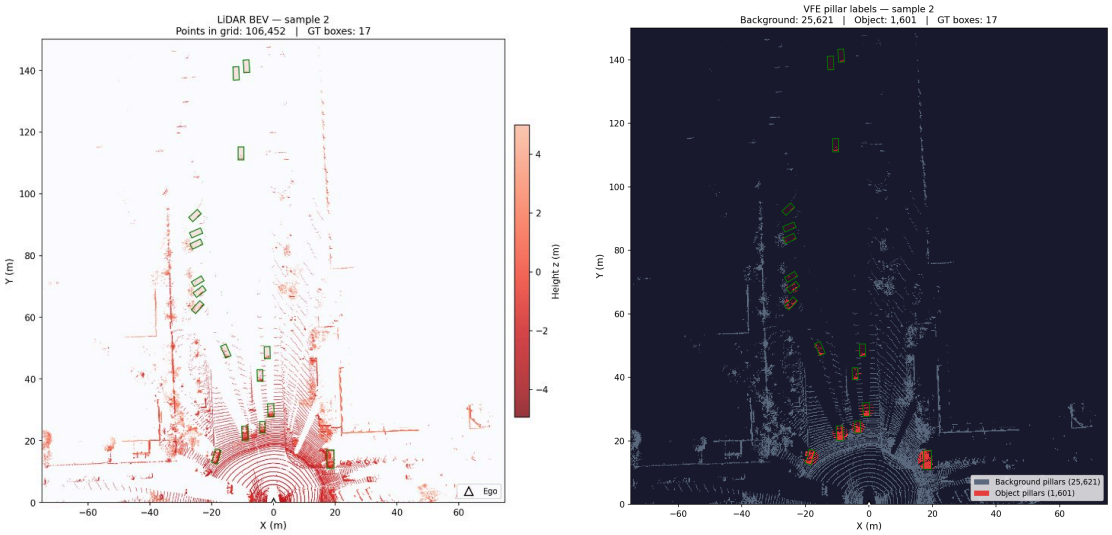
We chose to only include the object class "Vehicles" including subclasses, as it is a common class in the dataset and annotations exist in most frames. This was to make the object detection task stable in difficulty and the results relatively easy to interpret.

Timestamp was normalized to a delta-time relative to the first recorded point of each frame, and classes were encoded as integers.

5.4.2.3 Target generation

This section describes how the targets for the object-detection task were defined.

To assign which point pillars that contained an object, we chose to calculate a circle that surrounds the GT bounding-box by defining the radius as the Euclidean distance from the center to the corner of the box in the xy plane. All occupied pillars whose centers fall into this circle are assigned the respective class of the ground truth object. This approach was chosen to account for the fact that many objects only have LiDAR points on the side or a corner, and not always active points in the middle of the object. Using a circle ensures that pillars near the corners of the bounding-box are also included in the target assignment, an example is shown in Figure 5.3. A weakness of this approach is that some pillars that are not part of the object will still have to predict the object that it is close to, which can be challenging, and an alternative would have been to calculate exactly which pillars fall into the exact bounding boxes, but that approach would have been less compute-efficient. Since the circles can overlap, the pillars that fall into more than one radius are assigned the object whose center is the closest. The result is a target vector of shape $(N\_positives, N\_classes+1)$ where the $+1$ allows for predicting "no object".



(a) Lidar point cloud with vehicle annotations

(b) Corresponding encoded empty pillars and target object pillars

Figure 5.3: Example of LiDAR point cloud, ground truth boxes, and target pillar assignment.

For every positive object pillar *N_positives* the corresponding position, size and rotation of the bounding-box were assigned to form the regression targets. To create targets within reasonable scales, rescaling was done to create the following regression targets. X and Y as offset from the pillar center to the ground truth center, so that the model learns a small, bounded correction rather than an absolute scene coordinate. Z was assigned as its absolute value, since the variation is already in a reasonable range. Size in (*width*, *length*, *height*) was converted to log scale for more stable training. Finally rotation in quaternions ($q1$, $q2$, $q3$, $q4$) was converted to sin and cos yaw angles, to provide a more appropriate scale for training. This resulted in a target vector of shape (*N_positives*, 8):

$$\text{target} = [\, X_{\text{offset}},\ Y_{\text{offset}},\ z,\ \log w,\ \log h,\ \log l,\ \phi_{\sin},\ \phi_{\cos}\,] \tag{5.1}$$

5.4.2.4 Loss function

The loss function determines the targets for the model, and calculates how far off the model predictions are to the GT annotation bounding boxes and classes.

For the classification we used a weighted categorical cross entropy loss, as described in Section 4.1.3.1. Since the assigned object pillars are few compared to the background, it results in a class imbalance. This class imbalance biases the model towards predicting background, as correctly classifying the few positive pillars contributes little to reducing the overall loss. This was mitigated by increasing the loss weight for positive predictions with a factor of 5.

The regression loss was calculated for every pillar that had an active object assign-

ment. To calculate the regression loss we used Smooth L1 as described in 4.1.3.2. Furthermore, we decided to weight the loss to balance their relative magnitudes, as initial unweighted runs showed that the xy offset loss dominated the total regression loss. As a result the following weight factors for the loss were applied:

- $(X, Y)Offset$: 3.0
- $Rotation$: 1
- $Size(x, y, z)$: 0.5
- Z: 1

5.4.2.5 Post-processing

To be able to measure the performance of the model, the predictions need to be converted into concrete bounding-box predictions.

From the classification head, the predictions are filtered by a selected threshold of $x = 0.5$, where a prediction with a classification score under it will be classified as non-object pillars. For each positive prediction, its corresponding prediction for the object bounding-box was calculated. Regression output had to be converted into real bounding-box features. This was done by calculating the center coordinates, by mapping back to world coordinates from the grid index and subtracting the offset prediction. Yaw rotation is converted back to quaternions, and log size predictions were mapped back to original size in meters.

Since each object pillar is making a prediction, but multiple pillars might be predicting the same object, redundant predictions had to be sorted out. This was done by implementing an Non-Maximum Suppression (NMS) algorithm. In our implementation of NMS, all predictions are sorted according to confidence, and the most confident prediction is selected and all other predictions overlapping more than a certain threshold $y = 0.01$ are sorted out. From the remaining predictions, the one with the highest confidence is selected and again all remaining predictions overlapping it above the threshold are discarded. This process repeats until there is at most one prediction per object.

Intersection over Union (IoU) calculates the degree of overlap between the prediction and the GT bounding-box. In 2D it is calculated for the prediction bounding-box area B_p and the ground truth area B_t as follows:

$$IoU = \frac{\text{Area}(B_p \cap B_t)}{\text{Area}(B_p \cup B_t)} \tag{5.2}$$

For 3D-IoU the same principle is applied but for the overlapping volume. Calculating the 3D overlap is both computationally inefficient and potentially causesing the sorting to become too harsh resulting in decent detections being discarded, therefore

we decided to calculate the overlap in 2D.

5.4.2.6 Cycle 1 - Objectives and Internal Evaluation

To be able to gather insights and to iteratively improve the model, we first defined the relevant evaluation metrics for the object detection task, and then continuously iterated over the design to improve the performance, and incrementally made decisions about the final design based on the results.

Our metrics are standard object detection metrics, as well as a number of system performance metrics, to evaluate how training efficiency and hardware resource use are affected. These are described below:

Mean Average Precision (mAP) is one of the most common metrics in object detection [69], and is the main downstream performance metric used in this study.

First, the ratio of True Positive (TP) (number of correctly identified objects), False Positive (FP) (a prediction made where there is no object), and False Negative (FN) (objects missed by the model) are calculated.

A prediction is labeled a TP if $IoU(B_p, B_t) > threshold$, where the threshold is between 0 and 1. It is labeled FP if the prediction is not meeting the threshold for any GT box. If a GT box does not have any prediction within the threshold, it is missed by the model and labeled as FN.

Precision and recall measure the ratio between the correct predictions and the total predictions, respectively:

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN} \tag{5.3}$$

Average Precision (AP) balances the precision and recall across different IoU thresholds. For a specific IoU-threshold and class, it is calculated as the area under the precision-recall curve:

$$AP = \int_0^1 p(r), dr \tag{5.4}$$

If the performance is averaged over all classes, we get mAP.

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{5.5}$$

Mean Absolute Error (MAE) measures the average of the error for each predicted value. The error is the absolute difference between the predicted value and the target value:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{5.6}$$

MAE is used to evaluate the predicted bounding-box properties such as size, rotation and center location, to measure to what extent the model manages to correctly predict each of the features.

Time-to-Train is a typical measurement of how long it takes to train to a certain quality threshold. To ensure fair comparison, timing can be targeted to a specific part of the process. Following [42], we record from where the system touches the training data to the point where a threshold is reached, excluding initialization and data-reformatting and other overhead operations. The threshold is set to model convergence, meaning that some variable, for example loss or performance, has reached a plateau or started to decline during training, and the convergence criteria we define as limited further improvement over a set number of consecutive epochs [32].

Time/Epoch shows the time it takes to train one epoch, and can be used as a measure of individual impacts of architectural changes or in aggregation to estimate time-to-train.

Peak-Video Random Access Memory (VRAM) Utilization represents the maximum VRAM consumed during training. The hardware has limits to the amount of data it can hold and memory footprint scales with the number of parallel operations that are done, therefore the peak memory utilization is used to measure the memory requirements for the training process.

5.4.2.7 Pipeline Robustness

The pipeline relies on standard machine learning capacity parameters. While determining the absolute optimal configuration for these is not the primary research focus, a reliable training pipeline must demonstrate robustness against variations in these parameters.

To validate to what extent this was reached, we conducted a targeted ablation study on some of the parameters listed in Table 5.1. The goal was not finding optimal configurations, but rather a stress test of the software infrastructure. By varying the embedding dimensions and loss weights, we confirm that the model consistently converges, handles memory allocation dynamically without crashing, and maintains a stable training loop. Training hyper-parameters were kept constant and one independent variable was changed at a time, treating the remaining as control variables. The experiments with independent and dependent variables are listed below:

Table 5.3: Robustness Ablation Study, see 6.1 for results

Nr	Independent variable	Change	Dependent Variables
1	Transformer Embedding Dim	$128 \rightarrow 256$	mAP,VRAM, Time/Epoch
2	PFE Embedding Dim	$64 \rightarrow 128$	mAP,VRAM, Time/Epoch
3	OD Head Dim	$128 \rightarrow 256$	mAP,VRAM, Time/Epoch
4	Backbone Layers	$3 \rightarrow 4$	mAP,VRAM, Time/Epoch
5	Custom Loss Weights	$XY : 3.0, Size : 0.5, Rot : 0.5, Pos : 5.0$	mAP
6	NMS Threshold	$0.5 \rightarrow 0.01$	mAP
7	Grid Res (XY)	$0.8m \rightarrow 0.2m$	mAP,VRAM, Time/Epoch

The ablation study also served as an indicator for compute tradeoffs (RQ2) and the results guided us in the later cycles of development.

5.4.2.8 Configuration of Backbone Encoder

The specific hyper-parameters used in our implementation of the backbone (Section 4.1.1) are summarized in Table 5.4, as it entered the second cycle of development.

Table 5.4: Parameters for the encoder

Parameter	Value
PFE	
x, y, z range	$[-75, 75] \times [0, 150] \times [-3, 5]$
Grid Resolution $(\Delta x, \Delta y)$	(0.2, 0.2)
Embedding Dimension (C)	256
Transformer	
Number of Layers (L)	4
Embedding Dimension (D)	256
Attention Heads (H)	4

5.4.2.9 Configuration of Object Detection Head

The parameters of the object detection head (Section 4.1.2) are fixed since it will be consistently reused across our models without any changes.

Table 5.5: Parameters for the object detection head.

Hyperparameter	**Value**	
Embedding Dimension (D)	256	
Hidden Dimension	Embedding dim x4	
Number of Hidden Layers	2	$\phi = yaw\ angle$
Activation Function	ReLU	
Classification Output	$N_{classes} + 1$	
Regression Output	8 $(x, y, z, w, l, h, \phi_{\sin}, \phi_{\cos})$	

5.4.3 Design Cycle 2 - SSL Pipeline

Following the DSR iterative loop, this cycle aims to transition the backbone architecture from cycle 1 into a teacher-student framework to enable pretraining on unlabeled data. By iteratively taking different design decisions this cycle further aims to disclose what design decisions affect the learned representation and consequently how they impact the scalability, maintainability and robustness of the perception component.

5.4.3.1 Design Variables

During this cycle, we treated the masking logic and the EMA update rate as parameters for heuristic exploration (summarized in Table 5.6). Several design variables from Table 5.1 were also reused and treated as parameters, such as grid resolution and model capacity parameters.

Table 5.6: Design Variables: SSL Pipeline and Task Formulation

Component	Description	Quality attribute	Rationale
Self-Supervised Pipeline Design			
Masking strategy and Ratio	The spatial masking pattern applied to the point cloud, and the fraction of visible context.	Robustness	Different masking strategies can impact semantic learning and how the model learns global context, or affect training instability, which is unwanted in a robust training pipeline.
EMA Update Rate	The fraction of the student model weights transferred to the teacher model each epoch.	Robustness	Slower updates can increase training stability, while faster updates adapt quicker to recent gradients.
Predictor Size	Number of layers and embedding size of the predictor transformer	Scalability	The predictor needs to have enough capacity to perform its task while balancing it's capacity in relation to the encoder. The transformer-based structure makes it computationally intensive and impacts scalability.

5.4.3.2 Masking Strategies

Initially we tested two masking strategies, random masking and block masking. The masking was performed by sampling grid indices throughout the grid, each corresponding to a point pillar, based on a ratio and a masking strategy. All points that fall into those cells are removed before the point cloud is sent to the student encoder. We have two main approaches to select those grid indices.

- Random masking: Grid indices are selected randomly
- Random block-based Masking: The grid is divided into blocks of a certain size spanning a number of indices. Non-occupied masking blocks are sorted out and blocks to mask are randomly selected from the remaining based on a certain mask-ratio.

The masking ratio corresponds to the percentage of the scene that is masked out. Previous work trained on images with one prominent object, a mask ratio of 70% with block-based masking has been shown to be most successful in forcing the model to learn semantic representations [4]. However, AD scenes include multiple small objects and the mask rate and block size were explored with this in mind.

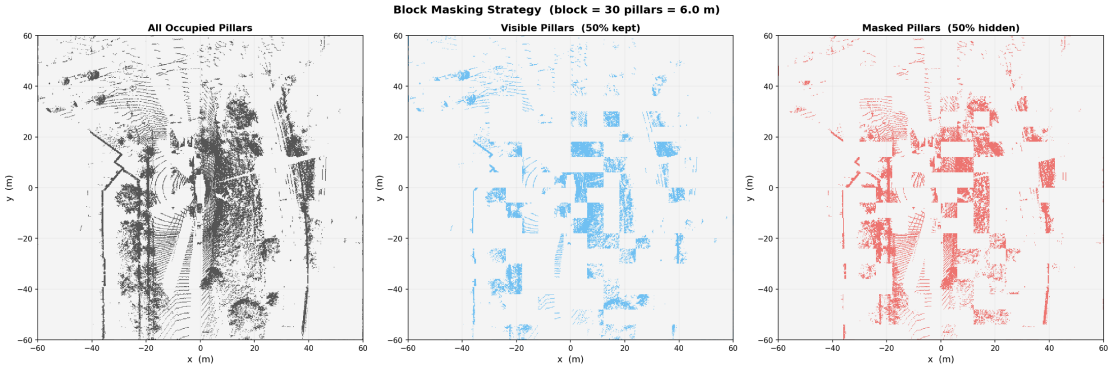


Figure 5.4: Visualization of block masking.

5.4.3.3 Cycle 2 - Objectives and Internal Evaluation

The focus of the second design cycle was to implement an initial self-supervised training pipeline with the objective to get initial indications of a working model. To efficiently be able to search for viable approaches we evaluated the quality of the representations based on two different approaches, first by applying a small frozen backbone evaluation step to see the evolution of downstream task performance, and second, through visual qualitative evaluation.

Step 1: Frozen backbone evaluation. To evaluate the quality of the representations, the model can be assessed on its performance on a downstream task following the frozen protocol, also known as Linear Probing. In this setting, a pre-trained model $\hat{\mathbf{h}}$ has its parameters frozen, meaning that no gradient updates are applied, and a task specific head g_D is trained on a small amount of labeled data. The quality of $\hat{\mathbf{h}}$'s representations is then determined by how well g_D performs on the downstream task [46]. Sometimes a single linear layer is used as the head g_D, but this study utilizes a MLP as a non-linear probe (the object detection head), to be able to capture the complexity of 3D-object detection. Since neither a linear nor non-linear classifier is complex enough by itself to perform such tasks, the performance shows whether the underlying representations contain sufficient information. From a software engineering perspective, this protocol evaluates the reusability of the backbone.

In this case after each epoch of pretraining, the object detection head was re-initialized and trained on 50000 labeled samples for one epoch and evaluated on object detection performance on a set of 5000 samples. The dataset-splits are constant across all training runs. Even though 50000 samples does not ensure reaching convergence, it allows for efficient testing and comparative results across epochs and between training runs. This way, we can estimate to what extent the internal representations evolve in terms of semantic understanding of objects, since the object detection head itself is too shallow to learn it from scratch.

Step 2: Visual qualitative evaluation. Principal Component Analysis (PCA) is a dimensionality reduction technique that can be utilized to visualize the information

of the representations. It is commonly used for models such as DinoV2 [47] as a tool to indicate how the model sees its input. It works by linearly transforming the high-dimensional embedding space into a lower-dimensional representation, capturing the orthogonal directions of maximum variance within the data. In the context of representation learning, it is common practice to extract the first three principal components of the dense feature maps and map them to Red, Green and Blue (RGB) color channels. Then we project the encoded LiDAR pillars back into their grid and coloring them based on these PCA components. The goal was to see patterns in the plots that indicates what the model has learned, that being similar colors for similar objects or areas.

We also mapped the evolution of the *Feature Entropy* as it serves as a diagnostic metric to monitor the diversity of the learned representations and to detect potential dimensional collapse. By calculating the average entropy of the embedding space across a batch of samples, we can quantify the information density of the network. High entropy indicates that the model is actively utilizing the available embedding dimensions, whereas a drop in feature entropy acts as a warning sign of training instability and network collapse [39].

5.4.4 Design Cycle 3 - Refined SSL-Model

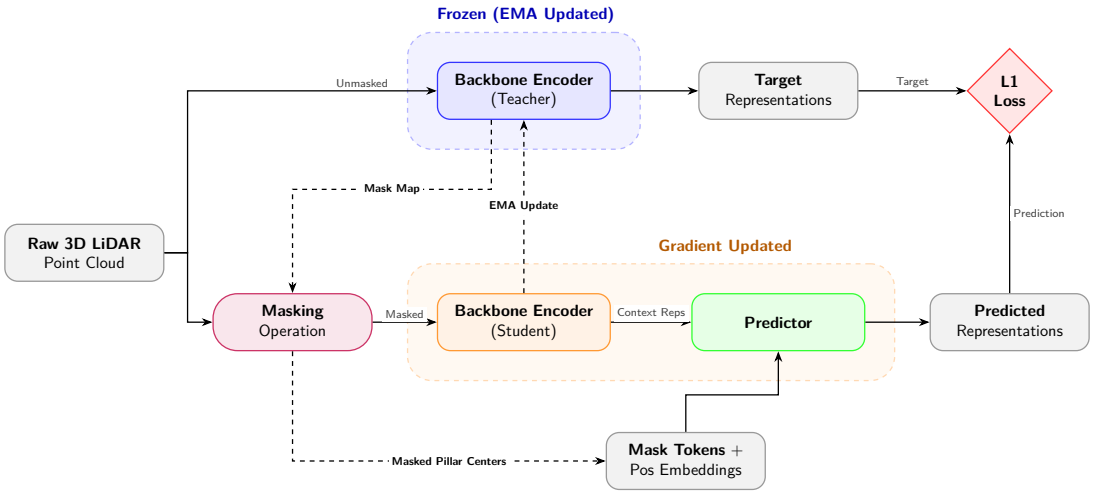


Figure 5.5: Extended architecture of our Teacher-Student pipeline

This section describes the continued development of the LiDAR-JEPA model first introduced in Cycle 2. Rather than starting from scratch, Cycle 3 treats the Cycle 2 model as its baseline and refines it by introducing data augmentations, extending the pretext task, and experimenting with different masking strategies. Together, these additions represent an iterative step forward, shaped by the knowledge and results of the previous cycle.

5.4.4.1 Data Augmentations

While I-JEPAs pretext task is designed to learn semantic representations, point cloud data introduces a unique failure mode. Since the pillar center coordinates serve as positional encoding in the predictor, the features carry real-world spatial information. This means the model may exploit spatial position rather than learning meaningful features, which can result in dimensional collapse. Furthermore the augmentations serve as implicit regularization, encouraging the model to learn representations that are invariant to the applied transformations, which has been shown to improve generalization and robustness of representations [79, 28, 18].

To prevent this, we applied different rotations and translations to the teacher and student inputs during pretraining. The teacher and student inputs are each transformed independently, receiving their own randomly sampled rotation around the z-axis and a random translation of up to ± 10 m along both x and y. This ensures that the two views see the same scene from slightly different spatial perspectives. To ensure the predictor receives correct positional encodings for the masked pillars despite the differing transformations, the centers of the masked pillars from the teacher view are transformed into the student coordinate frame before being passed to the predictor. This ensures that the predictor gets the correct positional encodings for the masked tokens.

By transforming each branch with a different rotation angle and translation offset, the pillar center coordinates of the student and teacher are in entirely different coordinate frames. Consequently, absolute positional information becomes not only uninformative but actively misleading. A predictor that attempts to exploit spatial position to predict the teacher's target representations will fail, as the two coordinate frames are inconsistent. The model is therefore forced to ground its predictions in local geometric structure and contextual relationships, which remain consistent across coordinate frames, rather than position. This will furthermore help to make the features invariant, i.e. that a car will have car features, regardless of its position or rotation.

5.4.4.2 Extended Pretext Task

While the data augmentations generated improvements in mAP and more structured PCA visualizations, the model still showed signs of exploiting a shortcut. Since the masking strategy exclusively targets occupied pillars, the model can learn that any token presented as a prediction target is occupied, trivially solving part of the pretext task without reasoning about the surrounding scene context. In the context of images, this is not a problem, as all patches in an image will always contain information, there is no such thing as empty space.

To break this shortcut, we introduced decoy pillars, i.e., mask tokens of empty pillars. We sampled pillars adjacent to the occupied ones, and randomly sampled a subset of these, that were then included in the set of prediction targets alongside the actual mask tokens. Rather than assuming occupancy from the masking itself, the model must reason about scene structure to distinguish signal from noise among

the prediction targets.

5.4.4.3 Predictor Variations

An important part of the architectural balance point out by Assran er al. [4], is that having a more narrow predictor in comparison with the encoder yields better downstream performance. The idea is that if the predictor acts as a bottleneck, the encoder is forced to learn more meaningful representations. Therefore we introduced variations in the width of the predictor in comparison to the encoder, to see how that affects the outcome.

Secondly, we introduced a variation of the transformer architecture of the predictor, to utilize cross-attention instead of self-attention. This means instead of letting all mask tokens and all context tokens form both the query and key vectors, we treat the mask tokens as queries and the context tokens as keys. That means that all tokens can no longer attend to each other, only the mask tokens to the context tokens.

This was another attempt to break the shortcut of merely interpolating between context tokens and predicting the positions of the mask tokens. In these experiments the decoy pillars were redundant and where therefore disabled during the runs with the cross-attention predictor.

5.4.4.4 Cycle 3 - Objectives and Internal Evaluation

The objective of this cycle was to refine and extend the architecture of our initial SSL setup, through iteratively trying new strategies to achieve a successful pretraining strategy. We use the same approach as in cycle 2, where we evaluated the representations learned by visualizing the PCA, mapping the feature entropy, as well as the small frozen backbone evaluation step after each epoch.

We introduced visualizations of cosine similarity as a complementary diagnostic plot. Cosine similarity measures the angular alignment between two vectors, and is defined as the dot product of the vectors divided by the product of their lengths:

$$S_C(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|}$$

In the context of our SSL architecture, cosine similarity was applied between the different pillar embeddings to qualitatively evaluate if different objects of the same class had similar embeddings. This was done by computing pair-wise cosine similarities and visualizing the scores as a heatmap.

5.5 Demonstration and Final Experiments

The fourth step of the DSR process involves demonstration of the software artifact. Our approach to do this was to conduct a series of controlled experiments that showcase the performance of the best performing pre-trained model at the end of the last

development cycle, compared to the supervised baseline. To enable a fair comparison, the baseline was trained using the same architectural setup as the final SSL model, including grid dimensions and capacity for the PFE, transformer and object detection head. The environment, dataset splits and training hyper-parameters were kept constant.

The experiments were divided into two steps. The first step was focused on RQ3, where we wanted to investigate the performance when training on limited label budgets. The models were trained until convergence, and we focused on small labeled splits. That means that the label-budget experiments were designed to showcase the potential gain in performance in the low label-budget regime.

While step 1 showcased the different models performance under constraint, step 2 was intended as a comparison of the models performance with the full label-budget available.

5.5.1 Experimental Setup

5.5.1.1 Training Strategy

We adapt the pre-trained encoder using both the frozen protocol (described in Section 5.4.3.3) and fine-tuning protocol, where the pre-trained backbone weights are unfrozen and jointly optimized together with the object detection head. Since the backbone already carries meaningful features from pretraining, it is optimized with a small learning rate to avoid moving too far from the pre-trained weights. While this approach requires more computation than the frozen protocol, it empirically performs better as it allows to adapt to the downstream task [46].

The frozen protocol can be argued to be better suited for evaluation of the representations, since by strictly decoupling the backbone from the task head it ensures that the performance reflects the knowledge gained from pretraining rather than newly learned patterns during supervised fine-tuning. The fine-tuning protocol however can further show the full potential performance when utilizing a pre-trained model.

The OD-head received the same training hyper-parameters such as learning rate for all training strategies, and a learning rate of 1/10 was applied to the pre-trained backbone when following the fine-tuning protocol. For training stability during fine-tuning, the backbone weights were frozen for the initial 5 epochs before allowing gradients to flow through the encoder as well.

Each experiment training was run until convergence, which we defined as no improvement over 5 consecutive epochs after reaching a peak in mAP for step 1 (label-budget experiments). It was extended to 8 consecutive epochs for step 2 (full dataset training) to avoid missing a potential further peak. To exclude parts of the process that were not part of training, such as data-loading, evaluation and plotting, time-to-train was calculated by measuring training time/epoch and averaging over all epochs, and peak VRAM was defined as the peak over all epochs.

For experiment efficiency, we did not run evaluation after every epoch in the small label budget regime, and instead after an interval starting at 50 epochs for label budget 1 (see Table 5.8). This means that a higher mAP might have been in between those evaluation runs, but since we kept it constant for both models it still allows for comparison, and we qualitatively evaluated the mAP-curve to ensure that they were smooth.

5.5.1.2 Control Variables

To isolate the effects our independent variables across training runs, all other factors were treated as control variables and kept constant across all experimental runs. This includes hardware configurations and training hyper-parameters.

The same full evaluation set is used across all experiments, and for the different training splits we randomly sample from the full training dataset using random sampling with fixed seed.

However, due to the nature of the experiments, some of the variation can also occur due to the stochastic nature of training neural networks. We mitigate this as much as we can by using fixed seeds where it is possible, such as always initializing the OD-head with the same weights, but fully deterministic training is still unachievable and should be taken into consideration when interpreting the results.

5.5.1.3 Dependent and Independent Variables

Target Software Artifacts:
The pre-trained model was chosen based on the setup that achieved the best performance in terms of mAP from the internal evaluation of Cycle 3 in the development phase. A checkpoint is chosen where the best performance was registered, since more training epochs might not necessarily yield better results during pretraining.

The supervised baseline-model and the final SSL model share identical encoder backbones, and receive the same model capacity and configurations for the summative experiments to allow fair comparison. During evaluation, the transformer predictor head of the SSL-model is discarded and replaced with an identical OD-head as the baseline, which means that the two models have identical architectures, with training strategy as the only difference between them.

Dataset Splits:
For the label budget experiments, we used a subsample of the validation set of 2000 samples for efficient testing. This split was kept constant across all runs. We also sub-sampled the training set depending on label budget. We checked that there were annotated vehicles in most of the frames, and that their distribution was not too different from that of the validation set. Note that the distribution differs from the ZOD dataset, and this is because of the fact that all annotations that fall outside our defined grid are excluded.

Table 5.7: Evaluation Dataset Distribution

Dataset	Samples	GT Boxes/Sample
Label Budget Evaluation	2000	min=0 max=38 avg=6.2, empty: 263/2000
Full Evaluation	10023	min=0, max=58, avg=6.4 empty: 1272/10023

Table 5.8: Training Dataset Splits

Samples	GT Boxes/Sample	Eval Interval
1	min=19, max=19, avg=19.0, empty: 0/1	50
10	min=2, max=19, avg=6.8, empty: 0/10	20
100	min=0, max=23, avg=6.7, empty:10/100	2
250	min=0, max=26, avg=6.4, empty: 26/250	2
500	min=0, max=27, avg=6.3, empty: 54/500	2
1 000	min=0, max=35, avg=6.0, empty: 148/1000	2
10 000	min=0, max=38, avg=6.3, empty: 1312/10000	1
Full (89 972)	min=0, max=75, avg=6.4 empty: 11826/89972	1

Table 5.9: Summary of Dependent and Independent Variables during Summative Experiments

Step	Independent Variables	Dependent Variables
Step 1	Label Budget, Training Strategy	mAP, Time-to-train
Step 2	Training Strategy	mAP, MAE, Time-to-train, Peak VRAM

Data Efficiency Metrics To evaluate to what extent the pretraining strategy lowers the dependency on annotated samples, it is common to measure label efficiency by leveraging some percentage (e.g. 1%, 10%) or fixed amount of samples from the labeled dataset and comparing performance[80, 15, 36]. This can be done either by training for a limited amount of epochs or until convergence.

The **Performance-Label Curve** is a visualization of the performance achieved for varying label-budgets, where the performance (e.g. mAP) is plotted against the amount of labels. This can show potential gains when the label-budget is limited.

5.5.2 Final Training and Model Configurations

Table 5.10 presents the final training and model configurations used in the final demonstration experiments.

Table 5.10: pretraining parameters of the chosen SSL-model

Parameter	**Value**
Masking Strategy	Block-Based
Masking Ratio	0.5
EMA Update Rate	0.998
Predictor Depth	2
Predictor Embedding Dim	128
Predictor Architecture	Self-Attention + Decoys

Table 5.11: Final grid size and model configuration (Baseline and SSL)

Component	**Parameter**	**Value**
Grid	X range (m)	[-60,60]
Grid	Y range (m)	[-60,60]
Grid	Z range (m)	[-5,5]
Grid	Resolution (m)	[0.2x0.2]
Encoder	pfe Hidden Dim	128
Encoder	Embedding Dim	512
Encoder	Model Depth	5
Encoder	Number of Heads	8
OD-Head	Embedding Dim	256
OD-Head	Depth	2

The grid size defines the limits of the grid, but in practice for supervised training, the loss is only calculated on predictions that are made in the field of view, meaning positive y-direction with a 120 degree angle, as can be seen in Section 5.4.2.2.

6

Results

6.1 Formative Results: Iterative Development

This section covers the results from the design and development stage of the DSR process, divided by the 3 design cycles.

6.1.1 Cycle 1: Infrastructure and Baseline Model

The primary goal of the first design cycle was to establish a robust, fully supervised baseline, and to establish an infrastructure that could be reused in the self supervised pipeline. Since the desired objective was a robust pipeline, we conducted a robustness ablation study to verify that we had reached the objective of Cycle 1.

6.1.1.1 Pipeline Robustness Evaluation

To ensure that the model was not too sensitive to changes in parameters, we conducted a systematic ablation study. Starting from an initial heuristic configuration (Run 0), we varied individual parameters one at a time across eight training runs. We utilized the full training dataset and a grid size of 100x100, and trained each version for 10 epochs. After each training run, we measured the performance and added the next modification. Note that each modification was cumulative, and changes were measured relative to the previous run. This isolates the impact of the change, but potential combination effects cannot be excluded. However, the goal of this experiment was not to prove direct causality, but rather test that the model was not overly sensitive to any of these changes. Table 6.1 summarizes the quantitative impact of these variations on both downstream performance indicators and computational efficiency.

Table 6.1: Cycle 1 Ablation Study: Robustness test to see varying Parameter's effect on Performance and Compute. All runs were executed on 2x A100 GPUs.

Run	Design Modification	Peak mAP	Time/epoch (min), Peak VRAM (GB)
0	*Initial Baseline Configuration*	0.701	91.2 min, 0.57GB
1	Embed Dim: 128 → 256	+2%	[+41% Time, +19% VRAM]
2	PFE Hidden Dim: 64 → 128	+1.5%	[+3% Time, +66% VRAM]
3	OD Head Dim: 128 → 256	0%	[0% Time, 0% VRAM]
4	Backbone Layers: 3 → 4	-1.8%	[+1.8% Time, +1.8% VRAM]
5	Custom Loss Weights	+8.5%	No Effect
6	NMS Threshold: 0.5 → 0.01	+3.4%	No Effect
7	Grid Res (XY): $0.8m \rightarrow 0.2m$	-0.1%	[+13% Time, +78% VRAM]

6.1.1.2 Summary of Findings

Performance (mAP): In general, our results show quite small variations in mAP, where the biggest variation of 8.5% can be attributed to the change in loss function, which is not an architectural change, and 3.4% in the change of NMS-threshold which only guides the post processing and does not affect the prediction quality. This means that the total variation stemming from changing the model capacity is in the range (-1.8, +2)%. The grid resolution shows a minimal negative change of -0.1%.

Training Efficiency (Time/epoch), VRAM: The training time per epoch varied in the range of (0,+41)%, and the memory consumption remained constant or increased across the runs, varying in the range (0,+78)%. This was expected based on the increased complexity of the calculations. The results indicate that increasing the embedding dimension of the transformer has quite significant impact on both training time and memory consumption, while increasing the PFE dimension mostly impacts the memory. Grid resolution showed an impact on memory consumption, but going from 0.8m to 0.2m wide pillars results in 16 times more cells in the grid, so with that in mind the impact is not necessarily large.

Trade-offs: The results indicate that the performance is quite insensitive to changes in the model capacity. From these results, it seems that increasing model capacity yields relatively small performance gains compared to significant increases in epoch training time and memory consumption. However, since we only trained for 10 epochs, it is possible that the changes would be larger in later stages of training. Larger models usually need longer training before reaching convergence, but can also potentially reach a higher maximum performance. The same goes for using a finer grid, where it can take more time to train with a large number of pillars, but also allows the model to capture finer spatial details. Therefore these results are not intended as showcasing absolute impact on performance, but rather to show that the model steadily arrives to an mAP in the range around 0.7-0.8 within the first ten epochs, regardless of the selected capacity variations.

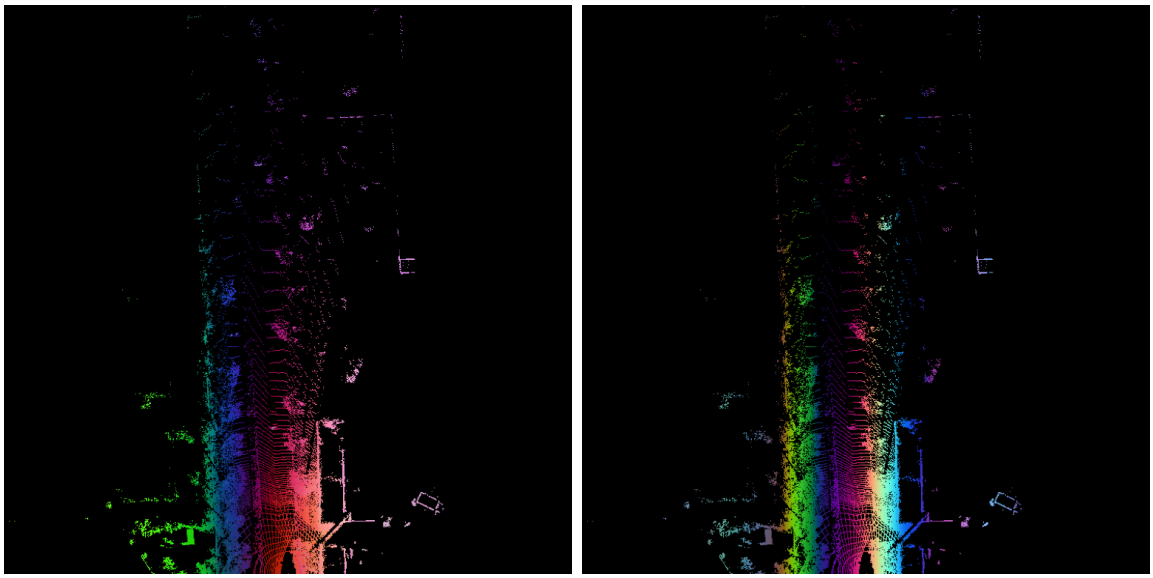
6.1.2 Cycle 2 - SSL Pipeline

The primary objective of this cycle was to build the SSL architecture using the developed components from cycle one. The representations learned were assessed by running a short frozen protocol after each epoch, which provided mAP and PCA visualizations.

Run 1: The initial configuration of the SSL model was based on the final configurations of the baseline model, SSL-specific parameters such as mask rate and EMA teacher update rate were set inspired by the I-JEPA set up.

Table 6.2: Initial configuration of SSL model

Component	Value
Grid size	150x150
Grid Resolution	0.2
PFE hidden dim	128
Embedding dimension	256
Model depth	4
SSL-specific parameters	
Masking strategy	Random
Masking ratio	0.7
EMA update rate	0.996
Predictor depth	1
Predictor embed. dim	256

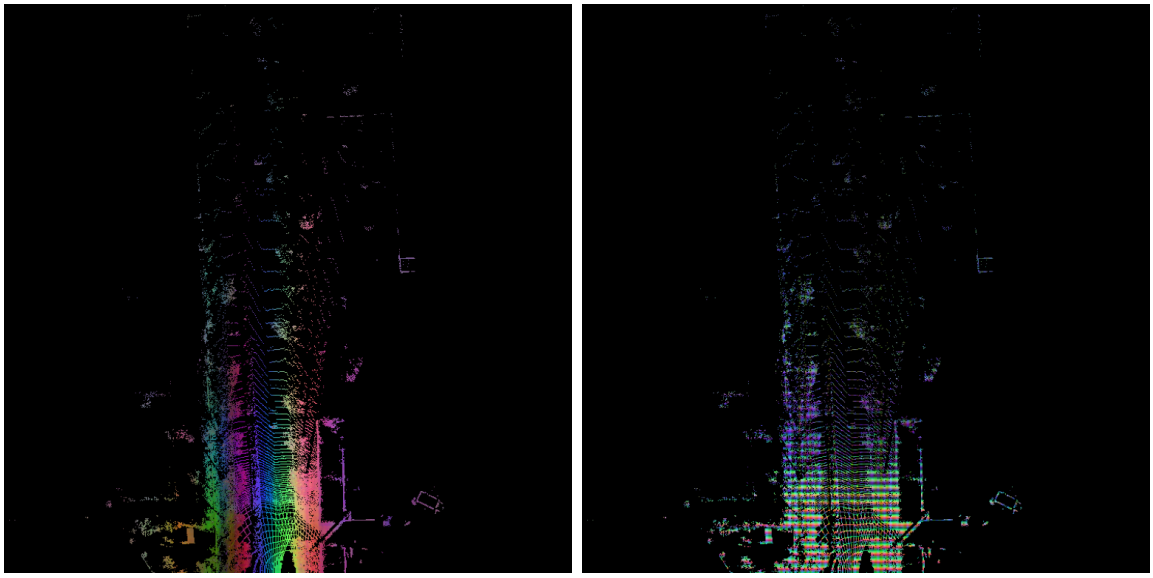


(a) PCA epoch 1 **(b)** PCA epoch 13

Figure 6.1: First iteration cycle 2: Ideally the with the PCA we should like to see some structure, where objects of the same class have the same color. E.g. all trees in one color, all cars in another color and the road a third.

The PCA visualizations consistently showed similar color gradient across all samples and epochs as shown in 6.1, suggesting the embeddings did not capture any depth information, resulting in a constant feature mapping along the y-axis. This was supported by the downstream task performance where mAP from the frozen protocol reached 0.116 at epoch 11 with only marginal improvement.

Run 2 Based on the observed collapse, several changes were introduced. The masking strategy was replaced with block masking using a 2-meter block size, to encourage the model to learn more structured spatial representations. Weight tying was added to the attention keys and queries with a ratio of 0.95, sharing parameters to encourage consistent spatial feature learning. The EMA update rate was increased from 0.996 to 0.999 for more stable teacher updates. Finally, gradient accumulation of 4 steps was introduced to get a larger batch size.

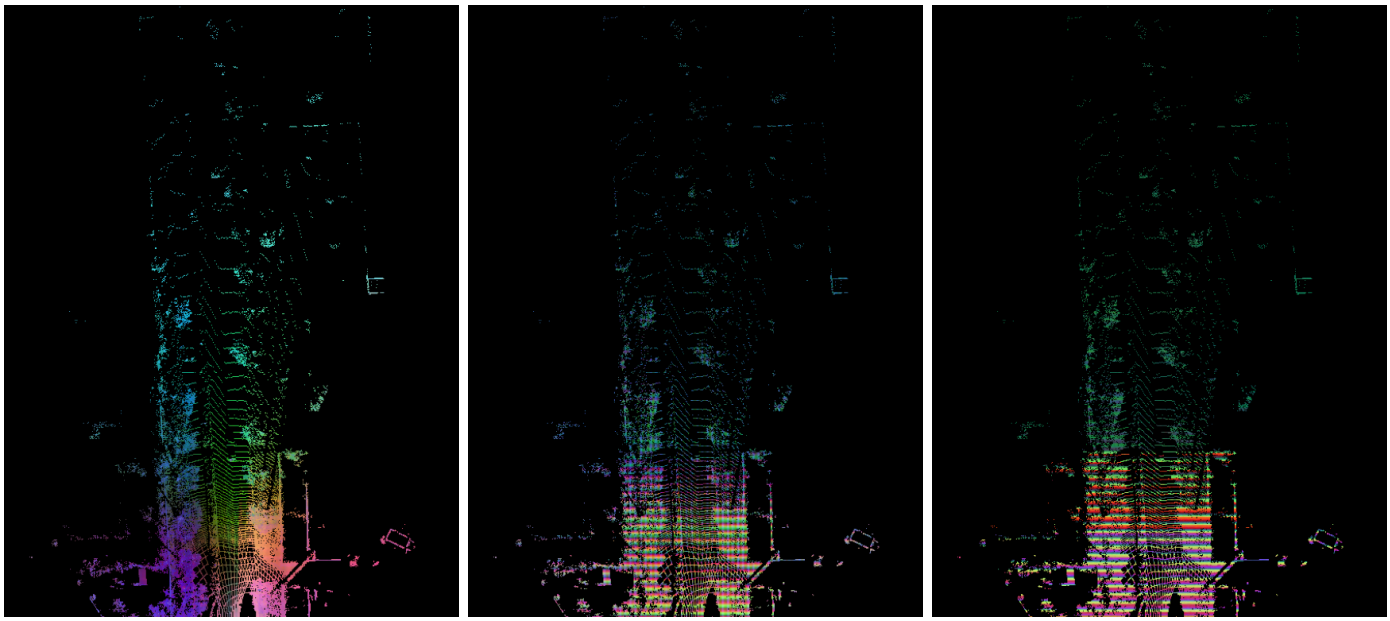


(a) PCA epoch 1 **(b)** PCA epoch 3

Figure 6.2: Second iteration cycle 2: PCA plots across epochs, visualizing collapse.

The PCA visualizations showed similar color gradient the first epoch, but from epoch 2 a horizontal stripe pattern started to emerge, suggesting the model had instead collapsed along the x-axis, assigning similar representations to all points regardless of their lateral position. This was supported by the finetuning results, where mAP remained low at 0.122 from epoch 3 and dropped to 0.024 from epoch 5, indicating that the learned representations did not transfer meaningfully to the downstream detection task and that the model collapsed with longer training.

Run 3: Given the persistent collapse in previous runs, we hypothesized that the pretext task might be too hard. To investigate this, the mask ratio was reduced from 0.7 to 0.5, giving the model access to a larger portion of the input.

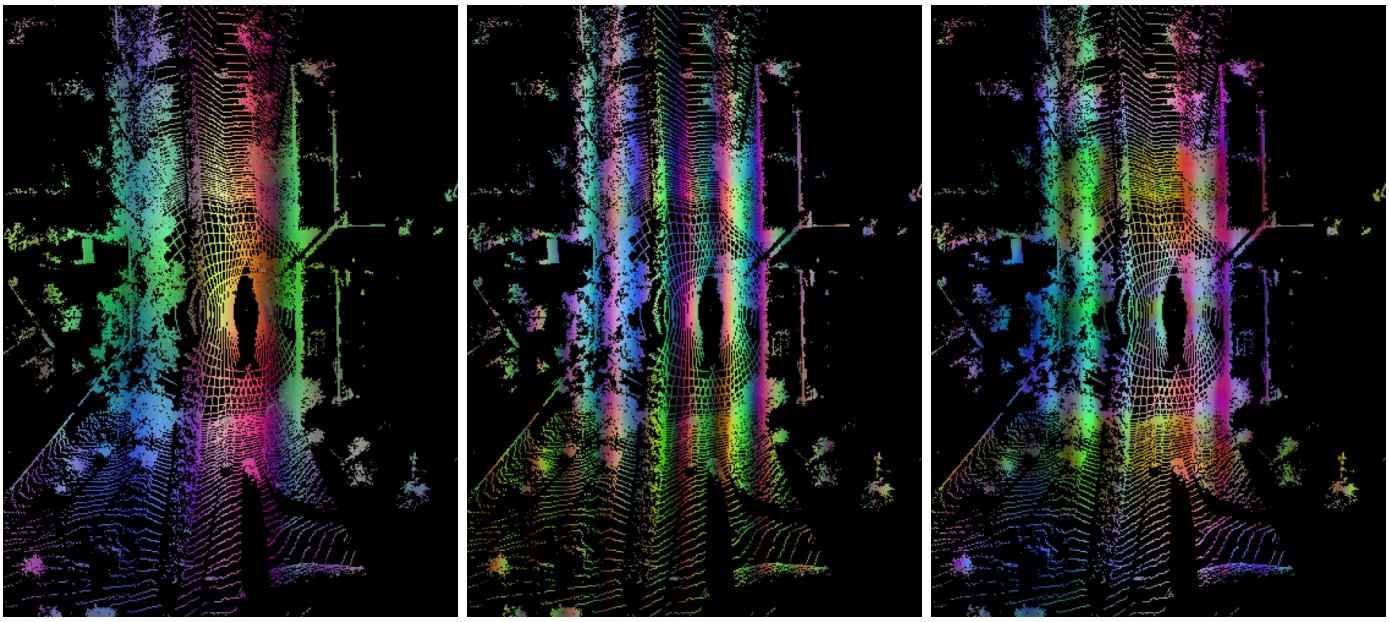


(a) PCA epoch 1 **(b)** PCA epoch. 5 **(c)** PCA epoch.15

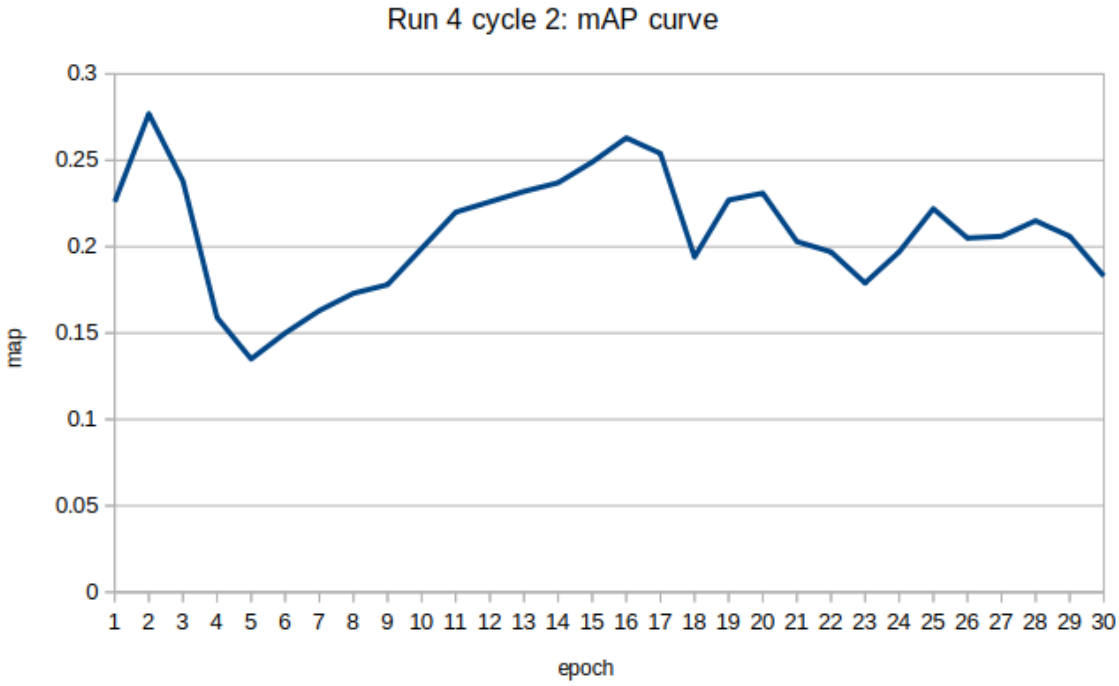
Figure 6.3: Third iteration cycle 2: PCA plots across epochs, visualizing collapse.

The finetuning results showed the highest mAP so far, reaching 0.23 at epoch 3, suggesting that the reduction in mask ratio had a positive effect. However, performance degraded significantly with further pretraining, dropping to 0.044 at epoch 7. This was consistent with the PCA visualization from a later epoch 6.3 , which still showed the horizontal stripe pattern indicating collapse in x dimension. This suggests the model becomes unstable with prolonged pretraining.

Run 4: The grid was reshaped from a forward-facing view to a symmetric view centered on the sensor, covering the full 360° point cloud, thereby utilizing more of the available LiDAR data during pretraining. While the pretraining utilized the full 360 view, finetuning was restricted to the field of view to match the annotation coverage of the dataset. The positional encoding was changed from pillar indices (i.e., the indices of the pillars in the grid structure) to world coordinates, providing the model with absolute spatial context.



(a) PCA epoch 1 (b) PCA epoch 2 (c) PCA epoch 16



(d) mAP

Figure 6.4: Fourth iteration cycle 2

The PCA visualization at epoch 1 showed the most spatially diverse representations observed so far, with rich color variation across both dimensions, suggesting no immediate collapse. However, the vertical stripe pattern began emerging from epoch 2 and persisted until epoch 16, indicating the embeddings were not capturing any depth information resulting in y-axis collapse developing during training. While

spatial diversity is desirable, the model should not only learn depth dependence but also more semantic correlations between similar object or scene regions. 6.4d shows that mAP started with a peak at 0.21 before declining around epochs 5-6, recovering to approximately 0.27 at epoch 16, and oscillating around 0.2 for the remainder of training, suggesting the model is unstable but not fully collapsing.

6.1.2.1 Summary of Findings

Across the four runs of cycle 2, the model showed initial indications of a functional SSL pipeline, with Run 4 achieving the highest mAP of 0.27. However, a recurring x-axis collapse was observed across runs, and the model exhibited instability with mAP degrading with prolonged pretraining. These results highlight the need for further development to improve stability and downstream performance, motivating the objectives of cycle 3.

6.1.3 Cycle 3 - Refined SSL-Model

The objective of cycle 3 was to improve the model's performance on the downstream object detection task, building on the functional SSL pipeline established in cycle 2. To address the instability and representational collapse observed in previous runs, several strategies were explored, including rotation and translation augmentations to encourage view-invariant representations, architectural modifications, and changes to the masking strategy. As many iterations were conducted in this cycle, this section presents only the most significant runs.

This cycle started with the configuration of the latest SSL model from cycle 2, as specified in Table 6.3 below.

Table 6.3: Initial configuration of SSL model at the start of cycle 3

Component	**Value**
Grid size	100x100
Grid Resolution	0.2
pfe hidden dim	128
Embedding dimension	256
Model depth	4
SSL-specific parameters	
Masking strategy	Block
Block size	2m
Masking ratio	0.5
EMA update rate	0.998
Predictor depth	1
Predictor embed. dim	256

Run 1: In cycle 2, the model repeatedly showed a positional collapse pattern as seen in 6.4b, suggesting that the network was taking a shortcut by relying on positional encoding rather than learning semantically meaningful features. Since positional information is directly provided as input to the encoder through the positional encoding and the input data itself, the model can learn to pass it through to the predictor, which in turn can use the known positions of the masked targets to produce predictions that minimize the loss without requiring a deeper understanding of the underlying content.

To investigate the role of positional encoding in this behavior, the first iteration of cycle 3 removed the positional encoding from the transformer encoder, while retaining it for the mask tokens. The hypothesis was that removing explicit positional information would force the model to learn content-based representations rather than relying on spatial shortcuts.

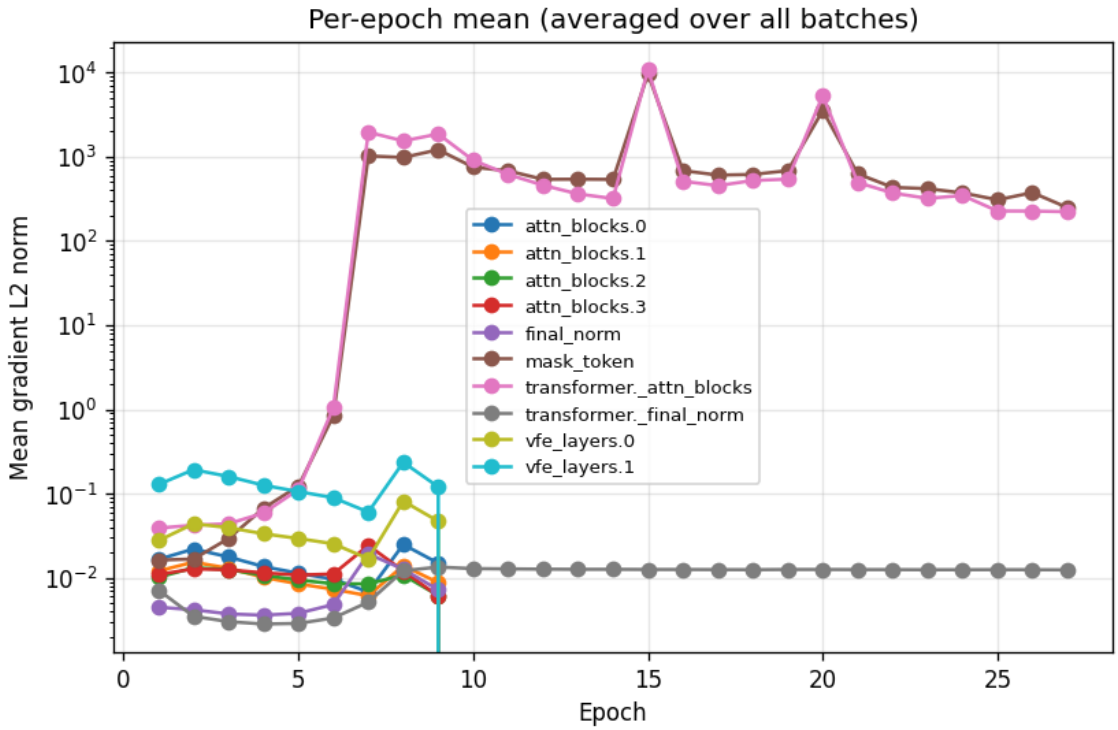


Figure 6.5: First iteration cycle 3: The gradient norm plot displays the per-layer, gradient magnitudes over epochs, where each line corresponds to a specific layer in the network, revealing how update signals are distributed across the model's depth. In this plot, attention blocks 0–3 and the final normalization layer belong to the student encoder, while the transformer attention blocks and transformer final normalization layer belong to the predictor

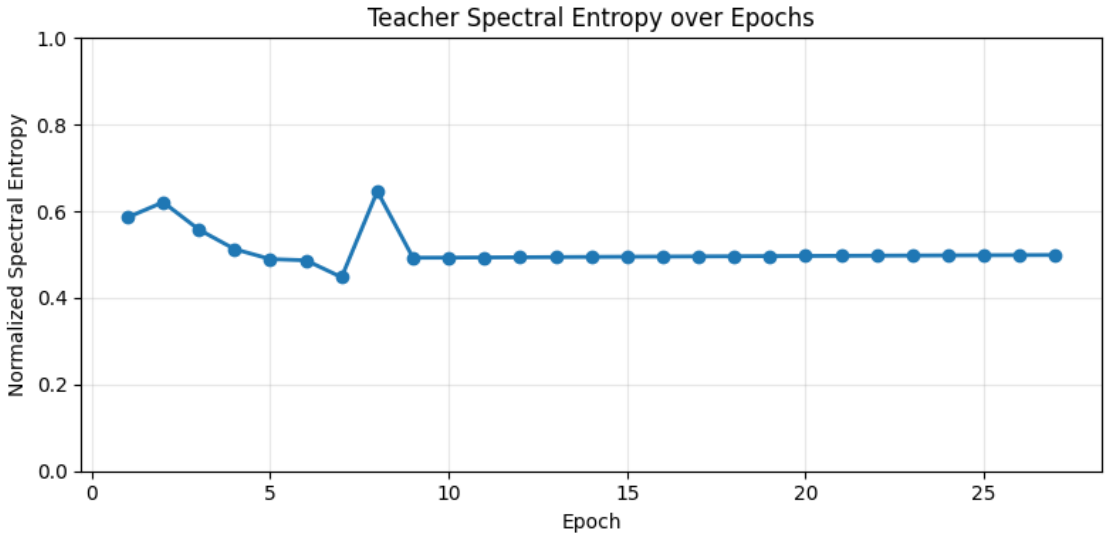
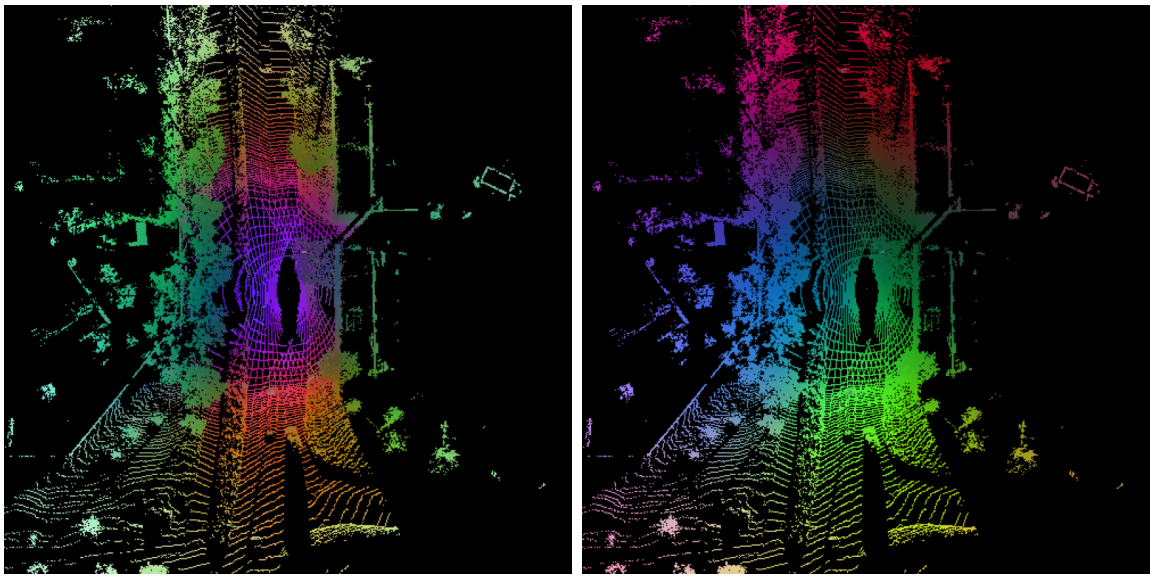


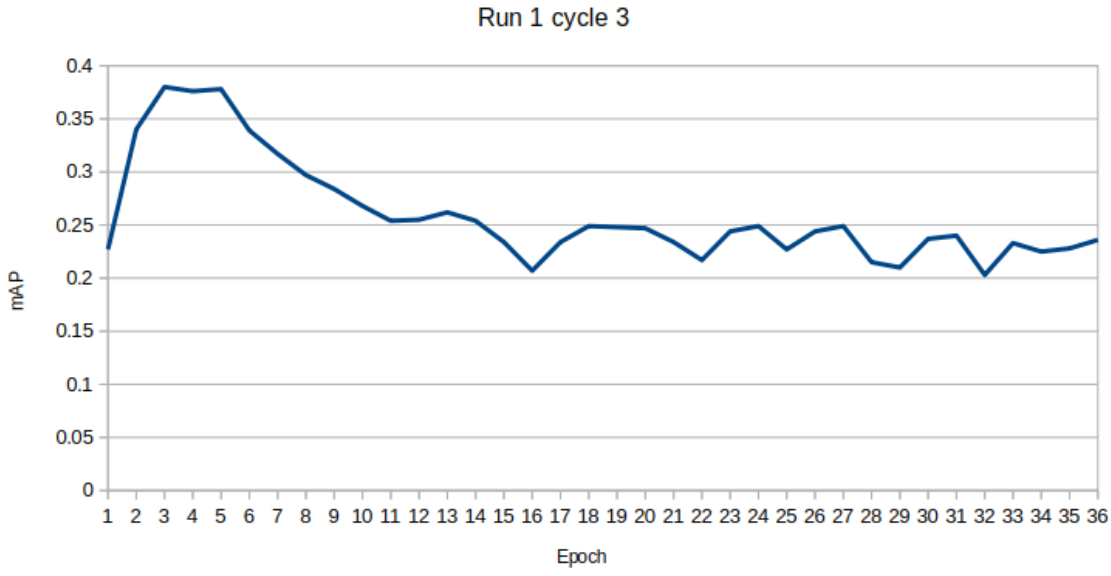
Figure 6.6: First iteration cycle 3: Teacher spectral entropy monitors the diversity of the teacher encoder's learned representations, where high entropy indicates the teacher is actively utilizing its available embedding dimensions, and low entropy signals dimensional collapse in the teacher's feature space.

Figure 6.5 reveals that this modification led to severe instability. As shown, several layers experienced either exploding or vanishing gradients, with most of the network's layers having near-zero gradient norms after a certain point in training, while a few layers exhibited extremely high values. This pattern indicates a collapse in gradient flow. The removal of positional encoding appears to destabilize gradients severely, leading to inconsistent updates. Furthermore, Figure 6.6 of the spectral entropy shows that the representations had converged to a low-level dimensional subspace indicating dimensional collapse.

Run 2: Given that the removal of the positional encoding showed a severe degradation in gradient flow, it was reinstated. To further address the dimensional collapse was to implement rotation augmentations to the point cloud before feeding them into the encoders. The point cloud was rotated in different angles for the student and the teacher input, resulting in two different views of the same scene (see Section 5.4.4.1 for details on the augmentation).



(a) ep. 5 **(b)** ep. 16



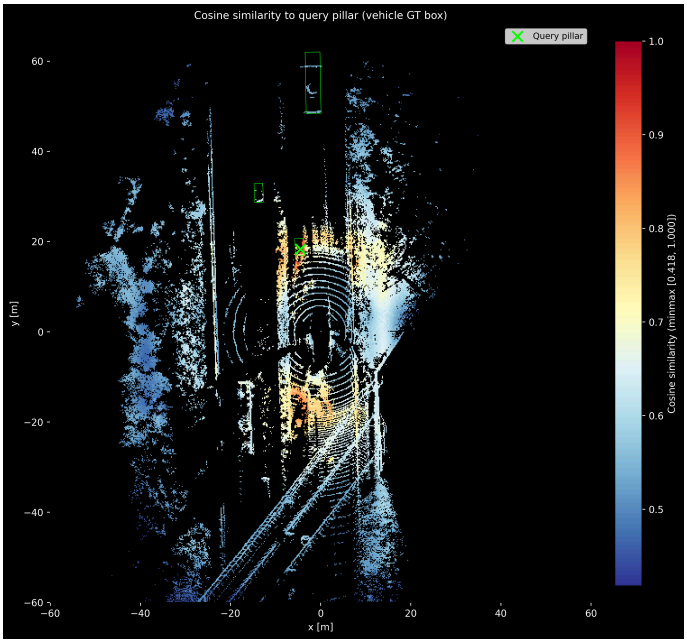
(c) mAP curve

Figure 6.7: Second iteration cycle 3

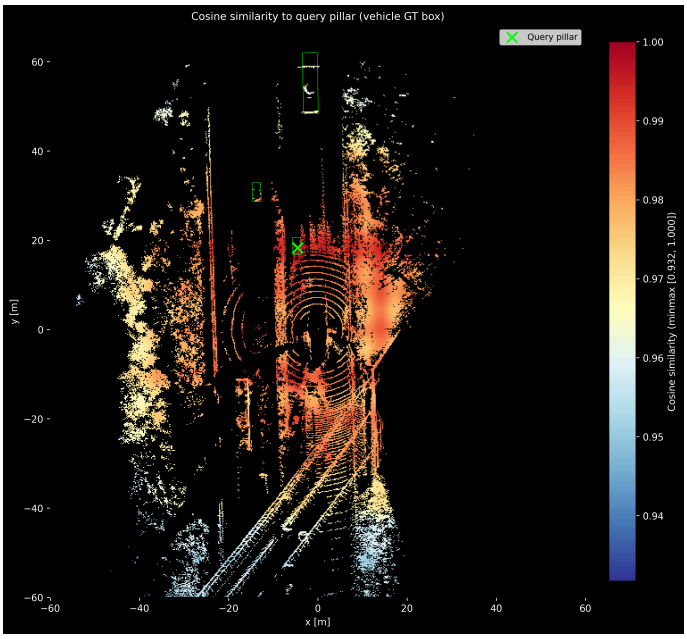
The rotation augmentation showed improved results in early epochs with a peak in mAP at 0.38 in epochs 3-5, before declining steadily to approximately 0.23 mAP

where it remained for the rest of training. The early peak indicates that the teacher network initially generates targets rich enough for the student to learn from. As training progresses however, the targets become less informative. While this approach resulted in degrading features and performance, the model avoided the complete x-axis or y-axis collapse observed in previous runs and achieved the best performance to that point. Therefore, the rotation augmentation was retained for the remainder of development.

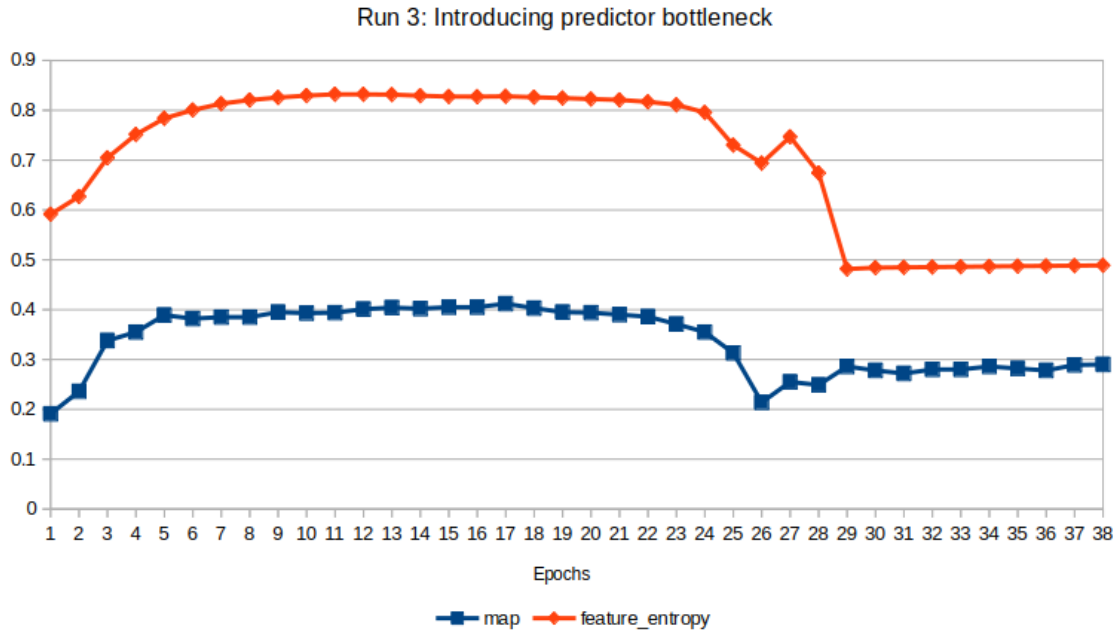
Run 3: Inspired by the I-JEPA structure we decided to decrease the dimensions of the predictor layer forming a bottleneck. We hypothesized that this could reduce reliance on the positional information from the mask token and therefore lessen the positional leakage and force the student to produce better representations.



(a) Cosine Similarity ep. 17



(b) Cosine Similarity ep. 26

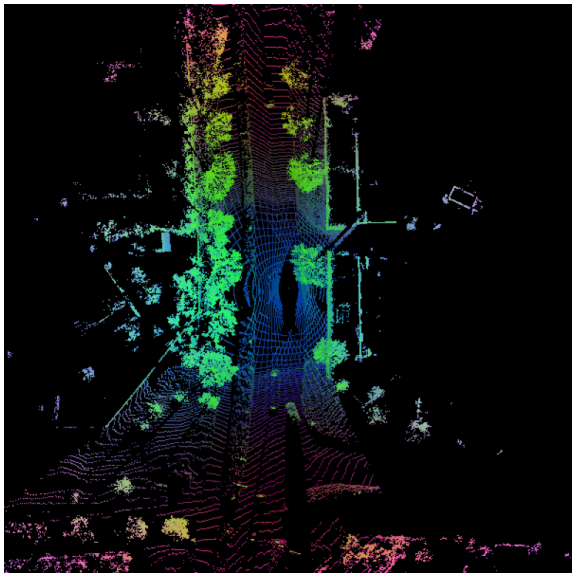


(c) mAP and Teacher Feature Entropy

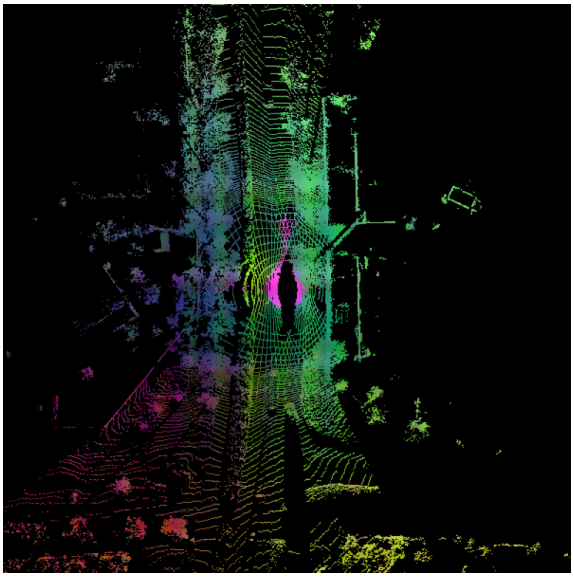
Figure 6.8: Third iteration cycle 3: In the cosine similarity plots (a) & (b), ideally we would see that the pillars that belong to the same object class as the query pillar, e.g. a car or a tree, were all red (high similarity core) while the rest of the scene was blue (low similarity score).

The introduction of the predictor bottleneck showed promising results in early training. Although performance improvement was modest, it reached 0.412 in mAP, the highest recorded to that point. However, the cosine similarity plots, which disolay the similarity in embeddings between a selected query pillar and every other pillar, display how the model's representations degrade between epoch 17 and 26. At epoch 17, similarity scores between the query pillar and the surrounding scene span from 0.418 to 1.0, reflecting the model's ability to distinguish between different scene elements. By epoch 26, this range has collapsed to 0.932–1.0, nearly every point in the scene receives the same score regardless of its semantic content. This aligned with the feature entropy that was relatively high until epoch 24 where a sudden drop appears followed by another drop at epoch 29 after which the entropy stays constant just under 0.5. All of this indicates that the model was no longer learning to produce any meaningful representations, collapsing and forgetting what it has learned.

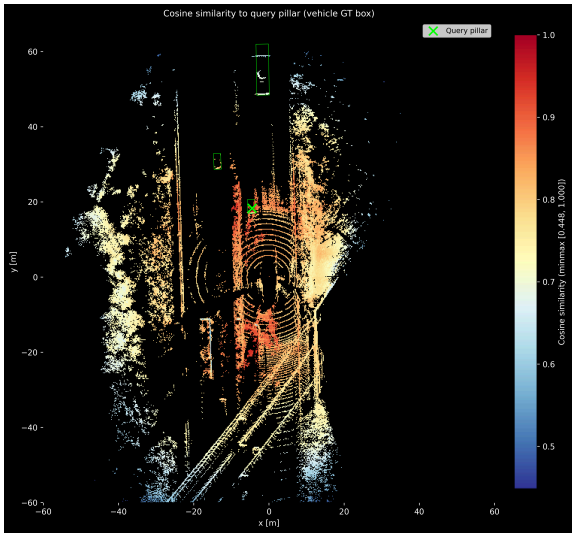
Run 4: Given that the single-layer bottleneck predictor showed the best early performance but suffered from representational collapse later in training, we hypothesized that increasing the predictor depth to two layers while maintaining the narrow 128-dimensional bottleneck could provide sufficient capacity to sustain learning without leaking positional information. Additionally, translation augmentation was introduced alongside rotation to further diversify the pretraining signal and discourage the model from relying on absolute spatial cues. After the rotation augmentation, the two rotated views were shifted by two different (x,y) offsets randomly sampled in the range [-10, 10].



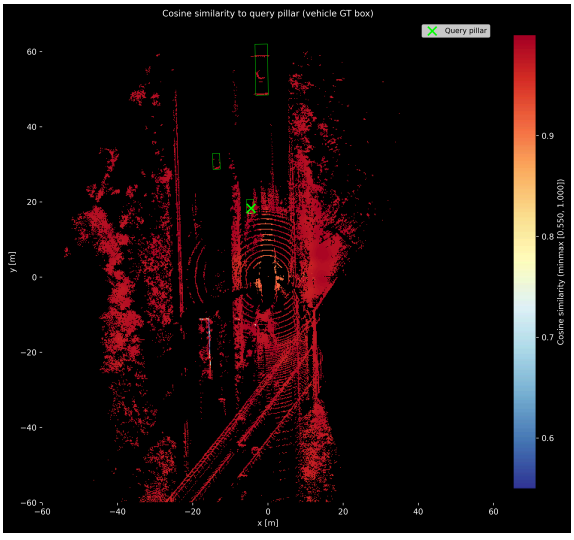
(a) ep. 28



(b) ep.32



(c) Cosine Similarity ep. 28

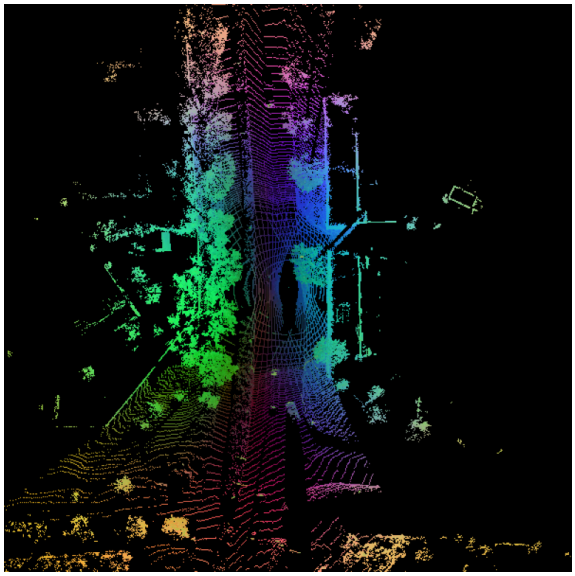


(d) Cosine Similarity ep. 32

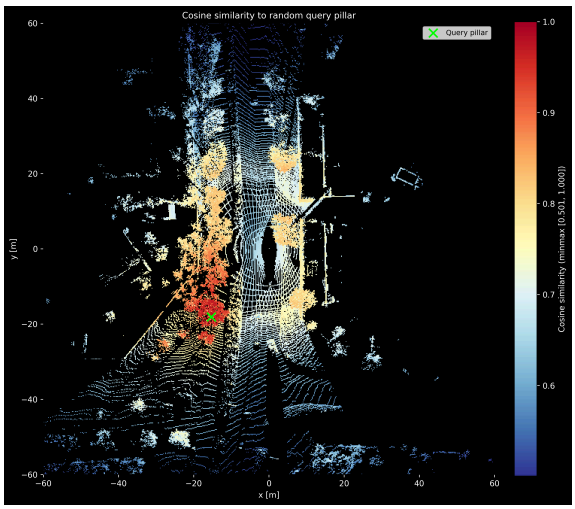
Figure 6.9: Fourth iteration cycle 3

At epoch 28, the features were more dispersed, indicating richer and more diverse representations. By epoch 32, the features appeared more clustered and less varied, suggesting a reduction in representational diversity. This shift reflects a collapse in the model's feature space, which negatively impacted its ability to distinguish between different classes of objects. The bottom row displays cosine similarity maps for the same epochs, visualizing the similarity of features to a reference point in the scene. At epoch 28, the similarity values are more distributed, but do not highlight clear semantic groupings. By epoch 32, the similarity becomes uniformly high, as shown by the dominance of red, reflecting a collapse in feature diversity. mAP and the teacher entropy showed very similar behavior as run 3 (Figure 6.8c).

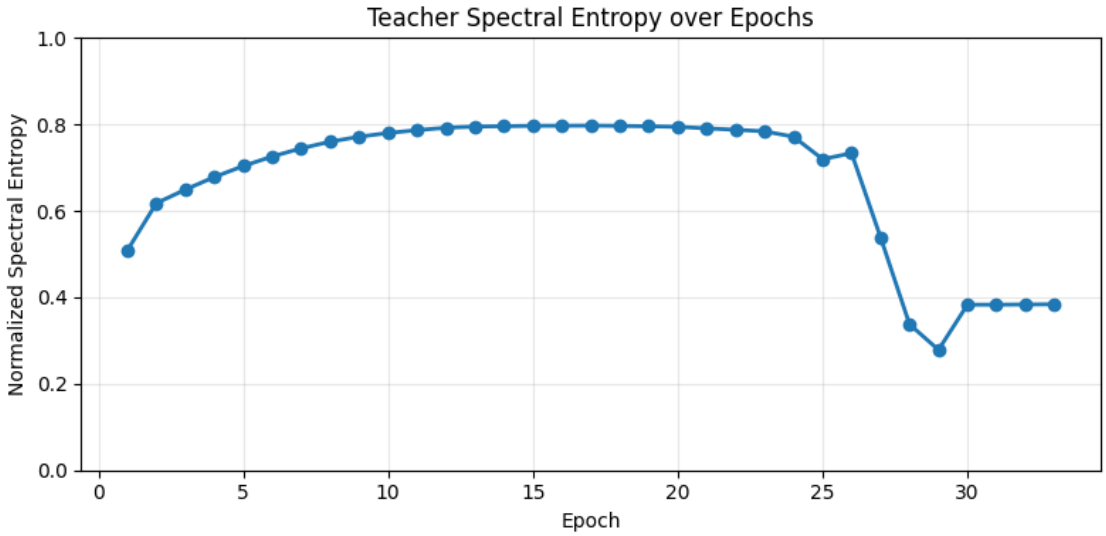
Run 5: Since the previous runs still exhibited collapse in later epochs, we hypothesized that making the pretext task more challenging would force the predictor to learn richer representations rather than defaulting to trivial solutions. Therefore, we introduced empty pillars that get fed in to the predictor as masked tokens (called decoy pillars), encouraging the predictor to no longer assume that all the masked pillars were occupied. However, loss was only applied to the occupied masked pillars as the decoys are not supposed to serve as a supervisory signal, only to remove implicit occupancy information. This was intended to prevent leakage of geometric information that would otherwise make the task trivially easy. The decoy pillars were sampled randomly among all the empty pillar neighboring occupied pillars with a ratio of 0.5 of the number of masked pillars. That makes 1/5 of the predictor input decoy pillars.



(a) PCA visualization ep. 9



(b) Cosine Similarity plot ep. 9



(c) feature entropy

Figure 6.10: Fifth iteration cycle 3

The cosine similarity plot 6.10b reveals that the decoy pillar strategy does introduce local geometric structure into the learned feature space. Pillars belonging to the same object got high similarity scores, while semantically distinct regions (e.g., the road surface) score lower. This suggests the model has begun learning discriminative features. However, the similarity scores decay with distance even for semantically identical structures, a tree further away receives a low score despite being structurally equivalent to the query tree. This points to the model encoding positional/range bias rather than purely semantic content.

Furthermore, despite the improved structure at epoch 9, the entropy plateau before a sharp collapse after epoch 26 from which it does not recover. This suggests the decoy pillars help prevent shortcut learning early on, but are insufficient to maintain a strong learning signal throughout a longer training.

Run 6: As the introduction of decoy pillars seemed to improve the representations at an early stage but was insufficient in the long-run, we increased the decoy ratio to be equal to the number of masked pillars. We hypothesized that this would strengthen the effect introduced by the decoy pillars in the previous run. Since Run 5 still collapsed in later epochs, we reasoned that a decoy ratio of 0.5 was not enough, the masked pillars were still mostly occupied, so the model could still partly use the mask as a hint for occupancy. By increasing the ratio to 1.0, occupied and empty masked pillars appear equally often, making it impossible for the model to assume a masked pillar is occupied.

The block mask sampling was modified to only sample from occupied pillars, rather than from the full BEV grid. Previous runs, the block were sampled randomly from the full grid resulting in block potentially falling entirely on empty space. This causes the effective masking ratio of occupied pillars to vary significantly between scenes depending on how much empty space was sampled. By restricting sampling to occupied pillars, the masking ratio is guaranteed to be consistent across scenes, making the pretext task equally difficult across samples. Furthermore the block size was increased from 2x2m to 6x6m, we hypothesized that this would provide a stronger and more sustained learning signal, helping to maintain meaningful representations throughout training.

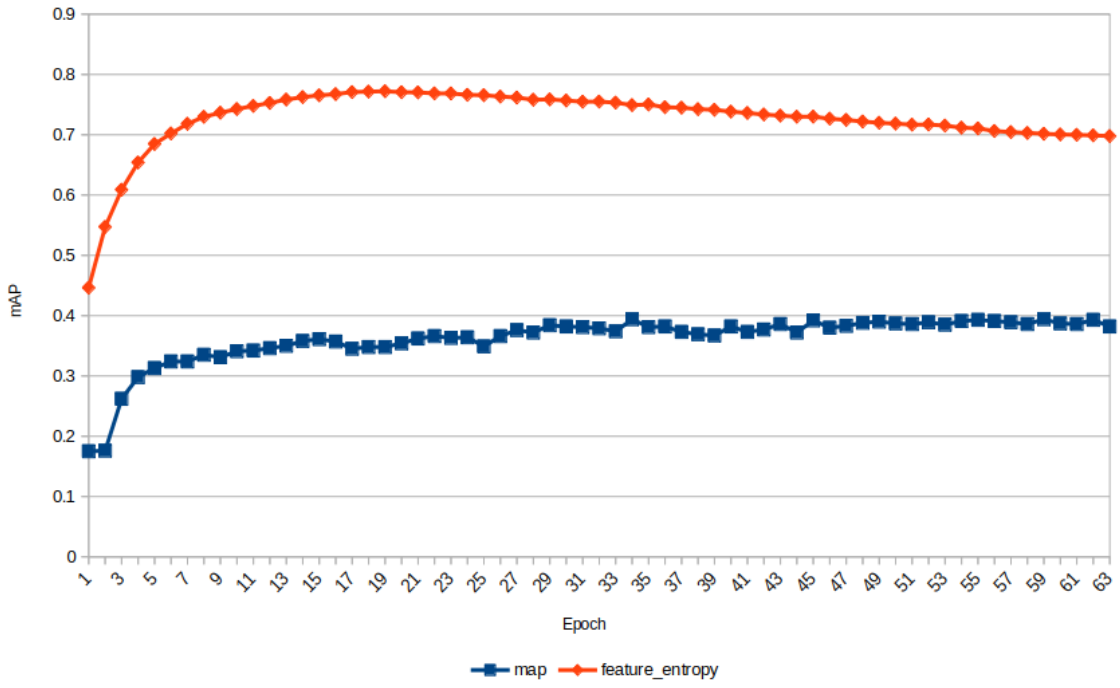
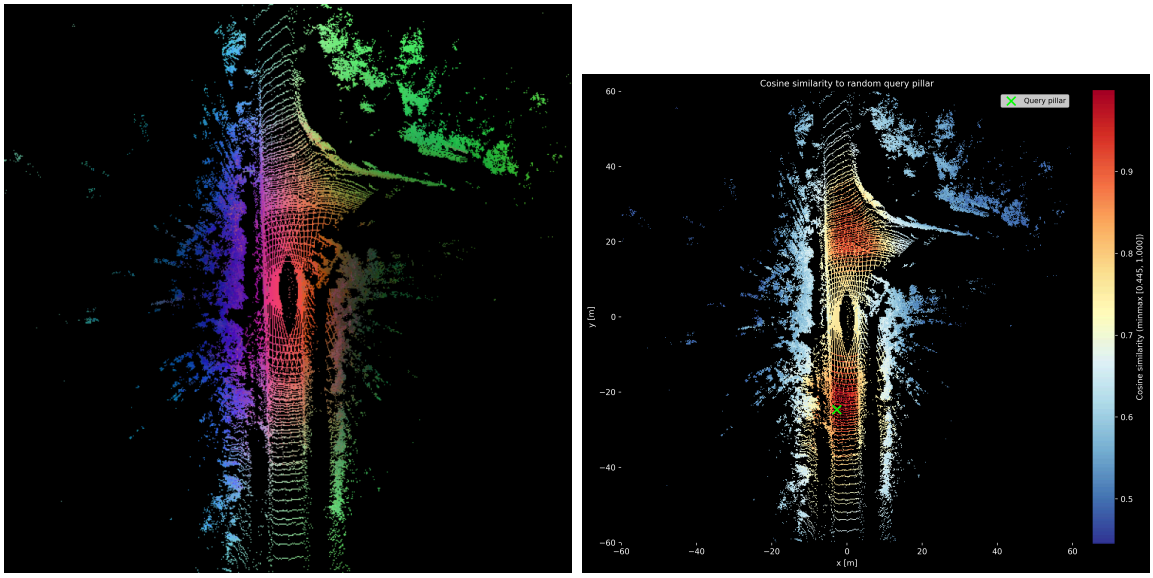


Figure 6.11: Sixth iteration cycle 3: map & feature entropy

This run demonstrated stable training dynamics, with both the performance and teacher spectral entropy increasing steadily during the early epochs before plateauing. Unlike the previous run, no collapse is observed, as the spectral entropy remains at a relatively high level throughout training. However, despite the stability, the model converges to a modest mAP of approximately 0.39, with minimal improvement after epoch 15. This suggests that while the learned features were diverse, they were not sufficiently informative for the downstream task.

Despite the improvements introduced in Run 6, subsequent experiments varying the masking parameters showed no meaningful improvement. We tested increasing the block size from 6 to 8, reducing the decoy ratio, increasing the masking rate to 0.7, and combinations thereof. In all cases, mAP and feature entropy demonstrated the same behavior as observed in Figure 6.11, suggesting these parameters have little influence on the fundamental training dynamics at this stage.

Run 7: The visualizations still showed signs of prioritizing position rather than semantics of the scene. Therefore, we hypothesized that the self-attention mechanism in the predictor transformer could silently cheat by leaking information about the masked token into the context tokens when all tokens attend to each other, which the predictor could exploit to find a shortcut. To test our hypothesis we implemented a cross attention based predictor to sever this communication, ensuring that masked tokens attend to context tokens only. This eliminates the need for decoy pillars as the mask tokens cannot attend to one another, so they were not included in trainings with cross-attention.



(a) PCA visualization epoch 25 **(b)** Cosine Similarity plot epoch 25

Figure 6.12: Seventh iteration cycle 3

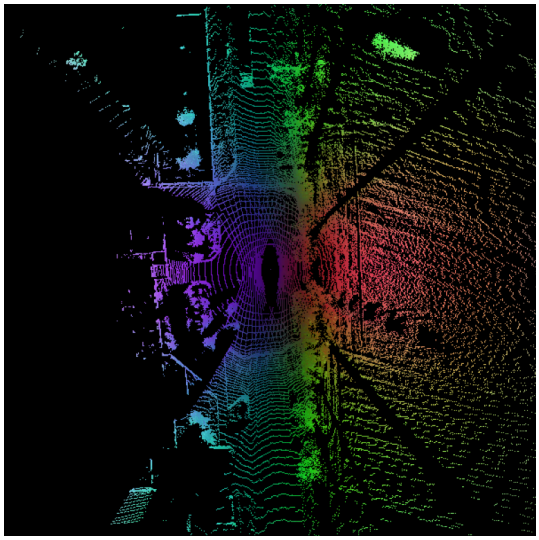
The results show no improvement compared to previous runs, but performance reached a similar result of a peak mAP around 0.38 at epoch 25. This indicated

that this strategy was a viable alternative to using self-attention + decoy pillars.

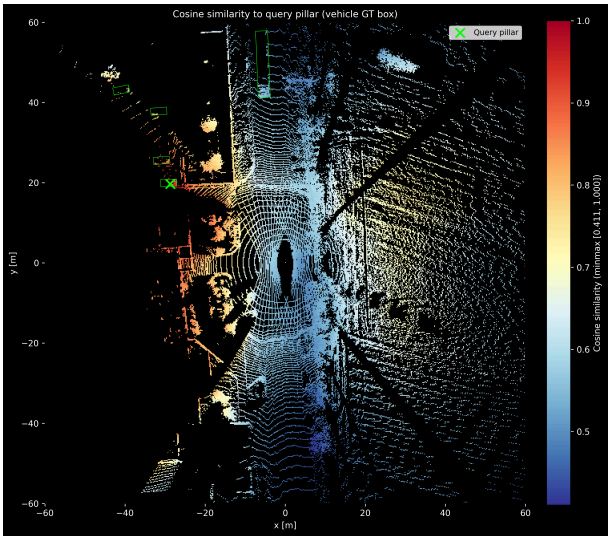
However, there was a notable change in compute efficiency between leveraging cross-attention, due to the matrix multiplications in the predictor-transformer are done between mask tokens and context tokens only. We observed a reduced training time per epoch of about 24%, as well as reduced peak memory consumption from 4.12 GB to 3.16 GB.

Run 8: Given that neither masking strategy variations nor predictor architecture changes produced any improvement, we shifted the focus to the encoder itself. If the encoder lacks sufficient capacity to capture meaningful spatial structure from the pillars, no amount of tuning the pretext task or predictor design can compensate for this. While developing the baseline object detection model, the robustness ablation study indicated that scaling did not significantly influence performance. However, the supervised object detection task is quite simple compared to the SSL-task, where the model needs to reason about the entire space instead of just vehicles.

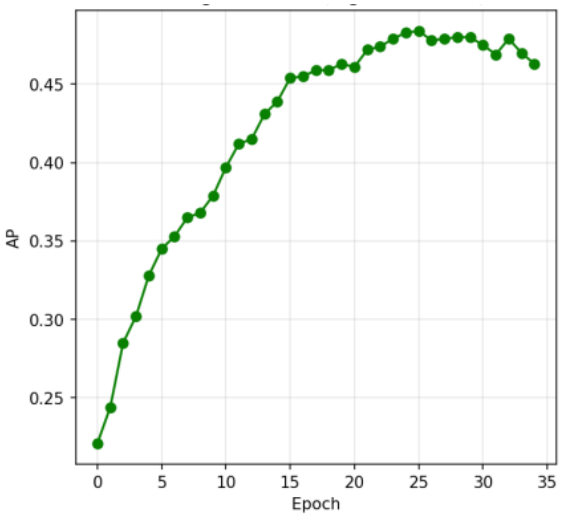
To test this hypothesis, we scaled up the encoder by increasing the embedding dimension to 512, the number of attention heads to 8, and the number of layers to 5, resulting in a significantly more expressive model than used in previous runs. All other settings were kept consistent with Run 6 (block size 6, masking rate 0.5, decoy ratio 1.0). We hypothesized that if the previous runs had been limited by encoder capacity, this scaled-up model should produce richer intermediate representations and consequently higher feature entropy and improved downstream performance.



(a) PCA visualization ep. 25



(b) Cosine Similarity plot ep. 25



(c) mAP plot

Figure 6.13: Eighth iteration cycle 3

Increasing the encoder dimensionality resulted in a meaningful improvement in mAP reaching approximately 0.48. To make sure that the gain in mAP could be strictly attributed to the improvements in training strategy, we also trained an OD-head on the same configurations but a randomly initialized backbone, and achieved a mAP of approximately 0.19. This showed that the OD-head itself cannot achieve that performance without the underlying knowledge of the backbone. While the mAP showed improvements (Figure 6.13c, PCA and cosine similarity plots are still not showing any clear structures across object instances. This suggests that while the

representations do contain some meaningful information, there is still some underlying issue making the model favor position.

Additional runs In addition to the presented experiments and results, we did a number of other tests that did not demonstrate any significant changes. A noteworthy one was that we once again tried to remove the positional encoding to see if it could be the reason to why the model still seemed to prefer encoding position. In contrast to the first attempt of removing positional encoding that resulted in severe gradient instability, this time around it only slightly reduced the performance.

We also tried to make the encoder deeper, including runs with 12 layers instead of 4, which produced similar results both with self-attention and cross-attention based predictors. Decreasing the masking rate to 0.4 showed no relevant impact, and increasing or decreasing block size from 6 to either 4 or 8 was also not effective.

6.1.3.1 Summary of Findings

The consistency of results across a wide range of configurations suggests that the observed limitations are unlikely to stem from any single parameter or architectural choice, but may instead point to a more fundamental challenge in the training setup. This interpretation is supported by the PCA visualizations, which reveal limited separation between object instances and scene regions across runs, and seem to rather favor distance separation. However, the PCA plots only represent the three most dominant feature dimensions, and even though the position might dominate the information, it does not rule out that the representations do carry semantically meaningful information that is just less prominent.

The performance of our best performing model suggests that the model has learned something valuable, since an mAP in the range up to 0.48 is highly unlikely to have been learned only from the shallow MLP-head. We tested this claim by training on a randomly initialized encoder, which only reached a score of 0.19. This supports our reasoning that the pretraining strategy results in representations that aids significantly with the object detection task.

In terms of training efficiency, the notable result besides changing model capacity was switching the predictor architecture to cross-attention, which reduced training time by 24%. However, it was not investigated further as it yielded no other improvements. Other variations such as masking strategy did not have notable impact.

6.2 Summative Results: Final Demonstration

This section presents the experimental results from the two-step summative evaluation described in section 5.5. Step 1 investigates the effect of label budget on model performance, addressing RQ3. Step 2 provides a full comparison between the best-performing pre-trained SSL model and a supervised baseline trained under identical conditions, addressing RQ1 and RQ2.

6.2.1 Step 1: Label Budget Experiments

To evaluate the utility of self-supervised pretraining in the low-label regime, the pre-trained model with an object detection task head and the supervised baseline were trained to convergence on label budgets ranging from a single annotated frame to the full training set. First, we report the results from freezing the encoder, and following are the results from fine-tuning. All runs share the same training splits and 2 000-sample evaluation split, as well as the same convergence criterion (five consecutive epochs without improvement).

6.2.1.1 Detection Performance vs. Label Budget

Table 6.4: Resulting mAP of the label budget experiments. Δ is the absolute difference (SSL − Baseline).

Training Samples	mAP (Frozen SSL)	mAP (Baseline)	Δ mAP
1	0.12	0.008	+0.112
10	0.276	0.02	+0.274
100	0.398	0.167	+0.231
250	0.439	0.395	+0.044
500	0.47	0.524	−0.054
1 000	0.485	0.576	−0.091

Figure 6.14: Detection performance for the frozen SSL and the baseline as a function of label budget (log-scale x-axis).

The graph shown in Figure 6.14 corresponds to the results in Table 6.4. The results show a significant advantage of the SSL model in the low label-budget regime even when the pre-trained encoder is frozen. It outperformed the baseline up to 250 samples, and is then surpassed by the baseline at 500 samples. This indicates that the SSL model learned more informative representations than what can be learned by labeled objects when annotations are scarce.

Table 6.5: Resulting mAP of fine-tuning on limited label budgets. Δ is the absolute difference (SSL − Baseline)

Training Samples	**mAP (Fine-tuned SSL)**	**mAP (Baseline)**	**Δ mAP**
1	0.158	0.008	+ 0.15
10	0.34	0.02	+ 0.32
100	0.453	0.167	+ 0.286
250	0.502	0.395	+0.107
1 000	0.637	0.576	+0.061
10 000	0.767	0.757	+0.01

Figure 6.15: Detection performance of the fine-tuned SSL and the baseline as a function of label budget (log-scale x-axis).

Fine-tuning was conducted by applying a learning rate (0.1*LR) to the backbone, unfreezing the encoder after one epoch for the 1- and 10-samples budgets, and after five epochs for remaining runs. The learning rate was small to not stray too far from the pre-trained weights, therefore the fine-tuning protocol in general required more iterations than the baseline to reach the defined convergence, which may have allowed for a smoother peak.

The fine-tuned SSL-model shows an even greater advantage than under the frozen protocol, and surpasses the baseline model up to 10 000 samples as shown in Figure 6.15.

6.2.2 Step 2: Full Dataset Comparison

In the second step, both models were trained on the full training set of 89 972 samples and evaluated on the complete evaluation set of 10 023 samples.

6.2.2.1 Detection Performance

Table 6.6: Detection results on the full training and evaluation set (a high mAP and low MAE is desired).

Metric	SSL(Frozen)	SSL(Finetuned)	Baseline
mAP	0.55	0.835	0.83
MAE (Scale)	0.199	0.141	0.138
MAE (Orientation)	0.557	0.139	0.136
MAE (Translation)	0.702	0.247	0.249

The results show that the supervised baseline model consistently outperforms the SSL-model trained with the frozen protocol on all metrics when the full annotated training dataset is available. The finetuning of the SSL reaches near-identical performance.

6.2.2.2 Computational Efficiency

Table 6.7 compares the two training strategies in terms of wall-clock training time and peak GPU memory VRAM usage. Training time was measured as aggregated time per epoch, excluding data loading, evaluation, and logging.

Table 6.7: Computational efficiency while training with the full training dataset on 4 GPUs.

Metric	**pretraining**	**SSL(Frozen)**	**SSL(Finetuned)**	**Baseline**
Training time (h)	71.7	6	67.8	35.1
Epochs	25	18	45	19
Peak VRAM (GB)	8.0	1	4.87	5.0

The results show that the total training time when aggregating pretraining and probing resulted in a total of 77.7 GPU hours, while the supervised baseline had a training time of 35.1 hours. The total required VRAM for the SSL model was 8GB and 5GB for the baseline, which means that aggregated, the supervised training was more efficient. However, if we treat the pretraining separately, the task head only required 6 training hours and 1GB VRAM, both of which were significantly lower than the baseline. The finetuning required more epochs to reach convergence, resulting in a longer training time of 67.8 hours even though training time per epoch was shorter than the baseline.

6.2.2.3 Summary of Full Dataset Results

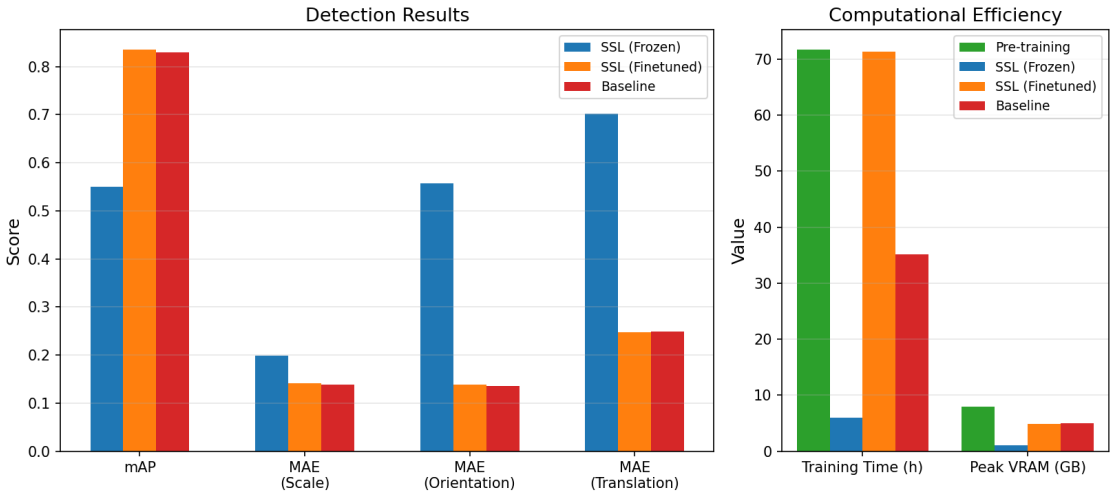


Figure 6.16: Detection performance and computational resource consumption for the full dataset trainings.

An important note is that convergence was determined by a peak in mAP followed by 8 consecutive epochs without further improvement. The frozen protocol and the baseline required about the same amount of epochs to reach this threshold, while the fine-tuning took substantially more iterations before fulfilling this criterion. This was expected since the learning rate for the SSL encoder was significantly lower. This should be taken into consideration when interpreting the results, as no optimization effort was made in terms of training hyper-parameters, and the results, especially in terms of training time, are therefore not strictly comparable, but still indicative.

7

Discussion

7.1 RQ1: Design Decisions and Representation Quality

Developing a self-supervised learning pipeline for LiDAR perception requires making a series of engineering decisions. Exhibited by the iterative development process (cycle 2 and 3) finding a way to make the model learn good representations was not a straightforward path.

As described in Section 5.1.1.1, we view our first research question from three different perspectives. Firstly, we wanted to examine "*How design decisions influence the quality of learned representations*" and secondly, "*how these effects can be interpreted in terms of robustness, maintainability and scalability of the perception component*", divided into internal effects from design variables, and system-level effects of SSL as training strategy.

7.1.1 Effect of Design Decisions on Representation Quality

This section examines the most significant design decisions made throughout the DSR process, how these shaped the representations ultimately learned by the model.

For several early experiments, the model produced near-zero mAP that consistently collapsed regardless of other configuration changes, such as adjustments to masking ratio, block size or architectural changes. The first sign of learning came from the introduction of the rotation augmentation. This single change improved downstream performance to approximately 0.38 mAP, and the PCA visualizations, which had previously exhibited a striped collapse pattern, began showing more distributed feature structure for the first time. The implementation of data augmentations prevents the model from simply encoding absolute point positions into its features, as doing so would generate high loss because the teacher and student absolute positions used to produce the encoded features differ from each other.

Another significant finding concerned how masked blocks were sampled. Initially, blocks were sampled across the full grid, meaning patches could land on completely empty space. This caused the effective mask ratio to vary across samples, making

the difficulty of the pretext task inconsistent between training steps. Switching to sampling larger contiguous blocks containing nonempty pillars stabilized this, and the training dynamics improved noticeably as a result. Later epochs no longer showed signs of severe collapse.

Predictor size had a notable impact on both performance and learning dynamics. Reducing the dimensionality from 256 to 128 improved downstream mAP while also delaying collapse. A predictor with low capacity cannot compensate for weak encoder representations, thereby forcing the backbone to learn more abstract, semantics-focused features. This is consistent with Garrido et al. [22], who show that weaker predictors produce more invariant representations that perform better in the frozen protocol evaluation. The collapse dynamics further reinforce this. The larger predictor showed earlier signs of collapse, suggesting the model found a shortcut through the predictor rather than developing stable encoder representations. The smaller predictor delayed this, allowing the model to train more stably.

As we hypothesized that self-attention in the predictor could pose a potential leakage of the mask tokens' positions into the context tokens, we explored two different strategies to mitigate this: decoy pillars and cross-attention. The decoy pillars served to prevent the model from short-cutting to assuming all pillars are occupied and predicting a similar feature vector for all masked pillars. Preserving decoys was supported by early indicators, that demonstrated initial formation of geometric structure, but the model could not build on this throughout the training. This observation implies the constraint lies elsewhere, possibly due to the inherent difficulty of the pretext task in this data domain. cross-attention was explored as an alternative architecture to mitigate potential positional leakage, as the context tokens no longer attend to the mask tokens. The self-attention setup performed slightly better, but both setups approached 0.4 mAP, suggesting that the choice of attention mechanism had limited impact on representation quality.

Beyond architectural choices, the definition of the pretext task itself shapes what the encoder is driven to learn. The results in cycle 3 show that variations in the masking rate and strategy did not produce significant differences in downstream performance. This is notable given that the definition of the pretext task is widely considered one of the most meaningful design decisions in SSL. Garrido et al. [22] emphasize that the pretext task must be sufficiently difficult for the model to learn meaningful representations, and that too simple a task fails to drive useful learning in the encoder. That this effect was not observable in our results may be because the variations we explored were limited. Given that the model was trained on complex driving scenes with large variation in the spatial scales of objects, perhaps the task was too simple to force the model to capture this complexity, instead allowing it to find shortcuts. Alternative explanations for the observed results include that the masking ablations may have been confounded by underlying training instability, the model may have had a strong positional bias, or the task may have been too difficult.

Increasing the encoder dimensionality produced notable improvement in downstream performance approaching 0.5. However, this came at a cost of a 48% increase in

training time per epoch. This observation suggests that the smaller model was capacity constrained, and that the representations it could form were insufficient to fully capture the complexity of the scene. Furthermore, the SSL-model must learn general scene representations, which is inherently more complex than learning features optimized for a specific task such as object detection. This aligns with findings in the SSL literature, where encoder performance has been shown to degrade when the model is too small relative to the dataset size and complexity [39].

The DSR process revealed that variables outside the pre-determined design space had a significant impact on both learning dynamics and downstream performance. Factors such as data augmentation strategy and decoy pillars, none of which were the primary focus of the cycles, proved to be meaningful contributors. This suggests that in novel SSL settings, the most impactful design variables are not always known in advance, and that an exploratory engineering process driven by diagnostics is useful.

7.1.2 Effect of Design Decisions on Software Quality

The iterative development process itself is an empirical observation about the engineering complexity of SSL. Multiple design cycles were required before the model produced any meaningful representations at all, and the variables that mattered the most were identified through diagnostics-driven iteration rather than the initial problem statement. The effects of changing them were largely up to subjective interpretation. This suggests that SSL pipelines carry a higher upfront engineering cost than equivalent supervised pipelines, and that this cost should be factored into any practical assessment of the approach.

When focusing internally on the pretraining pipeline design, a notable maintainability challenge stems from the limited interpretability of the effects of different designs. As the model's behavior is largely of black-box nature, it is hard to identify the source of potential failures. The same way we found that an exploratory engineering process was appropriate, the diagnostic effort also requires exploratory investigation, rather than targeted fixes, making the system difficult to maintain. This is true also for supervised approaches, but as the SSL training pipeline consisted of a larger number of components and processes, this effect in terms of traceable effects of varying components becomes more prominent. With an increasing number of components comes a corresponding increase in potential points of failure and inflated complexity [63].There is a general view in deep learning that a simpler solution is often better than an overly complex one [54]. From a maintainability perspective, that is an argument for prioritizing simple components over maximizing performance. However, as highlighted by Ran et al. [54], the pretraining phase is usually not the dominant part of the effort when developing general backbones, but rather the surrounding system of adapting and integrating them. Therefore, system-wide effects might be more relevant in terms of maintainability.

There were a few variables that we could see a direct effect on the robustness of the training pipeline. First was the masked block sampling strategy. Sampling blocks from the full grid caused the effective masking ratio to vary unpredictably across

scenes, making the difficulty of the pretext task inconsistent and contributing to training instability. Restricting sampling to occupied pillars resolved this, producing noticeably more stable training dynamics. Secondly, we made the initial connection that the EMA update rate can be seen as a trade-off between robust training, with a slow update rate, and efficient and scalable training, as a higher update rate enables faster convergence. Similarly, we hypothesized that masking ratio would also affect the robustness of the features. However, both were varied heuristically throughout cycles 2 and 3 but did not produce a clear isolated signal. The EMA rate was increased from 0.996 to 0.999 as part of a bundle of changes, and no clean attribution to training stability could be made. Similarly, variations in masking rate and block size across cycle 3 produced no significant differences in downstream performance, suggesting the pipeline is relatively insensitive to these parameters within the ranges explored.

Regarding the robustness of the model's performance, data augmentation has been shown to have a positive effect on the generalization of learned features [18]. During the development it emerged as the most impactful contributor to representation quality, by discouraging the model from relying on a fixed orientation. Instead it is pushed toward representations that generalize across varying input conditions, hence improving the representation quality and thereby the robustness of the perception component.

Model capacity showed the clearest empirical connection to scalability. Scaling the encoder embedding dimension from 256 to 512 improved downstream mAP from approximately 0.39 to 0.48, but increased training time per epoch by 48%. This illustrates the central scalability trade-off, larger models produce better representations but at a direct computational cost, and the self-attention mechanism's quadratic complexity with respect to pillar occupancy means this cost grows non-linearly as scene density or model size increases.

The predictor architecture also proved to be a meaningful scalability lever. Switching from self-attention to cross-attention in the predictor reduced training time by 24% and peak memory consumption from 4.12 GB to 3.16 GB, with marginal difference in downstream performance. This demonstrates that the predictor design has a significant impact on computational efficiency independent of representation quality. However, since no significant performance benefit was observed, the self-attention predictor was retained in the final configuration for consistency, leaving cross-attention as an opportunity for future optimization.

7.1.3 System Effect of pretraining on Software Quality

Together, the results of this study support the view that SSL pretraining as an architectural strategy is viable for improving the maintainability and scalability of LiDAR-based perception pipelines, with the note that achieving stable and effective pretraining requires a non-trivial engineering effort.

Once a stable pretraining pipeline is in place, the architectural benefits identified

in Section 2.2 begin to materialize. Firstly, an effect on maintainability is that the backbone does not need to be updated in response to changes in annotation requirements. It only needs to be retrained when the distribution of the raw sensor data shifts substantially, such as when the system is deployed in a new geographic region or with different sensor hardware. This directly addresses the annotation engineering bottleneck described in Section 2.2, where the coupling between model behavior and human-produced labels limits the system's ability to evolve.

In principle, raw LiDAR data could be collected continuously from deployed vehicles without any annotation effort, which means the pretraining data pipeline has the potential to be automated in ways that a supervised annotation pipeline cannot. The scalability of the dataset is increased as there is no annotation step required before the data can be used for training.

The architectural implications of task adaptation depend on whether the frozen or fine-tuning protocol is used, and this distinction has consequences for how the SoC principle is upheld across the pipeline. Under the frozen protocol, the backbone is never exposed to task-specific labels. The architectural boundary between the general-purpose encoder and the task head is preserved, creating an increased level of modularity in the system [66]. The backbone can under these conditions be shared across any number of downstream tasks without modification, and adding a new perception capability requires only training a new lightweight head. Changes to annotation requirements propagate no further than the task head, leaving the backbone and all other task heads untouched. This is the cleanest approach of the SSL pretraining paradigm as a software engineering strategy.

Fine-tuning relaxes this boundary. When the encoder is allowed to be updated, the backbone adapts to the specific task and its labels, reintroducing a degree of annotation dependency into the shared representation layer. The label budget results demonstrate that this dependency is reduced compared to fully supervised training (fine-tuning on 11% of the labeled data recovers approximately 92% of full-supervision performance) but it is not eliminated. In a multi-task setting, fine-tuning the backbone separately for each task would produce task-specific encoders that can no longer be strictly shared, reducing the architectural benefit of pretraining. However, the benefits of increased robustness remain, as the knowledge is not as strictly bound to the label taxonomy.

In practice, the choice between the frozen and fine-tuning protocols involves a trade-off between performance and architectural cleanliness. The frozen protocol preserves the full modularity of the pipeline, but may not achieve the desired performance. The fine-tuning protocol recovers that performance but at the cost of tightening the coupling between the backbone and the specific task. For production systems where performance is the primary concern, fine-tuning is likely preferable. For systems that must adapt frequently to new tasks or domains with limited labeled data, the frozen protocol better preserves the maintainability properties that motivate the SSL approach in the first place. However, in both cases, the pretraining pipeline is a single shared system to be maintained instead of the multiple diverse architectures

of fully supervised systems.

Whether the backbone developed in this study is sufficiently general to support this kind of multi-task reuse remains an open empirical question, since evaluation was limited to object detection. However, the fact that the pretraining objective was entirely task-agnostic supports the hypothesis that the representations have broader applicability, and this is a target for future investigation.

7.2 RQ2: Training Efficiency

The design variation that most positively impacted the performance, also implied the most significant impact on training efficiency and memory, which was increasing the encoder capacity. This was expected as a larger model naturally takes longer to process the data. This became the primary computational bottleneck in the pipeline, while the remaining operations did not show any significant changes for single training runs.

Part of this effect is likely connected to the self-attention mechanism, due to its quadratic complexity when attending over all occupied pillars, scaling with pillar occupancy density as well as embedding dimension. As discussed in 7.1, an alternative attention formulation was explored that reduced training time and memory consumption, by restricting mask tokens to attend only to visible pillars. However, as it offered no additional benefits beyond compute savings the initial method was kept. It may be worth reconsidering cross-attention in this setup in future work, as compute efficiency is an important consideration when deploying models in real-world applications like AD.

The most notable difference in overall training efficiency lies in the choice of learning paradigm. While the SSL pretraining phase is computationally intensive (71.7h, 25 epochs, 8.0GB), this cost is a one-time investment in learning general representations from unlabeled data. The downstream frozen protocol is then a comparatively lightweight process (6h, 18 epochs, 1.0GB), requiring less than a fifth of the compute of the supervised baseline (35.1h, 19 epochs, 5.0GB). This difference is significant as the pretrained backbone only needs to be trained once, meaning the upfront cost can be amortized over time, whereas supervised training must be repeated in full for each new task.

The definition of scalability adopted in this study emphasizes the systems ability to handle increasing workloads without degrading efficiency. The I-JEPA architecture, on which this work is based, has been shown to benefit from larger data sets and model sizes [4], demonstrating that performance scales favorably with data volume rather than degrading under it, which is in line with general knowledge about neural networks [35]. This leads us to believe that similar gains could be achievable in this setting.

However, this comes with a direct cost in the development process, as increased model capacity demands increased computational resources. We observed a limita-

tion where experimenting on large-scale models proved inefficient during iterative development, but down-scaling models might hide the true potential of other pipeline design choices when given the appropriate capacity.

7.3 RQ3: Limited Label Budgets

Having established how design decisions shape representation quality and at what computational cost, the central question of whether SSL pretraining proved efficient under annotation constraints remains open.

To evaluate the actual benefits from the SSL pipeline, the final demonstration was conducted to evaluate the two models performance with different label budgets. The label budget experiments showed that just by training a task head on a frozen pretrained model, it outperforms the supervised model in the low label budget regime, up to 250 annotated samples. After that, at 500 samples the baseline surpasses in performance, up to the biggest difference at the full training set of around 90,000 samples.

This demonstrates that the representations learned during pretraining are sufficiently rich that a lightweight nonlinear probe can achieve meaningful detection performance at very low label budgets, without any task-specific encoder tuning. As the label budget grows beyond 250 samples, the supervised model accumulates enough learning signal to begin learning strong representations independently, gradually closing the gap. This crossover was expected to happen at some point, as SSL's advantage generally is strongest where labeled data is limited. Although the performance advantage of the SSL is most pronounced at low label budgets, this does not necessarily limit the practical relevance of SSL. While real-world datasets are often large in total scale, they frequently contain underrepresented classes or rare scenarios where labeled examples are scarce. In such cases, the ability to perform well with few labels remains a meaningful property, as extensive annotation is rarely feasible for every class or edge case encountered in autonomous driving.

The fine-tuning results support this. When we applied fine-tuning to the self-supervised model instead of following the frozen protocol, it managed to outperform the supervised model on 1000 samples by 10.4%, all the way up to near identical performance to the baseline at full label budget, approximately a mAP of 0.83. At 10000 samples, the fine-tuned model manages to reach a mAP of 0.767, achieving approximately 92% of the fully supervised models performance using only 11% of the labeled data. This highlights the label efficiency of SSL pretraining and its potential to reduce dependency on human annotation, by effectively squeezing more utility out of each labeled sample.

The strong performance on limited labeled data has another implication. In these experiments we used vehicles as the only object class, but in real-world settings there will be many more, often rare object classes that the model should learn to identify. If the model can learn what a vehicle is from just a few samples, we hypothesize that

it could potentially learn rare classes where annotations are scarce more efficiently than a fully supervised model, which would mean that adding a new class requires significantly fewer labels than a fully supervised model to reach a useful performance threshold.

Whether this holds for rare classes beyond vehicles remains to be evaluated empirically, but the fine-tuning results suggest that a SSL-pretrained backbone can at least partially decouple annotation effort from model capability.

7.4 Threats to Validity

7.4.1 Conclusion Validity

Given the non-deterministic nature of ML, repeated runs of the same experiment may produce slightly different results. To reduce the influence of this natural variation, the label-budget experiments could have been repeated multiple times and the results averaged across runs. However, the label-budget experiments were only run once, meaning that we cannot quantify how great the variance is for individual experiments nor can we draw absolute conclusions from the comparison with the baseline.

The experiments were performed under identical conditions with fixed seeds, fixed hyper-parameters, and fixed dataset splits, to minimize any stochastic effects, but all variation is practically impossible to enforce[84].

7.4.2 Internal Validity

While some design variables were tested in isolation during the development process, the large design space made exhaustive ablation infeasible, meaning we cannot confirm the isolated impact of every design decision on representation quality. Once the model reached a point of stability, more targeted experiments were conducted, attempting to vary one variable at a time to assess individual impact on the representations. However, since modifications were applied on top of the previous configuration rather than against a fixed baseline, interaction effects between design choices cannot be fully ruled out.

7.4.3 Construct Validity

Since there is no direct measure of representation quality, it was evaluated on the performance of the downstream task. However, evaluating on vehicle detection only may not fully capture the quality of the representations, and could have introduced an unknown bias towards training strategies that happen to be good for detecting vehicle-sized objects rather than learning general representations. We partially addressed this by also inspecting representations qualitatively via PCA visualizations and cosine similarity maps, which provide a complementary, task-independent signal of structure in the learned embedding space. However, these visualizations are

inherently subjective, there is no formal criterion for what a "good" PCA plot looks like in this setting, meaning different observers might interpret the same results differently. Getting similar performance gains for different object categories or tasks would help rule this out, but it was outside the scope of this thesis.

Furthermore, the study did not directly measure or experimentally establish the implications of design decisions on scalability, maintainability, or robustness. Since the model was never integrated into an operational perception pipeline, we could not empirically verify whether the theoretical benefits materialize in practice. For instance, maintainability claims are grounded on the assumption that backbone reuse reduces update costs, but this was not observed across actual system updates. Similarly, scalability was not tested beyond the model sizes feasible within the computational and time constraints of this project. Conclusions in these areas are based on reasoning and literature rather than empirical evidence and should therefore be interpreted as architectural expectations rather than empirically grounded findings.

7.4.4 External Validity

Given that our model has learned representations that can solve the object detection task achieving reasonable performance without having been trained on it, which is promising for generalizability and suggests it could perform equally well on other tasks it hasn't been trained on. However, within the scope of this thesis the representations were only evaluated on object detection, without evaluation on additional downstream tasks such as sematic segmentation or scene understanding. We cannot empirically confirm the representations transfer equally well across tasks. The use of a single dataset also means that findings may not generalize to other sensor configurations, geographic regions, or driving conditions not represented in ZOD. Evaluation on additional datasets and downstream tasks is the natural next step to establish broader generalizability.

7.4.5 Disclosure of Generative AI Use

Claude AI was used during the writing of this thesis as a means for finding spelling and structural grammar mistakes. It was not used as a method for generating results.

8

Conclusion

This thesis investigated whether self-supervised pretraining using a JEPA-based architecture could learn informative representations from LiDAR point clouds of autonomous driving scenes. Furthermore, it explored whether such representations could reduce the annotation dependency and maintenance complexity of an AD system.

The results demonstrate that the pretraining strategy did produce transferable representations, most clearly demonstrated by the limited label budget experiments. The SSL model outperformed the supervised baseline in the low label-budget regime, showcasing that leveraging a pretraining strategy for LiDAR-based perception is a label-efficient alternative to supervised approaches.

Under the full label budget, the finetuned model reached comparable performance to the supervised baseline, while the frozen model did not reach the same standard. Since other works using similar approaches, such as I-JEPA [4], Occupancy-MAE [44], AD-L-JEPA [83] demonstrate that improved performance from pretraining is possible, we believe that further stabilizing the SSL-training pipeline could result in more informative and useful representations.

From a software engineering perspective, the approach shows potential for reducing annotation dependency and improving maintainability through backbone reuse, but these benefits come with trade-offs in engineering complexity and computational cost that should be weighed carefully in practice, such as long-term label-efficiency and architectural benefits versus high upfront engineering effort.

8.1 Future Work

To build on these insights, and to address the limitations of this study, future work could focus on:

- Further investigation into training strategies to improve the quality of the representations: Since the point cloud is essentially positional information only, this setup keeps favoring geometric rather than semantic structures. Further investigation into various masking strategies, as well as the role of positional encoding and attention-mechanisms would be relevant in order to find a better

difficulty level for the pretext task.

- Broader evaluation protocol to measure generalizability. Extend the object detection task to include multiple object classes, to investigate robustness across varying object types. Further investigation into robustness in different kinds of edge cases, distances and dataset splits would further examine the potential gains or limitations of the pretraining strategy. Moreover, extending to multiple downstream tasks would broaden the investigation of generalization of the achieved representations.

- Explore multi-task fine-tuning: Given that the backbone demonstrates strong performance under the fine-tuning protocol, it could be worth investigating multi-task fine-tuning to leverage unlabeled data even under the fine-tuning protocol, to minimize the reintroduction of annotation dependence.

- Empirical study of maintainability in practice: The improved maintainability effects discussed in this thesis, such as backbone reusability and simplified system maintenance, are explored from the literature rather than observed empirically. Studying the artifact in active use over time, for example by measuring how engineering effort scales as the system is extended to new tasks or updated in response to data distribution shifts, would clarify whether these theoretical benefits materialize in practice.

Bibliography

[1] Mohamed Abdel-Aty and Shengxuan Ding. A matched case-control analysis of autonomous vs human-driven vehicle accidents. *Nature Communications*, 15, 06 2024.

[2] Simegnew Yihunie Alaba and John E. Ball. A survey on deep-learning-based lidar 3d object detection for autonomous driving. *Sensors*, 22(24):9577, 2022.

[3] Mina Alibeigi, William Ljungbergh, Adam Tonderski, Georg Hess, Adam Lilja, Carl Lindström, Daria Motorniuk, Junsheng Fu, Jenny Widahl, and Christoffer Petersson. Zenseact open dataset: A large-scale and diverse multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[4] Mahmoud Assran, Quentin Duval, Ishan Misra, Piotr Bojanowski, Pascal Vincent, Michael Rabbat, Yann LeCun, and Nicolas Ballas. Self-supervised learning from images with a joint-embedding predictive architecture. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15619–15629, 2023.

[5] Abhishek Balasubramaniam and Sudeep Pasricha. Object detection in autonomous vehicles: Status and open challenges. *arXiv preprint*, 2022.

[6] Firas Bayram and Bestoun S. Ahmed. Towards trustworthy machine learning in production: An overview of the robustness in mlops approach. *ACM Comput. Surv.*, 57(5), January 2025.

[7] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Jürgen Gall, and Cyrill Stachniss. Towards 3d lidar-based semantic scene understanding of 3d point cloud sequences: The semantickitti dataset. *The International Journal of Robotics Research*, 40(8-9):959–967, 2021.

[8] Rishi Bommasani, Drew Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney Arx, Michael Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Davis, Dora Demszky, and Percy Liang. On the opportunities and risks of foundation models, 08

2021.

[9] Alexandre Boulch, Corentin Sautier, Björn Michele, Gilles Puy, and Renaud Marlet. Also: Automotive lidar self-supervision by occupancy estimation. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13455–13465, 2023.

[10] Houssem Ben Braiek and Foutse Khomh. Chapter 3 - machine learning robustness: a primer. In Marco Lorenzi and Maria A. Zuluaga, editors, *Trustworthy AI in Medical Imaging*, The MICCAI Society book Series, pages 37–71. Academic Press, 2025.

[11] Razvan-Alexandru Bratulescu, Robert-Ionut Vatasoiu, George Sucic, Sorina-Andreea Mitroi, Marius-Constantin Vochin, and Mari-Anais Sachian. Object detection in autonomous vehicles. In *2022 25th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 375–380, 2022.

[12] Jan vom Brocke, Alan Hevner, and Alexander Maedche. *Introduction to Design Science Research*, pages 1–13. Springer International Publishing, 09 2020.

[13] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[14] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119, pages 1597–1607. PMLR, 2020.

[15] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. Big self-supervised models are strong semi-supervised learners, 2020.

[16] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15750–15758, June 2021.

[17] Yukang Chen, Jianhui Liu, Xiangyu Zhang, Xiaojuan Qi, and Jiaya Jia. VoxelNeXt: Fully sparse voxelnet for 3d object detection and tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21674–21683, June 2023.

[18] Weijian Deng, Stephen Gould, and Liang Zheng. On the strong correlation between model invariance and generalization. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[20] George Eskandar, Robert A. Marsden, Pavithran Pandiyan, Mario Döbler, Karim Guirguis, and Bin Yang. An unsupervised domain adaptive approach for multimodal 2d object detection in adverse weather conditions, 2022.

[21] Timo Freiesleben and Thomas Grote. Beyond generalization: a theory of robustness in machine learning. *Synthese*, 202(4):109, 2023.

[22] Quentin Garrido, Mahmoud Assran, Nicolas Ballas, Adrien Bardes, Laurent Najman, and Yann LeCun. Learning and leveraging world models in visual representation learning, 2024.

[23] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent a new approach to self-supervised learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[24] Rahul Gupta, Shashank Singh, Vandana Yadav, Namrata Dhanda, and Pakhi Sharma. A comparative study of cnn and transformer models for image recognition in autonomous driving. In *2025 2nd International Conference on Computational Intelligence, Communication Technology and Networking (CICTN)*, pages 237–244, 2025.

[25] Khan Mohammad Habibullah, Hans-Martin Heyn, Gregory Gay, Jennifer Horkoff, Eric Knauss, Markus Borg, Alessia Knauss, Håkan Sivencrona, and Polly Jing Li. Requirements and software engineering for automotive perception systems: an interview study. *Requirements Engineering*, 29(1):25–48, 2024.

[26] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15979–15988, 2022.

[27] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the*

IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 9729–9738, June 2020.

[28] Alex Hernández-García and Peter König. Data augmentation instead of explicit regularization, 2020.

[29] Georg Hess, Johan Jaxing, Elias Svensson, David Hagerman, Christoffer Petersson, and Lennart Svensson. Masked autoencoder for self-supervised pre-training on lidar point clouds. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW)*, page 350–359. IEEE, January 2023.

[30] Hans-Martin Heyn, Khan Mohammad Habibullah, Eric Knauss, Jennifer Horkoff, Markus Borg, Alessia Knauss, and Polly Jing Li. Automotive perception software development: An empirical investigation into data, annotation, and ecosystem challenges, 2023.

[31] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, Lewei Lu, Xiaosong Jia, Qiang Liu, Jifeng Dai, Yu Qiao, and Hongyang Li. Planning-oriented autonomous driving. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17853–17862, 2023.

[32] Hussein, Bootan M. and Shareef, Shareef M. An empirical study on the correlation between early stopping patience and epochs in deep learning. *ITM Web Conf.*, 64:01003, 2024.

[33] ISO Iso. Iec25010: 2011 systems and software engineering–systems and software quality requirements and evaluation (square)–system and software quality models. *International Organization for Standardization*, 34(2910):208, 2011.

[34] Li Jing, Pascal Vincent, Yann LeCun, and Yuandong Tian. Understanding dimensional collapse in contrastive self-supervised learning, 2022.

[35] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[36] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[37] Moez Krichen. Convolutional neural networks: A survey. *Computers*, 12(8), 2023.

[38] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition*

(CVPR), pages 12689–12697, 2019.

[39] Alexander C. Li, Alexei A. Efros, and Deepak Pathak. Understanding collapse in non-contrastive siamese representation learning. In Shai Avidan, Gabriel Brostow, Moustapha Cissé, Giovanni Maria Farinella, and Tal Hassner, editors, *Computer Vision – ECCV 2022*, pages 490–505, Cham, 2022. Springer Nature Switzerland.

[40] Jinlong Li, Runsheng Xu, Xinyu Liu, Jin Ma, Baolu Li, Qin Zou, Jiaqi Ma, and Hongkai Yu. Domain adaptation based object detection for autonomous driving in foggy and rainy weather, 2024.

[41] Gengchen Mai, Weiming Huang, Jin Sun, Suhang Song, Deepak Mishra, Ninghao Liu, Song Gao, Tianming Liu, Gao Cong, Yingjie Hu, Chris Cundy, Ziyuan Li, Rui Zhu, and Ni Lao. On the opportunities and challenges of foundation models for geoai (vision paper). *ACM Trans. Spatial Algorithms Syst.*, 10(2), July 2024.

[42] Peter Mattson, Christine Cheng, Cody Coleman, Greg Diamos, Paulius Micikevicius, David Patterson, Hanlin Tang, Gu-Yeon Wei, Peter Bailis, Victor Bittorf, David Brooks, Dehao Chen, Debojyoti Dutta, Udit Gupta, Kim Hazelwood, Andrew Hock, Xinyuan Huang, Atsushi Ike, Bill Jia, Daniel Kang, David Kanter, Naveen Kumar, Jeffery Liao, Guokai Ma, Deepak Narayanan, Tayo Oguntebi, Gennady Pekhimenko, Lillian Pentecost, Vijay Janapa Reddi, Taylor Robie, Tom St. John, Tsuguchika Tabaru, Carole-Jean Wu, Lingjie Xu, Masafumi Yamazaki, Cliff Young, and Matei Zaharia. Mlperf training benchmark, 2020.

[43] Gaurav Menghani. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *ACM Comput. Surv.*, 55(12), March 2023.

[44] Chen Min, Liang Xiao, Dawei Zhao, Yiming Nie, and Bin Dai. Occupancy-mae: Self-supervised pre-training large-scale lidar point clouds with masked occupancy autoencoders. *IEEE Transactions on Intelligent Vehicles*, 9(7):5150–5162, 2024.

[45] Curtis G. Northcutt, Lu Jiang, and Isaac L. Chuang. Confident learning: Estimating uncertainty in dataset labels, 2022.

[46] Kento Nozawa and Issei Sato. Empirical evaluation and theoretical analysis for representation learning: A survey, 2022.

[47] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jégou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual

features without supervision, 2024.

[48] Yatian Pang, Wenxiao Wang, Francis E. H. Tay, Wei Liu, Yonghong Tian, and Li Yuan. Masked autoencoders for point cloud self-supervised learning. In *Computer Vision – ECCV 2022*, volume 13662 of *Lecture Notes in Computer Science*, pages 604–621. Springer, Cham, 2022.

[49] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77, 2007.

[50] Yi Peng, Hina Saeeda, Hans-Martin Heyn, and Jennifer Horkoff. Data annotation: A requirements engineering for machine learning systems perspective. In *2025 IEEE 33rd International Requirements Engineering Conference (RE)*, pages 572–575, 2025.

[51] Yi Peng, Hina Saeeda, Hans-Martin Heyn, Jennifer Horkoff, Eric Knauss, and Fredrick Warg. A data annotation requirements representation and specification (dars). *arXiv preprint arXiv:2512.13444*, 2025.

[52] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 652–660, July 2017.

[53] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[54] Dezhi Ran, Mengzhou Wu, Wei Yang, and Tao Xie. Foundation model engineering: Engineering foundation models just as engineering software. *ACM Trans. Softw. Eng. Methodol.*, 34(5), May 2025.

[55] Gilberto Recupito, Giammaria Giordano, Filomena Ferrucci, Dario Di Nucci, and Fabio Palomba. When code smells meet ml: On the lifecycle of ml-specific code smells in ml-enabled systems. *Empir Software Eng*, 30(139), 2025.

[56] Ricardo Roriz, Jorge Cabral, and Tiago Gomes. Automotive lidar technology: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6282–6297, 2022.

[57] Hina Saeeda, Tommy Johansson, Mazen Mohamad, and Eric Knauss. Data annotation quality problems in ai-enabled perception system development, 2025.

[58] Hina Saeeda, Mazen Mohamad, Eric Knauss, Jennifer Horkoff, and Ali Nouri. Re for ai in practice: Managing data annotation requirements for ai autonomous driving systems, 2025.

[59] Emanuele Sansone, Tim Lebailly, and Tinne Tuytelaars. Collapse-proof non-contrastive self-supervised learning, 2025.

[60] Rajendramayavan Sathyam and Yueqi Li. Foundation models for autonomous driving perception: A survey through core capabilities. *IEEE Open Journal of Vehicular Technology*, 6:2554–2582, 2025.

[61] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[62] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. PointRCNN: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[63] Karthik Shivashankar, Ghadi Al Hajj, and Antonio Martini. Maintainability and scalability in machine learning: Challenges and solutions. *ACM Comput. Surv.*, 57(12), July 2025.

[64] Karthik Shivashankar and Antonio Martini. Maintainability challenges in ml: A systematic literature review. In *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 60–67. IEEE, 2022.

[65] Miroslaw Staron, Silvia Abrahão, Grace Lewis, Henry Muccini, and Chetan Honnenahalli. Bringing software engineering discipline to the development of ai-enabled systems. *IEEE Software*, 41(5):79–82, 2024.

[66] Haozhe Sun and Isabelle Guyon. Modularity in deep learning: A survey. In Kohei Arai, editor, *Intelligent Computing*, pages 561–595, Cham, 2023. Springer Nature Switzerland.

[67] Pei Sun, Mingxing Tan, Weiyue Wang, Chenxi Liu, Fei Xia, Zhaoqi Leng, and Dragomir Anguelov. SWFormer: Sparse window transformer for 3d object detection in point clouds. In *Computer Vision – ECCV 2022*, volume 13670 of *Lecture Notes in Computer Science*, pages 426–442. Springer, Cham, 2022.

[68] Mingkun Tan, Daniel Langenkämper, Michael Kloster, and Tim W. Nattkemper. Scaling down annotation needs: The capacity of self-supervised learning on diatom classification. *iScience*, 28(4):112236, 2025.

[69] Juan Terven, Diana-Margarita Cordova-Esparza, Julio-Alejandro Romero-González, Alfonso Ramírez-Pedraza, and E. A. Chávez-Urbiola. A comprehensive survey of loss functions and metrics in deep learning. *Artificial Intelligence Review*, 58(7), April 2025.

[70] Hugues Van Assel, Mark Ibrahim, Tommaso Biancalani, Aviv Regev, and Randall Balestriero. Joint-embedding vs reconstruction: Provable benefits of latent space prediction for self-supervised learning. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen, editors, *Advances in Neural Information Processing Systems*, volume 38, pages 21897–21937. Curran Associates, Inc., 2025.

[71] Gustavo Velasco-Hernandez, John Barry, Joseph Walsh, et al. Autonomous driving architectures, perception and data fusion: A review. In *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 315–321. IEEE, 2020.

[72] John Venable, Jan Pries-Heje, and Richard Baskerville. Feds: a framework for evaluation in design science research. *European Journal of Information Systems*, 25(1):77–89, 2016.

[73] Hironori Washizaki, Foutse Khomh, Yann-Gaël Guéhéneuc, Hironori Takeuchi, Naotake Natori, Takuo Doi, and Satoshi Okuda. Software-engineering design patterns for machine learning applications. *Computer*, 55:30–39, 2022.

[74] Jaeyeon Won, Willow Ahrens, Saman Amarasinghe, and Joel S. Emer. Insum: Sparse gpu kernels simplified and optimized with indirect einsums. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '26, page 993–1006, New York, NY, USA, 2026. Association for Computing Machinery.

[75] Aoran Xiao, Xiaoqin Zhang, Ling Shao, and Shijian Lu. A survey of label-efficient deep learning for 3d point clouds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):9139–9160, 2024.

[76] Dongqiangzi Ye, Zixiang Zhou, Weijia Chen, Yufei Xie, Yu Wang, Panqu Wang, and Hassan Foroosh. Lidarmultinet: towards a unified multi-task network for lidar perception. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023.

[77] Huilin Yin, Ziming Zhao, Jun Yan, and Daniel Watzenig. Certified robustness in automated driving perception: A review. *Automotive Innovation*, 8(4):817–837, 2025.

[78] Sangdoo Yun, Seong Joon Oh, Byeongho Heo, Dongyoon Han, Junsuk Choe, and Sanghyuk Chun. Re-labeling imagenet: from single to multi-labels, from global to localized labels, 2021.

[79] Changyu Zeng, Wei Wang, Anh Nguyen, Jimin Xiao, and Yutao Yue. Self-supervised learning for point cloud data: A survey. *Expert Syst. Appl.*, 237(PB), March 2024.

[80] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning, 2019.

[81] Zhenguo Zhang, Yuanzhouhan Cao, Hui Zhang, Naiyue Chen, Chao Ren, and Yidong Li. Vision bev-mae: Masked bird's eye view autoencoders for camera-based 3d object detection. In *2024 China Automation Congress (CAC)*, pages 4409–4414, 2024.

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

[83] Haoran Zhu, Zhenyuan Dong, Kristi Topollai, Beiyao Sha, and Anna Choromanska. Self-supervised representation learning with joint embedding predictive architecture for automotive lidar object detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40:13925–13933, 03 2026.

[84] Donglin Zhuang, Xingyao Zhang, Shuaiwen Song, and Sara Hooker. Randomness in neural network training: Characterizing the impact of tooling. In D. Marculescu, Y. Chi, and C. Wu, editors, *Proceedings of Machine Learning and Systems*, volume 4, pages 316–336, 2022.