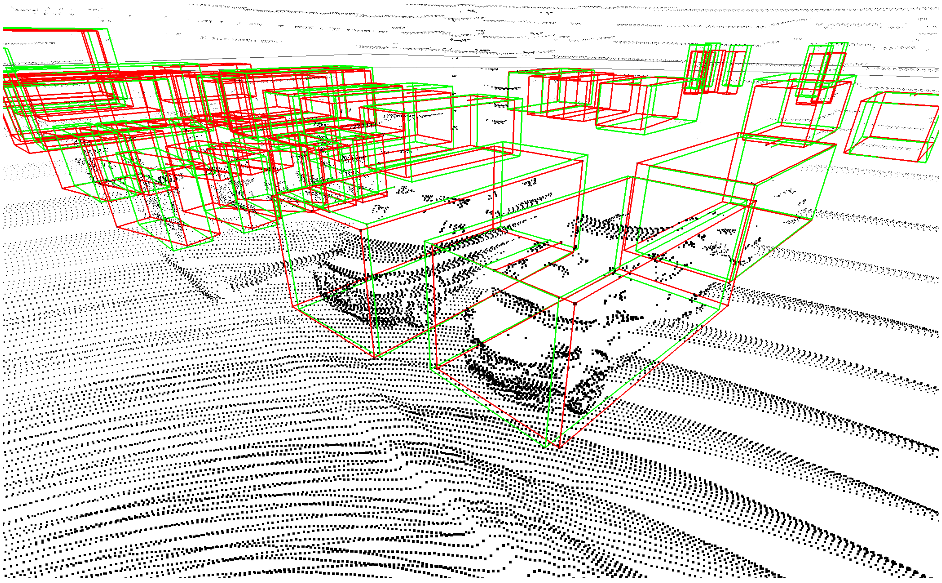
CHALMERS
UNIVERSITY OF TECHNOLOGY



Spatial Distribution Data Augmentations for Long Range LiDAR Object Detection

Master's thesis in Systems, Control and Mechatronics

KRISTIAN CEDER & ARVID ENLIDEN

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Spatial Distribution Data Augmentations for Long Range LiDAR Object Detection

Kristian Ceder & Arvid Enliden

Department of Electrical Engineering
Division of Signal Processing and Biomedical engineering
Chalmers University of Technology
Gothenburg, Sweden 2023

Spatial Distribution Data Augmentations for Long Range LiDAR Object Detection
Kristian Ceder & Arvid Enliden

Supervisor: Jakob Holmgren, Einride
Supervisor: Hugues de Valon, Einride
Supervisor: Prof. Lennart Svensson, Department of Electrical Engineering
Examiner: Prof. Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2023
Department of Electrical Engineering
Division of Signal Processing and Biomedical engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Point cloud sample from the nuScenes dataset with ground truth bounding boxes in green and predicted bounding boxes in red.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Spatial Distribution Data Augmentation for Long Range LiDAR Object Detection
Kristian Ceder & Arvid Enliden
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Since the KITTI dataset was introduced in 2013, research in LiDAR-based object detection (OD) has increased significantly in popularity. The spike in research has also led to improved performance, but the predictions of these models are still not robust or accurate enough to safely be used for autonomous driving. Although the overall performance of the models is insufficient, long-range predictions is an area where the need for better accuracy is particularly apparent. The emergence of large datasets has improved the overall performance of such models, but most datasets still require data augmentations to help models converge and generalize well to unseen data. Many methods used for data augmentation in LiDAR OD have been accepted as standards in the field since they have been shown to improve overall performance. Techniques like *Ground Truth Database Sampling* (GTDS) and *Fade* are used without thoroughly analyzing the class-specific effects and results they impose on the models.

This thesis investigates how spatial distribution data augmentations can improve long-range accuracy of LiDAR OD. This is done by downsampling ground truth objects from the original GTDS and moving them further away from the sensor. The goal is to increase diversity in the long-range data while keeping a realistic spatial distribution of points for objects sampled at that range. Furthermore, a class-specific analysis of the widely accepted GTDS and Fade augmentations is conducted to further explore the effects these data augmentations have on the popular nuScenes dataset. All experiments are conducted on the two common benchmarking models PointPillars and CenterPoint. The work shows that GTDS can negatively impact the detection accuracy of less frequent classes in nuScenes, even if the overall accuracy increases. Moreover, we demonstrate the importance of using Fade in conjunction with GTDS and how it can mitigate class-specific accuracy losses introduced by GTDS. Lastly, this thesis analyzes why the long-range sampler ultimately fails at increasing long-range accuracy.

Keywords: Object detection, LiDAR, Data augmentation, Deep learning, nuScenes, Long range.

Acknowledgements

Firstly we would like to thank our examiner and academic supervisor Prof. Lennart Svensson for dedicating time, support and expertise all the way from the first thesis proposal to the final product.

Secondly we would like to thank and acknowledge our supervisors at Einride, Hugues de Valon and Jakob Holmgren for their feedback and support during the course of this project.

Lastly, a special thanks goes out to Einride's employees for their help, curiosity and encouragement in this thesis.

Kristian Ceder & Arvid Enliden, Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AAE	Average Attribute Error
AD	Autonomous Driving
AOE	Average Orientation Error
(m)AP	(mean) Average Precision
ASE	Average Scale Error
ATE	Average Translation Error
AVE	Average Velocity Error
BCE	Binary Cross Entropy
BEV	Birds Eye View
FN	False Negative
FP	False Positive
FPS	Farthest Point Sampling
GPS	Global Positioning System
GPU	Graphics Processing Unit
GT	Ground Truth
GTDS	Ground Truth Database Sampling
HRFiS	Hard Range Filtering Sampling
IMU	Internal Measurement Unit
IoU	Intersection over Union
LiDAR	Light Detection And Ranging
MLP	Multilayer Perceptron
NDS	NuScenes Detection Score
NN	Neural Network
OD	Object Detection
RFD	Random Factor Downsampling
RRD	Random Range Downsampling
SOTA	State Of The Art
TP	True Positive
WOD	Waymo Open Dataset

Contents

List of Figures

List of Tables

1 Introduction

In the pursuit of increased traffic safety and fully autonomous driving, the need for accurate detection and localization of surrounding objects is becoming increasingly important. Vehicles becoming fully autonomous could include several benefits, e.g., saving lives in traffic, making transport more efficient and freeing up time during travel for humans. This often requires advanced object detection methods utilizing multiple vision sensors. The most commonly used sensors for this purpose are cameras, LiDARs and radars.

LiDAR sensors have become a central sensor in 3D Object Detection (OD). Thanks to their precise distance measurements and low latency, LiDAR sensors excel at providing spatial information about objects that are far away from the sensor. Advances in deep learning have enabled analysis of the detailed geometric data, allowing detection algorithms to distinguish objects like pedestrians, bicycles and cars based on their size and shape. Additionally, LiDARs are comparatively robust to low-light environments and varying weather conditions, making them a good choice when robustness is a deciding factor.

Although State of the Art LiDAR OD models continue to improve, current implementations have neither the robustness nor accuracy to be able to guarantee safety in autonomous driving. A common problem with many of the algorithms using LiDAR point clouds is their inability to reliably detect and classify objects that are far away from the sensor. This is mainly caused by LiDAR point clouds becoming increasingly sparse further away from the sensor, but also a side effect of the lack of annotated LiDAR data available at longer ranges. A common solution used in [22] is to rely on a multi-modal fusion strategy e.g., combining a camera together with the LiDAR to increase accuracy at a distance.

However, an important reason not to solve this issue by early fusion with camera data is redundancy. The need for careful time synchronization and calibration between cameras and LiDARs makes early fusion models more sensitive to failure cases. Ideally, the two different approaches should both, independently of each other, be accurate enough to be used in the sensors' preferred environment. Combining the sensors too early in the research might inhibit the progress of the single modality models individually, ultimately affecting the overall safety and integrity of the system.

The amount of open-source annotated point cloud data available has increased in recent times, but there is still a need for more diverse datasets to help the models perform and generalize well to unseen data. In this thesis, we study techniques that augment the data available in open-source datasets to further diversify it. We also try to further enhance the techniques to provide more diverse data on longer ranges by moving and downsampling parts of the data.

1.1 Thesis aim

Data augmentations used in new models and their publications are used as common practice and their effects are often overlooked or not analyzed closely in their papers. The aim of this thesis is to investigate how some common augmentations affect performance, with special focus on Ground Truth Database Sampling (GTDS). Furthermore, it also aims to investigate if GTDS can be modified to increase detection accuracy on longer ranges by moving sampled objects from the standard GTDS further away from the sensor. This introduces additional challenges due to the vastly different point representations of objects depending on their location in relation to the sensor. Hence, we also aim to investigate how objects can be properly downsampled to conform to these point representations.

GTDS is commonly complemented with an augmentation strategy called *Fade* [31], which refers to training without GTDS during the later epochs of a model's training in order to better adapt to the real-world data distribution of the datasets used. Although it has been proven to result in significant performance gains, it has to the best of our knowledge never been examined systematically to determine its effect on different detection classes.

1.2 Related work

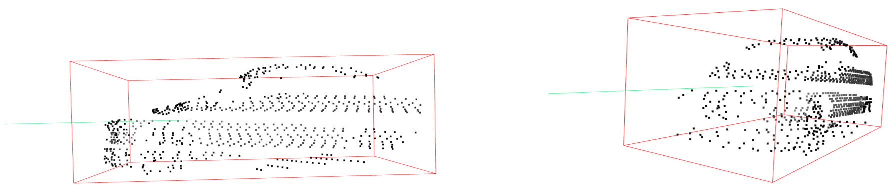
This section aims at introducing work that is similar to what will be presented in this thesis. It will focus on works that are closely related to GTDS and how they relate to ours.

A few attempts utilizing advanced techniques have been made in order to realistically model where ground truth objects should be placed and what the subset of points inside each object should be. The authors of [36] propose what they call *Shape Aware Data Augmentations* or *Frustum Augmentations*, which is a modification of GTDS where each bounding box is split into six different sections. Each section is then randomly augmented with one of the following techniques before being pasted into a scene: dropping all points in a section, swapping one of the sections with the same section from another object, or downsampling one of the sections using Farthest Point Sampling (FPS). According to the results presented in their ablation studies, this improves the AP of the Car class on the KITTI dataset by about 0.5 percentage points for all three difficulty levels (*easy*, *medium* and *hard*) [11].

Another work, [17], puts more focus on where objects are placed when sampled. They argue that placing objects randomly into scenes misleads the models to learn the wrong context information between objects and background. As a result of this, a method that extracts the valid space for where objects can be placed into a scene is developed and used when applying GTDS on the KITTI dataset. The results vary quite a lot between the different models the method is applied to but it generally improves the detection score, especially for the cyclist class.

A specific work that shares a lot of similarities with what we are proposing in this thesis is [16]. The authors of the paper aim to increase long-range detections on the KITTI dataset by moving them further away from the sensor by a fixed factor of twice the 2D Birds Eye View (BEV) distance. Downsampling the objects that are moved further away is done systematically by deleting whole scan lines of the LiDAR points inside a bounding box along with removing every other point in the scan lines that are left. This is easily manageable as long as the distance of the objects is exactly doubled, since this should result in dividing the number of scan lines hitting the object by 2 while also dropping every other point from each of the scan lines, staying true to the natural representation of the objects detected by a LiDAR sweep.

However, this is a bit trickier on the NuScenes dataset due to the common concatenation of LiDAR sweeps that is used to make the point cloud more dense. The concatenation adds noise to the scan lines between different sweeps, making it a lot harder to distinguish which points belong to which scan line. A clear example of this phenomenon is shown in Figure 1.1. This is especially true for objects that are originally detected close to the sensor, leaving little room between scan lines. Additionally, the concatenation process only takes into account the ego vehicle's movement, resulting in even more scan line noise for objects that are in motion.



(a) Side view. **(b)** Side-front view.

Figure 1.1: Two views of the same object to visualize the noise in scan lines for objects with 10 concatenated frames on the NuScenes dataset. The object is a Car located at a 13m distance from the ego-vehicle.

1.3 Contributions

The main contributions of this thesis are as follows:

- We investigate the effect of GTDS on two models; PointPillars and CenterPoint, both on the nuScenes dataset. We show that while GTDS may be beneficial in most cases, it can sometimes do more harm than good for specific classes. We also show that these results may vary depending on which model is used.
- We design range-specific tests for evaluating performance on the nuScenes dataset. The range intervals include short, medium and long range.
- We make an attempt at increasing the mAP of these models for long-range detections (on an interval ranging from 2/3 to 3/3 of the max range for each class) by modifying the GTDS-sampler using three different methods. This is done by moving objects further away from the sensor before downsampling them accordingly or by simply filtering them by range in order to stay consistent with the point distributions inside objects.
- We systematically examine the effects of Fade for CenterPoint and PointPillars on the nuScenes dataset, which has to the best of our knowledge not been done before. Our work show that Fade can mitigate the reduced accuracy that is introduced by using GTDS for uncommon classes.

1.4 Thesis outline

In Chapter 2 (Preliminaries), background and important theoretical concepts are explained in order to provide the reader with information that we find necessary to be able to understand the rest of the work conducted in this thesis. In Chapter 3 (Methods) we describe how the experiments are conducted and present the different versions of the modified GTDS to be evaluated. Chapter 4 (Experimental Evaluation) presents implementation details like tuning parameters of the methods explained in Chapter 3, the results achieved by carrying out these experiments and a detailed discussion about the outcome of these. Chapter 5 (Conclusion) starts with the main takeaways from this thesis and is continued with proposed subjects for further research. It is then finished with reflections on the possible ethical and societal impacts related to research in OD.

2
Preliminaries

This section aims to give the reader knowledge of important theoretical concepts related to LiDAR-based 3D OD and common data augmentations. It begins by introducing the basics of LiDAR sensors and common OD datasets which include LiDAR data. Furthermore, fundamental theory related to neural networks (NNs) will be presented before moving on to subjects like 3D OD, metrics and data augmentations, which are more specific to this thesis.

2.1 LiDAR

LiDAR is an abbreviation for *Light Detection And Ranging*. It is a sensor type that is commonly used when precise 3D information is required. A LiDAR sensor cast beams of light and measures the time it takes for that beam to reflect back to the sensor. The LiDAR also measures the intensity of the reflected light. This information is then used to create a sparse but highly accurate 3D environment of points. The most common type of LiDAR is the 360° rotating LiDAR. The resolution of rotating LiDARs is mainly affected by the number of beams (layers) used to scan the environment. Most LiDAR sensors use layer counts of 16, 32, or 64. Each layer is a laser with a fixed angle relative to the sensor's rotational axis. The layer dimension is usually called elevation and the rotating angle around the sensor's z-axis is called azimuth.

As the sensor rotates, each laser in the layers measures the distance to a point in space. The distance d of a point is calculated using the formula

$$d = \frac{c \cdot t}{2}, \tag{2.1}$$

where c is the speed of light and t is the time it takes for the ray of light to bounce on an object and back to the sensor. Given a traditional spherical coordinate system parameterized using elevation Θ, azimuth φ and distance d, a point in 3D can be defined. All points acquired from one rotation of the sensor are then combined into a point cloud represented with spherical coordinates. The point clouds are usually converted to a Cartesian coordinate system for further processing. The Cartesian coordinates of each point x, y and z are calculated using

$$\begin{aligned} x &= d \sin\Theta \cos\varphi, \\ y &= d \sin\Theta \sin\varphi, \\ z &= d \cos\Theta. \end{aligned} \tag{2.2}$$

2.2 Object detection in 2D and 3D data

2D OD is a well-researched subject that refers to detecting instances of objects, usually in an RGB image, while also classifying and localizing them. In images, this task has during the last ten years been dominated by deep learning models. Some of the most well-known models in later years for this task include YOLO [2], SSD [21] and DETR [4]. YOLO and SSD are one-stage detectors that mainly utilize convolutional neural networks, while DETR is based on a transformer architecture. In order to be utilized for autonomous driving purposes, the most important OD metrics are *accuracy* and *speed* (commonly measured as mAP and inference time [s] or frequency [Hz], respectively).

Although 2D OD can be argued to be a crucial part of autonomous driving, safely navigating complex traffic environments relies on accurately localizing objects in a 3D environment. While this is theoretically possible using only cameras and Monocular Depth Estimation like [27], high-end precision for 3D OD currently relies on other sensor modalities like RADAR and LiDAR. Although it has gained more traction in later years as sensors and computational hardware has been evolving and getting cheaper, 3D OD is a substantially less researched area than 2D OD. The main enabler of these methods was when the KITTI dataset was extended to include benchmarks for 3D OD in 2017.

LiDAR 3D OD is similar to 2D OD in the sense that accuracy and speed are of great importance, but the data that the NN's operate on is vastly different. LiDAR data is sparse by nature, which makes it hard to directly transfer 2D OD models to 3D OD tasks. Generally, 3D OD performance is measured using mAP which is usually calculated by Intersection over Union (IoU) matching [11, 30] or center-distance matching [3].

2.3 Artificial neural networks

The term deep neural networks is commonly used for NNs consisting of many layers. A layer generally consists of a fixed number of neurons, represented by a weight matrix, $\boldsymbol{w}$, and a bias vector, $\boldsymbol{b}$, that take some kind of vectorized input $\boldsymbol{z}$ and calculate an output $\boldsymbol{a}$ as

$$\boldsymbol{a} = g(\boldsymbol{z} \cdot \boldsymbol{w} + \boldsymbol{b}) \tag{2.3}$$

where $g(\cdot)$ is an nonlinear activation function. The simplest form of a neural network has one neuron that takes a scalar input x and outputs a scalar y. For this case, the dimensions of the w- and b-matrices would both be $\mathbb{R}^{1\times1}$. Neurons can be stacked on top of each other to form layers and next to each other to increase layer depth. In a fully connected NN, the output of all neurons in a layer is connected to the next layer as can be seen in Figure 2.1. In this figure, the input is of dimension 3. The NNs (or subparts of NNs) consisting solely of fully connected layers are commonly referred to as Multilayer Perceptrons (MLPs).

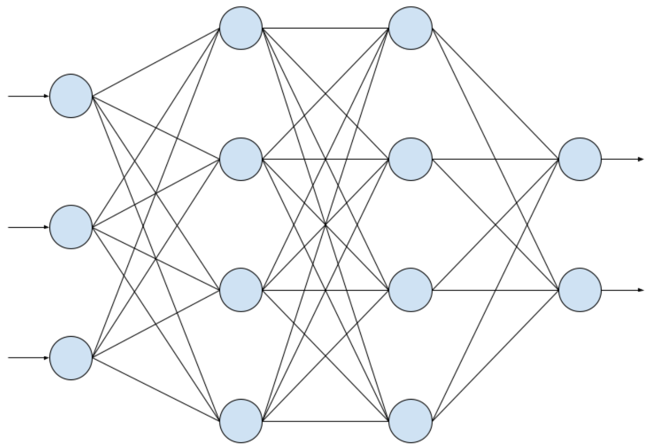


Figure 2.1: 2-layer fully connected artificial neural network. Each layer has 4 neurons. Input size (dimension of $\boldsymbol{x}$) is 3 and output size (dimension of $\boldsymbol{y}$) is 2

2.3.1 Convolutional neural networks

Convolutional neural network layers are commonly used for processing images or other regularly sized inputs of multiple dimensions that have a strong local connection between sections of the input. A convolutional layer takes a fixed-size input and performs convolutional operations on it using fixed-size filters. Convolutional layers, just like MLPs contain weights and biases.

To illustrate how convolutional layers work, let us look at a 2D convolutional layer example. The network takes an image of fixed dimensions $(width) \times (height) \times (c)$ (where c is the number of image channels) as input and applies a convolutional filter of size $3x3$ to each pixel in the image along with its closest neighboring pixels. In our toy case, the input is of size 4x4x1 and consists of the pixels $p_{i,j}$ where i is the horizontal coordinate and j is the vertical coordinate. Such a filter contains 10 learnable parameters; one for each "pixel" in the filter along with a bias that is shared for all filter pixels. The output $y_{k,l}$ will be of size $(width - 2) \times (height - 2)$, which would be 2×2 in our toy case. The height- and width-reductions are a consequence of the filter not being applicable to parts of the image where one or more filter pixels would end up outside the input image, and this is commonly dealt with by zero padding images in order to be able to keep the original dimensions.

If the filter weights are defined as $w_{i,j} \quad \forall i, j \in \{1, 2, \ldots, s\}$ where s is the filter size along with the bias b, in this case 3, the first convolutional operation of the filtering operation above can be described as

$$y_{1,1} = \sum_{i=1}^{3} \sum_{j=1}^{3} p_{i,j} \cdot w_{i,j} + b, \tag{2.4}$$

or more generally as a matrix multiplication

$$y_{i,j} = p_{(i-1:i+1),(j-1:j+1)} \cdot w + b. \tag{2.5}$$

It is common to add more filter channels in a convolutional layer. If a convolutional layer has c channels, this means it contains n filter which will each be applied to all of the output channels of the previous layer, producing an output of $(width - 2) \times (height - 2) \times (c)$, where c is the number of channels in the convolutional layer. A layer of 3×3 filters like this will contain $c \cdot (3 \cdot 3 + 1)$ learnable parameters.

Convolutional layers work similarly in 3D as in 2D but are not used as frequently due to the rapidly increasing computational complexity with respect to the input- and filter-sizes. Some adaptations like Sparse convolution [12] and Submanifold convolution [13] have been proposed to overcome this problem, and these are also commonly used within the field of 3D OD.

2.3.2 Supervised learning

NNs are trained by feeding data into a network and tasking it with minimizing a loss function $\mathcal{L}(\hat{\theta}, \theta)$ where $\hat{\theta}$ is the NN output and θ is the ground truth. Common loss functions in OD include L1 loss,

$$\mathrm{L1}(\hat{\theta}, \theta) = |\theta - \hat{\theta}|, \tag{2.6}$$

for regression and binary cross entropy loss (BCE) for classification

$$\mathrm{BCE}(\hat{\theta}, \theta) = -\Big(\theta \log(\hat{\theta}) + (1 - \theta) \log(1 - \hat{\theta})\Big). \tag{2.7}$$

The minimization procedure is realized using an optimizer and most common optimization algorithms used in deep learning are variants of gradient descent. Gradient descent is an iterative algorithm and the simplest form of gradient descent can be described by the equation

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \gamma \nabla F(\boldsymbol{x}_n) \tag{2.8}$$

where $\boldsymbol{x}$ is the parameters of the multivariate function F, n is the iteration step, γ is a step length or learning rate and ∇F is the gradient of F.

Backpropagation is an efficient way to calculate the gradient of a loss function and is the most common approach for learning in artificial neural network applications. Backpropagation in neural networks was first introduced in 1986 [28]. It makes use of the multivariate chain rule

$$\frac{d}{dt}F(x(t), y(t)) = \frac{\partial F}{dx}\frac{\partial x}{dt} + \frac{\partial F}{dy}\frac{\partial y}{dt}, \tag{2.9}$$

and calculates the partial derivatives of the NNs parameters layer by layer, sweeping backward through the network. Since the derivatives are calculated in a backward sweeping manner layer by layer, it is easy to parallelize these calculations and speed up the algorithm.

2.4 LiDAR OD model architectures

LiDAR OD models are commonly divided into *backbones* and *detectors*. This section provides a brief explanation about their differences and how they operate on the point cloud data.

2.4.1 Backbones

Three different types of backbones are generally used in LiDAR OD models. Pointwise backbones were popular at the inception of LiDAR OD, but are less popular nowadays due to their inability to effectively scale to high-resolution point clouds that contain a massive amount of points. Pillarization backbones are commonly used when inference speed is of great importance, while voxelization networks tend to perform better in terms of detection score, at least for outdoor LiDAR OD challenges.

Pointwise

Pointwise backbones generally demand a fixed number of points as the input. As a result of this, they require some kind of downsampling method to be applied to the initial set of points in a point cloud that is produced by a modern LiDAR sensor before they can be applied. This is usually done using FPS, which has a huge computational load when applied to big sets of points.

The first successful method to apply pointwise backbones was PointNet [25]. It was used to learn pointwise features directly from input point clouds to perform several different tasks on indoor point cloud datasets. The key component in the PointNet backbone is a max pooling operation that helps the network select points of interest while filtering out those that are not as important, all while encoding the reason for selecting the "interesting" points into features. The final fully connected layers of the model can then aggregate these features to make classification decisions. PointNet structures are also commonly used for object refinements in two-stage detectors by utilizing them on proposed regions of interest in the point clouds.

Pillarization

Pillarization was first introduced by [18] and is technically a voxelization technique. The point cloud is divided into evenly sized pillars along the x- and y- dimensions where the pillar size is a hyperparameter tuned during training. Each point in the pillar is then augmented with information regarding the offset of the point from the arithmetic mean of all points in the pillar, and also the offset in x- y- dimensions from the pillar center. Due to the sparse nature of the point cloud, a maximum number of non-empty pillars is enforced along with a maximum number of points per pillar. If the limit for maximum number of points or pillars is violated, random sampling is applied to achieve the correct number of points or pillars.

Each point in the pillar is further processed using a simplified version of PointNet [25]. A max pooling operation over all the processed points in the pillar is used to reduce the dimension further. These operations on all pillars result in a (C, L, W) tensor where C is the feature dimension acquired by the simplified PointNet, L and W indicate the length and width of the pillarized point cloud.

Voxelization

Voxelization for 3D OD was first introduced by VoxelNet [37]. The point cloud is divided into equally sized voxels where voxel size is a hyper-parameter. The points which reside in each voxel are grouped together for further processing. To handle the varying point density in the voxels, a fixed number of T points are randomly sampled in those voxels which contain more than T points. The voxels containing less than T points are instead zero-padded.

2.4.2 Detectors

The detectors, or *heads*, of the model can be divided into *single stage* and *multi stage* detectors. The main difference between the two is whether there is a box refinement process or not as mentioned in Section 1. We also differentiate between *Anchor-based* and *Anchor-free* detectors. Here lies a more fundamental difference between the two which will be presented below.

Anchor-based detection

Many of the initial efforts within 3D OD utilized anchors to define bounding boxes. This also is a technique commonly used within 2D OD [2]. It works by predefining a big number(usually thousands) of anchor boxes for each class in each scene. Due to the rapidly increasing complexity and the sheer amount of required anchor boxes when translating this technique to 3D scenes, these anchors are commonly proposed in the 2D Birds Eye View (BEV) for 3D OD. In the case where 3D anchor boxes are used, they are commonly fixed with respect to sizes in all three dimensions (length, width, height) [34]. Each ground truth in a scene is then assigned to the anchor box that produces the highest Intersection over Union (IoU), as long as the IoU is larger than a predefined threshold.

Anchor-free detection

Anchor-free OD models get rid of the anchor boxes altogether, usually in favor of some kind of object center prediction model [6, 35]. The object center prediction outputs keypoint heatmaps for each specific class. The center predictions and features extracted around them can then be used to regress all of the remaining parameters such as length, width, height and orientation.

2.5 Datasets

Since the KITTI dataset was released, several advancements in datasets for autonomous driving have been made to help drive the development of autonomous vehicle solutions. This section aims to cover the most commonly used datasets and their characteristics.

2.5.1 KITTI

The KITTI dataset for autonomous driving was introduced in 2013 [11] and was for a long time the go-to dataset for 3D OD in outdoor scenes. The test vehicle used to record the data for this dataset was equipped with two grayscale PointGrey Flea2 video cameras, a Velodyne HDL-64E LiDAR and a GPS/IMU localization unit. The 3D OD challenge presented in their paper supplies 7481 training images and 7518 testing images along with their respective point clouds. These are spread over 50 scenes, resulting in a total of about 80k labeled objects. The challenge performance is evaluated by the PASCAL criteria [9]. For the 3D OD challenge, it contains three different classes; *car*, *pedestrian* and *cyclist*. The individual samples for these classes are each split into one of three different difficulties (*easy, medium* or *hard*) depending on their 2D bounding box heights, occlusion level and truncation. True positives are defined as bounding boxes with more than 0.7 BEV IoU for cars and 0.5 BEV IoU for pedestrians and cyclists. Regarding the class balance, cars account for 82.99% of the annotated objects, with pedestrians and cyclists at 12.75% and 4.24% respectively [19].

The entirety of the KITTI dataset is recorded in Karlsruhe, Germany, and only contains recordings from sunny days, thus leaving a lot to be desired regarding dataset diversity. Due to this fact along with it only containing annotations in front of the ego vehicle and a relatively low number of scenes available in the KITTI dataset, it is no longer commonly used for training and evaluating 3D object detectors at the time of writing.

2.5.2 nuScenes

The nuScenes dataset was introduced in 2019 [3] by nuTonomy. It is claimed by the authors to be the first dataset to carry the full autonomous vehicle sensor suite and includes six cameras, five radars, one LiDAR and GPS/IMU localization. Each sensor modality has a 360° field of view and the LiDAR is a 32-beam version producing about 32K points per 360° scan. The data is recorded during different weather conditions both in Boston and Singapore, resulting in a more diverse environment than the ones provided by KITTI and WOD [23]. NuScenes consist of 1000 sequences, annotating 10 different classes for the 3D OD challenge. The challenge is mainly evaluated using *nuScenes Detection Score* (*NDS*), which is a weighted sum of the mAP and a number of other metrics that are only calculated for the true positive matches. Unlike for KITTI, the annotations are not categorized by difficulty. Also, true positives are not matched using IoU, but instead by the distance between a prediction box center and a ground truth box center. This is done for four different distances (0.5m, 1m, 2m and 4m) and the final AP is an average of the AP for all these distances. If more than one prediction is within a specific center distance, the one with the highest confidence score is chosen while the others are discarded.

The point clouds of nuScenes are annotated at 2Hz while the LiDAR operates at 20Hz. The annotated point clouds are commonly referred to as *frames* while the non-annotated ones are called *sweeps*. Due to the sparsity of the point clouds produced by the 32-beam LiDAR sensor, the sweeps are usually concatenated with the help of localization data to produce more dense point clouds. A downside of nuScenes is that it suffers from severe class imbalance, which is mostly caused by the number of classes represented. This is usually managed by utilizing data augmentations or specialized loss functions [38].

2.5.3 Waymo Open Dataset

The Waymo Open Dataset (WOD) was introduced shortly after nuScenes in a collaboration of Waymo LLC and Google LLC [30]. It is recorded using a sensor suite of five LiDARs, five cameras and a GPS/IMU localization system. Four of the LiDARs are of the short-range type (restricted to 20m range). These are placed at the corners of the test vehicle used to record the data, while a mid-range (restricted to 75m) LiDAR is placed on the top of the roof, resulting in a total of about 177K points per scan. The data is recorded in the areas of Phoenix and San Francisco, USA. Like for nuScenes, WOD consists of 20s sequences of data. There are 798, 202 and 150 scenes for training, validation and testing respectively. Regarding classes, WOD is limited to *vehicles*, *pedestrians*, *cyclists* and *signs*. While this alleviates some of the class imbalance seen in nuScenes, there is still a pretty big discrepancy between the number of annotated objects (6.1M vehicles, 2.8M pedestrians and 67k cyclists).

Similarly to KITTI, WOD uses two levels of difficulty for each detection sample. The metrics for the harder level, LEVEL_2, accumulate and thus include all samples for LEVEL_1 as well. LEVEL_2 is defined by bounding boxes that contain less than

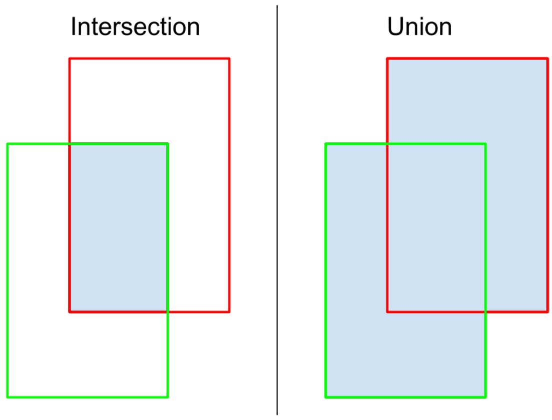


Figure 2.2: Examples of intersection and union used to calculate IoU. The predicted bounding box A is represented in red, and the ground truth bounding box B is represented in green. The intersection and union areas are marked in blue.

five points, or bounding boxes that are manually labeled as "hard". As for challenge scores, WOD evaluates each difficulty level using standard AP, including all vehicles with 2D BEV IOU of more than 0.7 and all pedestrians with a 2D BEV IOU of more than 0.5. However it also uses a metric called AP Heading which puts emphasis on the heading direction of the true positives. The WOD creators argue that this is a critical aspect of 3D OD for autonomous driving purposes. NuScenes uses a similar approach with their Average Orientation Error, but it is not weighed equally high in their NDS.

2.6 Metrics

To be able to compare and evaluate the performance of different OD models some metrics need to be defined. Commonly used metrics for 3D OD include mAP, IoU and NDS.

IoU for 3D bounding boxes is calculated as

$$IoU(A, B) = \frac{A \cap B}{A \cup B}, \tag{2.10}$$

where A is a predicted bounding box and B is a ground truth bounding box. $A \cap B$ is the volume of the bounding box intersection and $A \cup B$ is the union. IoU is therefore a scalar value between zero and one. A visual example in 2D can be seen in Figure 2.2.

Matching of predicted bounding boxes and ground truth bounding boxes can be done in different ways. A common approach is to use an IoU matching where a match is defined by a lower limit on the IoU. NuScenes utilizes another approach where the matching is defined by using prediction confidence and 2D center distance d between the prediction and the ground truth bounding box. The matching is done on the ranges $d = [0.5,\ 1.0,\ 2.0,\ 4.0]$m respectively. Their reasoning for using this instead of using an IoU threshold is that they want to decouple the orientation and

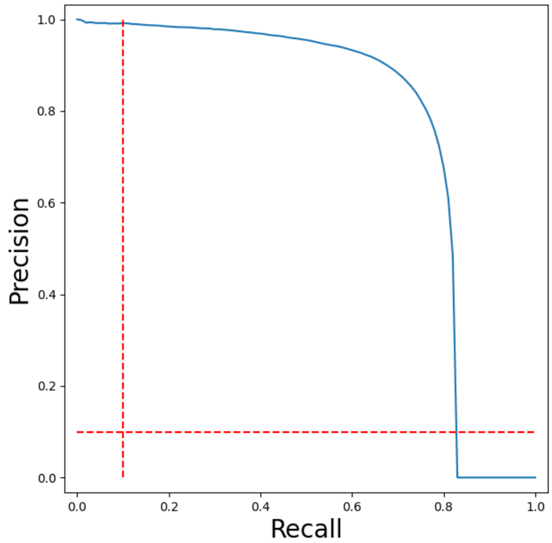


Figure 2.3: A Precision-Recall curve describing the performance of a binary classifier over different confidence intervals. The dashed red lines are filtering limits used to reduce noise in low precision/recall areas which are specifically used in the nuScenes OD challenge.

scale from detection. Another reason is that classes with a small 2D area footprint will have 0 IoU even for a minor translation error.

Precision and recall are used to measure the quality of the detections by calculating the AP. Precision is calculated as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{2.11}$$

where TP is a set of true positive matches and FP is a set of false positives. Recall is calculated as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{2.12}$$

where FN is a set of false negatives. AP is calculated as the normalized area under the precision-recall curve, which is a graphical representation of a binary classifier's performance over different confidence values. An example can be seen in Figure 2.3. The mAP is calculated as the mean over all classes and distance thresholds as

$$\text{mAP} = \frac{1}{|\mathcal{C}||\mathcal{D}|} \sum_{c \in \mathcal{C}} \sum_{d \in \mathcal{D}} \text{AP}(c, d), \tag{2.13}$$

where $\mathcal{C}$ is the set of all classes and $\mathcal{D}$ is the the set of matching distance thresholds. The true positive metrics TP_m used are average translation error (ATE), average scale error (ASE), average orientation error (AOE), average velocity error (AVE) and average attribute error (AAE). TP_m is only defined for correct matches at a fixed 2D center distance $d = 2.0$. ATE is the 2D Euclidean center distance between the ground truth bounding box and the matched predicted bounding box. ASE is calculated after box centering and alignment as $1 - IoU$. The smallest difference in yaw angle between the predicted and ground truth bounding box is used as AOE. AVE is the L2-norm of the difference between ground truth and predicted velocities and AAE is defined as $1-acc$ where acc is the attribute classification accuracy. The mean of each TP_m (mTP_m) is calculated as

$$mTP_m = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} TP_m(c). \tag{2.14}$$

The nuScenes detection score is the primary metric used on the nuScenes leaderboard of 3D OD and is defined as

$$\text{NDS} = \frac{1}{10}\left(5 \cdot mAP + \sum_{tp \in TP_m} \Big(1 - \min(1, mTP_m(tp))\Big)\right). \tag{2.15}$$

2.7 Data augmentations for LiDAR OD

The nature of LiDAR point clouds in combination with the OD task allows for a wide range of data augmentations. This section will structure the commonly used data augmentations and describe them in detail. Following [15], the augmentations will be divided into one of two categories; *Global* and *Local* augmentations.

2.7.1 Global augmentations

Global data augmentations consider the ones that augment the whole point cloud altogether. The most common ones include *Global Translation*, *Global Scaling*, *Global Rotation*, *Global Random Flip*, *Ground Removal* and the previously mentioned *Sweep Concatenation*.

Consider the case of describing the set of all points in a point cloud $\mathcal{P} = \{p_1, p_2, ..., p_n\}$ with n number of points and m number of corresponding ground truth annotations $\mathcal{A} = \{a_1, a_2, ..., a_m\}$. Each point $p_i \in \mathcal{P}$ can subsequently be described as $p_i = [x_i, y_i, z_i, r_i]$. The reflectivity return r_i will be dropped in this notation from here onward as none of the data augmentations alter it. Additionally, ground truth annotations are described using 3D bounding boxes $a_j = [x_j^c, y_j^c, z_j^c, w_j, l_j, h_j, \theta_j] : a_j \in \mathcal{A}$.

Global translation is usually implemented by sampling translations Δ_x, Δ_y and Δ_z individually from a normal distribution before applying these to all points as

$$p_i^{trans} = [x_i + \Delta_x, y_i + \Delta_y, z_i + \Delta_z] \quad \forall i \in \{1, 2, ..., n\}. \tag{2.16}$$

Additionally the center coordinates of the bounding boxes x_i^c, y_i^c and z_i^c also need to be adjusted accordingly

$$a_j^{trans} = [x_j^c + \Delta_x, y_j^c + \Delta_y, z_j^c + \Delta_z, w_j, l_j, h_j, \theta_j] \quad \forall j \in \{1, 2, ..., m\}. \tag{2.17}$$

Global Rotation essentially means rotating every point p_i around the upright axis (in most cases the z-axis) by an angle α that is usually drawn from a uniform distribution with limits $\pm\beta$. The bounding boxes a_j need to be rotated as well, which requires the bounding box angles θ_j to be modified, resulting in

$$\begin{aligned} a_i^{rot} = [x_j^c \cdot \cos(\alpha) - y_j^c \cdot \sin(\alpha), x_j^c \cdot \sin(\alpha) + y_j^c \cdot \cos(\alpha), z_j^c, w_j, l_j, h_j, \theta_j + \alpha] \\ \forall j \in \{1, 2, ..., m\}, \end{aligned} \tag{2.18}$$

and

$$\begin{aligned} p_i^{rot} = [x_i \cdot \cos(\alpha) - y_i \cdot \sin(\alpha), x_i \cdot \sin(\alpha) + y_i \cdot \cos(\alpha), z_i] \\ \forall i \in \{1, 2, ..., n\}. \end{aligned} \tag{2.19}$$

Global Random Flip is an augmentation that refers to flipping the point cloud over one of its Cartesian coordinate axes. Flipping over the z-axis is usually omitted due to the irregular point cloud structures that are caused by doing this. However, flipping over the x- and y-axes can be done both separately and simultaneously. In the case of flipping over the forwards facing x-axis, the points in $\mathcal{P}$ will be translated as

$$p_i^{xflip} = [x_i, -y_i, z_i] \quad \forall i \in \{1, 2, ..., n\}. \tag{2.20}$$

In addition to requiring the negated y-coordinates, the ground truth annotations also require alteration of their angles when flipped. For the same x-axis flip case, the annotations will be modified as

$$a_j^{xflip} = [x_j^c, -y_j^c, z_j^c, w_j, l_j, h_j, (\theta_j + \pi) \text{mod } 2\pi]. \tag{2.21}$$

Similar equations with minor differences in ground truth annotation angles and signs are used for flipping over the y-axis and flipping over both axes simultaneously.

Global scaling simply refers to multiplying all the coordinates of a point p_i by a scaling factor s, also drawn from a uniform distribution with bounds $1 \pm S$,

$$p_i^{scaled} = [x_i \cdot s, y_i \cdot s, z_i \cdot s] \quad \forall i \in \{1, 2, ..., n\}. \tag{2.22}$$

The ground truth annotations will be modified according to

$$a_j^{scaled} = [x_j^c \cdot s, y_j^c \cdot s, z_j^c \cdot s, w_j \cdot s, l_j \cdot s, h_j \cdot s, \theta_j] \quad \forall j \in \{1, 2, ..., m\}. \tag{2.23}$$

Finally, *Sweep Concatenation* was described briefly in section 2.5.2. For nuScenes, precise localization information is achieved by utilizing Monte Carlo Localization using LiDAR and odometry data. These localization measurements allow for good ego state estimation (<10cm error) which can be used to concatenate point clouds across multiple sweeps without these turning out to look dragged out due to ego movement [3]. It is however not able to account for the movement of surrounding objects, sometimes causing specific objects to have a dragged-out look. This phenomenon is visualized in Figure 2.4. In order to combat this problem and also provide the neural networks trained on the dataset with some temporal information, each point p_i is also augmented by concatenating them with a timestamp δ_i which describes how far from the key frame it was recorded

$$p_i^{aug} = [x_i, y_i, z_i, r_i, \delta_i]. \tag{2.24}$$

2.7.2 Local augmentations

Local augmentations are generally utilized less frequently than the global ones described in the previous section, with the exception of GTDS. Some of them have however been proven by other articles to have a positive effect on model performance [15], and we will thus be going over them in this section. The most commonly utilized ones that are known to us include *Local Translation*, *Local Rotation*, *Local Scaling* and the previously mentioned GTDS.

Local Translation is similar to Global Translation but is only applied to ground truth annotation and the points inside those. Just like for Global Translation, if this augmentation is used the translations are usually sampled independently from a normal distribution for x-, y- and z-coordinates.

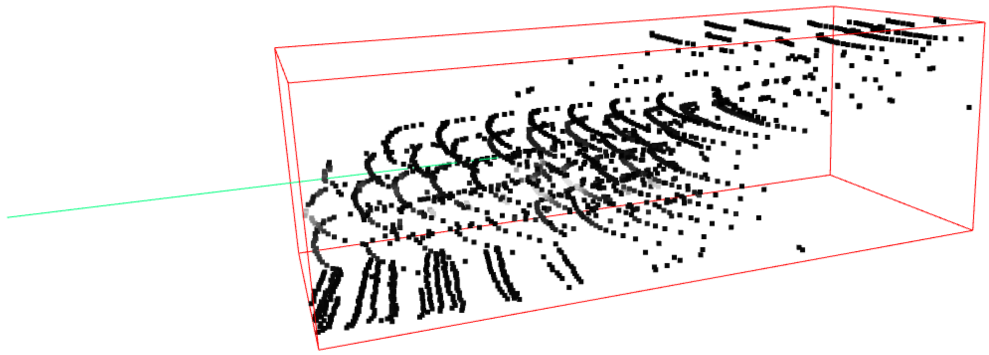


Figure 2.4: Moving Car at distance ~10m from the LiDAR sensor in the nuScenes dataset. Sweep concatenation of 10 sweeps used. The direction of movement is visualized by the green line.

Local Rotation implies rotating the points inside a ground truth annotation's bounding box around one of its center axes. Most commonly, this is done only around the upwards-facing z-axis. If applied, it would once again be done by uniformly sampling an angle α on an interval $\pm\beta$. It is important not to choose a value of β that is too high, as this could possibly alter the geometric information of an object too much, e.g. turning an object that should be convex at its current position into a concave one if rotated by π radians. The calculations applied to the points p_k (assuming the center coordinates of the annotation are moved to [0,0,0]) inside the ground truth annotation a_j would be,

$$p_k^{rot} = [x_k \cdot \cos(\alpha) - y_k \cdot \sin(\alpha), x_k \cdot \sin(\alpha) + y_k \cdot \cos(\alpha), z_k], \tag{2.25}$$

and the corresponding ground truth annotation change,

$$a_j^{rot} = [x_j^c, y_j^c, z_j^c, w_j, l_j, h_j, (\theta_j + \alpha) \mathrm{mod}\ 2\pi], \tag{2.26}$$

where mod is the modulo division operation.

Once again similar to its global counterpart, *Local Scaling* would be implemented by sampling a scaling factor s uniformly on an interval $1 \pm S$ before multiplying the coordinates of all the points inside a ground truth annotation bounding box the factor s (assuming $[0, 0, 0]$ at $[x_j^c, y_j^c, z_j^c]$) along with scaling the length, width and height of the annotation by the same factor.

Finally, *GTDS* was first introduced by [34] and has since been applied when training the majority of new SOTA LiDAR OD models. It collects all the ground truth annotations in the whole dataset along with their corresponding points and puts them into an offline database. This database can be sampled from to paste objects into other point clouds during training time in order to simulate more objects in each point cloud. GTDS was proposed by its inventors to be able to increase both converge speed and final detection results significantly. In order to minimize the possible negative effects of this augmentation, it is usually implemented in combination with a filter for minimum points inside each sampled annotation, a collision check in order to avoid overlapping objects when pasting, a difficulty-level filter (only applicable for WOD and KITTI) and class-specific sampling frequency parameters to allow for some mitigation of dataset class imbalance. It also helps even out the distribution between foreground and background points in each training sample point cloud.

In their 2021 paper, [31] proposed a GTDS modification called *Fade*. They argued that while the original GTDS augmentation helps the networks trained for LiDAR OD converge faster and improves their final accuracy, it skews the object distribution which may result in unwanted predictions. Thus, they trained their model for 20 epochs but disabled GTDS for the last 5 epochs, improving mAP by a significant amount. This has later also been utilized effectively by models like TransFusion [1].

Apart from these relatively common and general approaches to local data augmentations, there also exist more specialized and complicated ones. An example of this is frustum-based local augmentations, which were proposed in [7]. Just like the name suggests, these operate on a frustum of the points inside a bounding box. A frustum is a subset of the points in the bounding box which lie within a certain subset of the volume spanned by the box. Frustum augmentations include *Frustum Dropout* and *Frustum Noise*. Other examples include methods that specialize in using GTDS to sample objects in logical locations of the point clouds, for example pedestrians on sidewalks, or handling occlusion while pasting objects into scenes [29].

2.8 Farthest Point Sampling

Farthest Point Sampling (FPS) is an iterative approach to sample evenly spaced points in a set of points in arbitrary dimensions. For the case of simplicity, we will illustrate this in 2D but in practice it works for any number of dimensions.

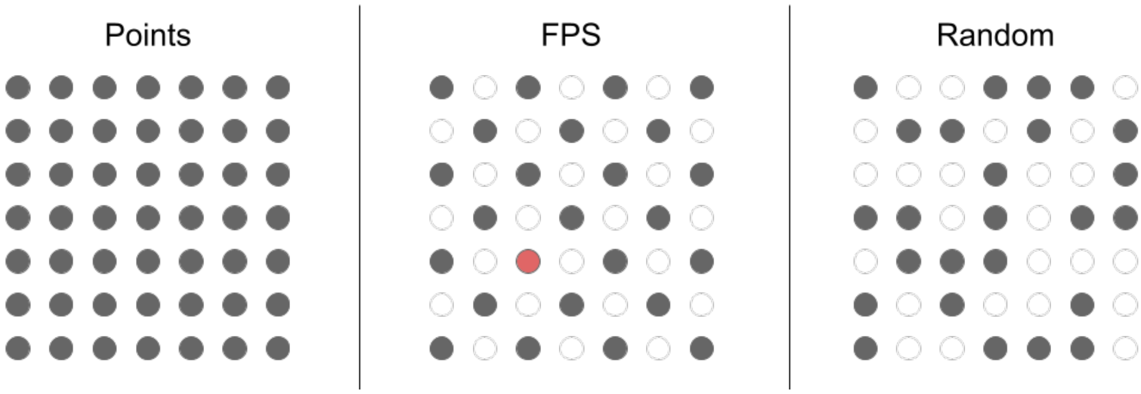


Figure 2.5: Comparison between points sampled from a 2D point grid using FPS and random sampling. The initially random sampled point in the FPS algorithm is marked in red.

Given a set of n points $\mathcal{P} = \{p_0, p_1, ..., p_n\} \in \mathbb{R}^2$ and the number of points to be sampled N. One point p is picked at random and removed from $\mathcal{P}$. The point p is then added to the set of sampled points $\mathcal{S}$. For each point in $\mathcal{P}$, calculate the shortest distance to any of the points in $\mathcal{S}$. Add the point with the farthest distance from $\mathcal{S}$ to $\mathcal{S}$ and remove it from $\mathcal{P}$. The process is repeated until $\mathcal{S}$ consists of N points. A visual example of the discrepancy between FPS and random sampling can be seen in Figure 2.5

3
Methods

With the aim of improving LiDAR object detection, this thesis focuses on investigating the effect of data augmentations and improving the GTDS technique that is commonly used while training models for this cause. The latter is mainly done by extending the functionality of the original GTDS in order to enhance data diversity and enable training to focus on more difficult objects, especially ones far away, instead of overtraining on closer objects that are generally easier for the models to learn to detect. We evaluate three different methods of modifying GTDS:

1. Random Factor 2D range increase with downsampling (RFD)
2. Random Range sampling for 2D locations to paste objects into (RRD)
3. Hard Range Filtering by object range without any downsampling (HRFiS)

Each of these techniques will be described in their respective subsections below. For all of our proposed methods, the number of sampled objects for each point cloud is consistent with the ones in [35], which tend to sample uncommon classes more frequently. These values are presented in Table 3.1. Keep in mind that the actual number of sampled objects is slightly lower due to objects being filtered out when they overlap with another object or when they end up outside the detection range.

Table 3.1: Number of objects sampled into each point cloud during training.

Class	**Num. Sampled**
Car	2
Truck	3
Bus	4
Trailer	6
Construction Vehicle	7
Motorcycle	6
Bicycle	6
Pedestrian	2
Traffic Cone	2
Barrier	2

3.1 Dataset selection

The nuScenes dataset is selected for this work as it has a similar amount of LiDAR data as WOD but with a wider range of classes. The long tail distribution of classes makes good performance for all classes a challenging task. The number of classes in the dataset opens up for performance improvements on the more difficult and less frequent classes like Construction Vehicle and Bicycle. NuScenes also comes with additional benefits compared to WOD. Since the dataset is collected both in the US and in Singapore, the variation of shapes is greater compared to a dataset collected in one country. There is also a larger portion of data collected during rainy days compared to WOD which adds to making nuScenes a more diverse dataset [23].

3.2 Model selection

A major factor in deciding which models to use for the experiments in this thesis is which ones are ready with open source configurations and support for the dataset being used. Although this limits the model choice quite significantly, there are still quite a few options available. With that in mind, we want to examine the effects of our proposed modified GTDS and a few other already established data augmentations techniques on models that have been included in several previous research articles and are commonly used as baselines for experiments like these. We also find it important to evaluate our methods using both one-stage and two-stage models. Hence, the models that will be used for evaluation in this thesis are PointPillars and CenterPoint.

3.2.1 PointPillars

First presented in 2018, PointPillars [18] was a pioneering work for 3D object detection, especially in terms of the tradeoff between accuracy and inference speed. At the time it was proposed, it achieved SOTA results on the KITTI dataset while operating at a 62Hz frequency on a single NVIDIA GTX 1080ti GPU. The pillar-based backbone that effectively enables the use of 2D convolutions for 3D object detection tasks is frequently used to this day, especially for models which aim for real-time inference speed.

For this study, we use a baseline implementation of PointPillars for the nuScenes dataset that is provided as-is by MMDetection3D [8] with the addition of GTDS. Other data augmentations used for the baseline include *Global Rotation* with limits $\pm\frac{\pi}{8}$, *Global Scaling* with limits 1 ± 0.05, *Global Flipping* only over the x-axis with a probability of 50% and *Multi-sweep concatenation* utilizing 10 sweeps in total. We also train a model which applies one version of our proposed modified GTDS and apply *Fade* augmentation separately to both of these training runs, applied for the last 25% of the epochs.

The PointPillars models that are trained for the experiments in this thesis use a multi-step learning rate schedule for 24 epochs (see Figure 3.1) and a batch size of 4

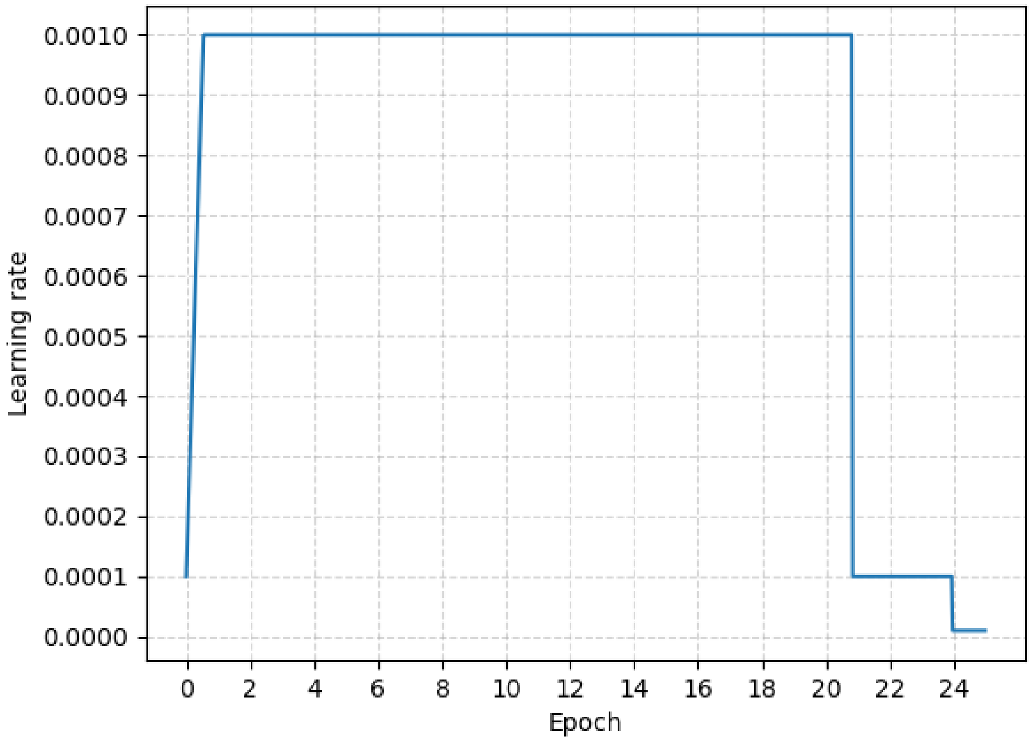


Figure 3.1: Visualization of multi-step learning rate schedule used for PointPillars training.

for each GPU, resulting in an effective batch size of 16 when trained on four GPU:s in parallel. The GPU:s used for all training runs in this thesis are NVIDIA A100. Losses used for training this model include Cross Entropy Loss for the classification task and Smooth L1 Loss for regression targets as per the baseline implementations available in MMDetection3D [8].

3.2.2 CenterPoint

CenterPoint is a two-stage model that was introduced by [35] in 2021 and has since become a common baseline within the field of LiDAR object detection. The authors propose getting rid of an anchor-based detector in favor of their anchor-free heatmap detector which aims to detect object centers in BEV as an auxiliary task. The heatmap hotspots are then sent to one of several detection heads that regress other important attributes like 3D orientation, 3D object size and velocity. Finally, a second network stage is used to refine these predictions by utilizing additional point features within the initially predicted object box. At the time of publication, CenterPoint achieved SOTA results on the nuScenes 3D object detection task with a score of 65.5 NDS for the version using a voxel-based backbone.

For the experiments in this thesis, the voxel-based backbone is replaced with a pillar-based one in order to reduce training time. Just like for PointPillars, the basis for the implementation is the one from MMDetection3D. In order to properly evaluate the general performance of GTDS on the nuScenes dataset, we start off by training CenterPoint without using GTDS at all. This has not been thoroughly investigated by any other publications to the best of our knowledge, even though [26] investigate the effect of data augmentations for models trained on nuScenes specifically for the Car class while [15] reviews its effect on the KITTI dataset. The augmentations that are present for this training run follow the principles from section 2.7 and are as follows; *Global rotation* with limits $\pm\frac{\pi}{8}$, *Multi-sweep concatenation* utilizing 10 sweeps in total and *Random flip* over both the x- and y-axes with a 50% probability.

The CenterPoint models for these experiments are trained using a 20-epoch cyclic learning rate schedule that starts at 10^{-4} and peaks at 10^{-3} at the start of epoch 9. It then gradually decreases and hits the minimum learning rate of 10^{-8} at the end of epoch 20. A visualization of this can be seen in Figure 3.2. Just as for PointPillars, the model is trained using 4 NVIDIA A100 GPUs in parallel with a batch size of 4 for each GPU. Loss functions used for training are Focal Loss [20] for classification heads and L1 loss for all of the regression targets.

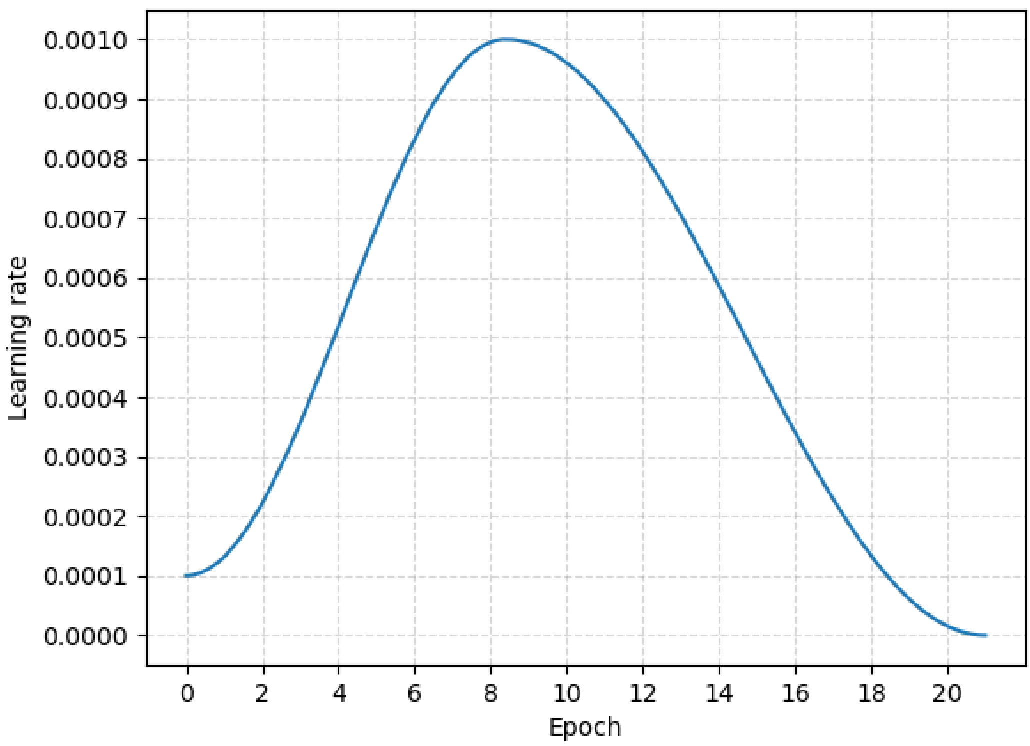


Figure 3.2: Visualization of cyclic learning rate schedule used for training CenterPoint.

3.3 Range specific performance

To be able to evaluate performance on specific ranges separately, we alter the nuScenes validation script to be able to filter out objects by range. This makes it possible for us to calculate the nuScenes metrics described in Section 2.6 for any range interval of our choice. We split the ranges into *short*, *medium*, *long*, where each range represents 1/3 of the maximum detection range for each class available in the nuScenes dataset. The range intervals of each class can be seen in Table 3.2.

Table 3.2: Detection range intervals for each class in the nuScenes dataset. C.V. is used as acronym for Construction Vehicle.

Class	Maximum detection range [m]		
	short	medium	long
Car, Truck, Bus, Trailer, C.V.	[0.00, 16.67]	(16.67, 33.33]	(33.33, 50.00]
Motorcycle, Bicycle, Pedestrian	[0.00, 13.33]	(13.33, 26.67]	(26.67, 40.00]
Traffic Cone, Barrier	[0.00, 10.00]	(10.00, 20.00]	(20.00, 30.00]

3.4 Modifying GTDS

In order to try to improve the GTDS and make it cater more towards long range, we present several versions of modified GTDS techniques in an attempt to improve performance on this range interval. The distinctions for each one compared to the original GTDS will be described below.

3.4.1 Random Factor range increase and Downsampling (RFD)

Our initial idea of how to modify the GTDS extends the functionality of the GTDS sampler by introducing three new parameters for each class available in the dataset. Firstly, downsampling rate R is a class-specific number on the interval $[0, 1]$ which determines how many of the originally sampled ground truths a_j that will be chosen for downsampling. Whether a sampled object should be downsampled and moved is decided by uniformly sampling a number X on the interval $[0, 1]$. In the case where $X \leq R$, the ground truth will be moved and downsampled, while if $X > R$ nothing will be done to alter the object range or points inside it. Along with this comes a Translation Factor (TF), which is given by a vector containing upper and lower limits specific to each class. For example, if the class is Car, $TF_{car} = [1.5, 2.5]$ denotes the lower and upper limits for how much further an object can be moved away in relation to its original coordinates if the object is marked for downsampling after comparing X and R.

When an object is marked for downsampling, another uniformly sampled variable S is generated on the interval defined in TF. Adopting the notation from Section 2.7, this number is then used to multiply the x- and y-coordinates of the marked object,

$$p_i^{DS} = [x_i \cdot S, y_i \cdot S, z_i], \tag{3.1}$$

in order to move it further away. To conform to the number of points an object should have at the new range, the points p_i inside the marked bounding box a_j are reduced according to

$$p_i^{DS} = f(p_i, S), \tag{3.2}$$

where the function $f(\cdot)$ is the downsampling function (in the case of this thesis, Random or FPS). The number of points to use in a downsampled object should theoretically be quadratically proportional to S but due to occlusions, which are much more frequently present at longer ranges, we find through experiments that the number of points left when moving an object further away have a relation to S that is much closer to cubic. The outcome of one of these experiments can be seen in Figure 3.3. Note that when generating the red plot of this figure, aggressive filtering with respect to the number of points in each object was used when setting up the GTDS (minimum points 100 for the Car class, the one visualized), which is why the red samples do not reach as far down as the blue ones. The plots are produced using Monte Carlo Simulations on the original implementation of GTDS (labeled as DBSampler v1 in Figure 3.3) along with the one utilizing RFD (labeled as DBSampler v2 in Figure 3.3).

As a result of these experiments, the number of points k_{DS} used for an object that is marked for downsampling is

$$k_{DS} = floor\left(\frac{k}{S^3}\right), \tag{3.3}$$

where $floor(\cdot)$ is the floor division operator and k is the initial number of points for the object.

As a last addition to RFD, we also add a global flip augmentation to the sampled objects, i.e., flipping these objects within the point cloud over the global x- and y-axes. This is a simple addition implementation-wise and is done in order to further diversify the data that comes out of the RFD. Even though this is partly covered by the global flip augmentation that is also used when training, we suggest that since global flip is applied on all the objects at the same time after sampling with GTDS this may be important to improve the diversity of objects in relation to each other in each specific point cloud used for training. Local flip is implemented with a

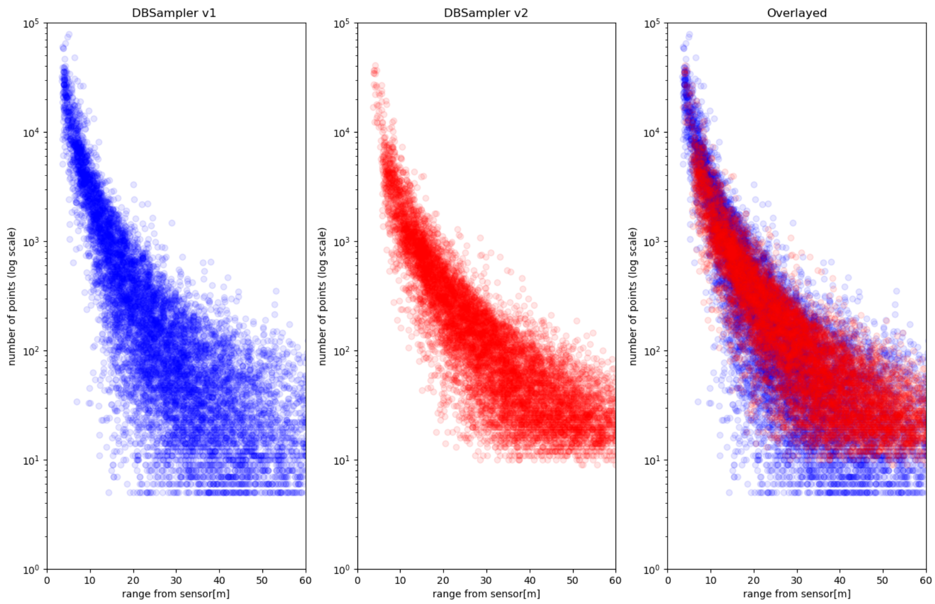


Figure 3.3: Number of points as a function of range from the sensor for original GTDS and modified GTDS. Number of points kept when downsampling cubically proportional to Y. Class *Car*.

probability of 50% for flipping over both the x- and y-axes in all experiments. The bounding boxes and points of flipped objects are modified accordingly to produce realistic objects when flipping.

3.4.2 Random Range generation and Downsampling (RRD)

Secondly, we propose another way to go about modifying GTDS in order to support the NNs to improve long-range detections. Just like for the method described in the subsection above, this one uses a downsampling rate R to determine whether an object from the base GTDS should be downsampled and moved away or not. The difference to the previous method is that the Translation Factor limits TF are removed in favor of uniformly sampling a class-specific BEV distance d for where the downsampled object is supposed to end up. The upper and lower bounds for d are the main tuning parameters for RRD. Once a distance d is generated, it can then be used to calculate how many points the downsampled object should contain according to the cubical relation described in the previous subsection. For an annotation of a sampled object $a_j = [x_j, y_j, z_j, w_j, l_j, h_j, \theta_j]$ with k corresponding points p_i $\forall i \in \{1, 2, .., k-1, k\}$, this is done as

$$S = \frac{d}{\sqrt{(x_j)^2 + (y_j)^2}}, \tag{3.4}$$

and

$$p_i^{DS} = f(p_i, S). \tag{3.5}$$

The number of points that should be used in the downsampled object can then be calculated as in equation 3.3. In the case where the generated distance d is less than the distance that the sampled object is originally at, all of the downsampling steps are skipped.

The reason for introducing this method instead of just tuning the S-parameters of the previous versions is that when trying to bias sampled objects more towards longer ranges, RFD is hard to tune correctly without a lot of the downsampled objects ending up outside the maximum detection range for the nuScenes dataset (Table 3.2). It does however introduce some additional problems regarding the point distribution within the objects, especially when the downsampling function $f(.)$ is set to Random. As opposed to the previous method where the upper limit of S is generally not set higher than 2.5 in order to avoid a lot of the downsampled objects ending up outside the maximum detection range, this method can potentially produce values of S up to 25. This can in turn result in very bad representations of the downsampled objects due to the nature of how points for downsampled objects are chosen when done randomly.

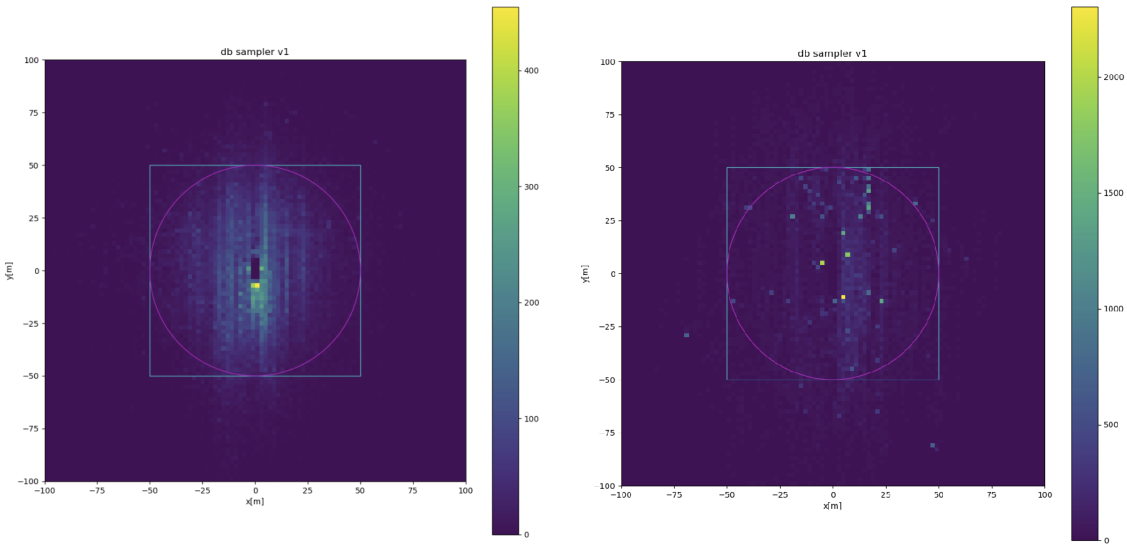
This method is tuned using two different configurations. One where half of the sampled objects are chosen for downsampling and translation away from the sensor (RRDv1), and one where all of the sampled objects are (RRDv2). Both configurations share the same lower and upper limits for d.

3.4.3 Hard Range Filtering Sampling (HRFiS)

Finally, we propose a third, much simpler modification to GTDS. In addition to being able to filter the database samples by number of points at the build phase of GTDS, we add the option of being able to filter objects by their 2D BEV range from the sensor. This enables us to exclusively sample objects that are far away from the sensor without having to downsample any of them. A major drawback of using a method like this is that it might severely limit the data diversity of the sample objects. However, it forces all of the sampled objects to stay true to their point distribution and location in the actual dataset, which is why we find it interesting to evaluate this option as well.

3.5 Tuning modified GTDS

Before training models with the three methods presented in the previous section, some offline tuning work is done using Monte Carlo simulations on the samplers. The number of simulations run for all experiments is 100k. Along with looking at the number of points in objects as a function of range as in Figure 3.3, we also examine the BEV distributions of objects sampled using 2D histograms (also referred to as heatmaps). For an illustration of this, see Figure 3.4. Heatmaps for classes that are not included in this section can be found in Appendix A. The size of each bin in the heatmap is 2x2m. The light blue square represents the voxelization range of the trained model while the pink circle is the maximum detection range for the current class in the nuScenes dataset. The color scale represents how many times an object is sampled within each grid cell of the heatmap. Regarding coordinate systems, the positive y-direction in the heatmap represents forwards of the ego vehicle, i.e., the driving direction.



(a) Heatmap for the Car class. **(b)** Heatmap for the C.V. class.

Figure 3.4: Monte Carlo simulations of sampling the Car and Construction Vehicle classes using the original GTDS sampler with no modifications. Note that the heatmaps do not share the same color bar scale.

This not only gives us a good idea of how objects are generally distributed when using the original version of the GTDS sampler but also of where objects are mainly appearing within the whole dataset. Figures 3.4a and 3.4b introduce two different problems. For the Car class, we observe that the heatmap distribution is heavily skewed towards objects at closer ranges. While it does make sense to train the models using more samples that are close to the sensor if this is what we will later evaluate on, this is probably not the case in a real-world scenario. The reason the data looks like this is rather a combined effect of objects being harder to annotate as they are further away from the sensor along with the increased probability that objects farther away from the sensor are occluded by something that is closer, especially in a city environment. Hence we find it reasonable to bias the sampled objects towards longer ranges in order to try to even out the distribution more regarding range from the sensor.

For the Construction Vehicle (CV) class, we notice a different problem. CVs tend to appear at very specific locations a lot of the time, resulting in a very biased distribution regarding heatmap location. This is likely a side effect of the small sample size of CVs available in the dataset (11,050 as opposed to 339,949 Cars). The same thing is also true for other less frequently appearing classes like Bicycle and Motorcycle. This should in theory be bad for helping the trained models generalize to be able to detect CVs at different locations in relation to sensor position. It makes our choice of uniformly sampling the position of downsampled objects for both RFD and RRD seem like a reasonable choice when it comes to further diversifying training data. It is also the main reason we add the local flip augmentation to both of the methods.

These conclusions lead us to tune the database samplers with two different goals. Firstly, we attempt using modified GTDS to achieve a more even distribution of objects as a function of BEV range from the sensor since the distributions are generally biased towards close objects. Secondly, we aim to perform experiments that bias the sampled objects more heavily towards longer ranges, since shorter ranges are already well represented in the training data before pasting more objects into it with GTDS. Close-range objects should theoretically also require less training data than long-range ones, since the number of points inside these will be drastically higher, making them easier to distinguish.

Thirdly, we aim to use the modified GTDS methods to create training distributions where objects don't end up in very specific places as often, as seen in Figure 3.4b. This is already somewhat mitigated by the *Global Flip* and *Global Rotation* augmentations, but does not take into account where objects tend to end up in relation to each other since these augmentations are always applied after GTDS.

3.6 Estimating training variance

Some variance between different training runs is to be expected due to the stochastic nature of database sampling, other data augmentations and minibatch training. Hence a CenterPoint Baseline and CenterPoint HRFiS model, both with and without fade, will be trained three times each to be able to get a good idea of how the end results on the validation set vary between runs. The mean of the three runs are calculated along with the difference between the best results and the worst one for each class and metric. This is intended to give a rough estimate of how big we can expect the difference for each metric to be in the other experiments performed and provide tools to determine if the results are significant.

4

Experimental evaluation

In this chapter, specific implementation details and results for the various configurations and models are presented. We move on to discuss the results of the different configurations and underlying factors of their performance.

4.1 Implementation details and tuning

Our work is made possible by building upon a few already existing libraries for deep learning, LiDAR object detection and 3D calculations. PyTorch [24] is used as the base library and our work builds directly on MMDetection3D [8].

4.1.1 MMDetection3D

MMDetection3D by OpenMMLab is an open source deep learning library for both indoor and outdoor 3D object detection. It supports single- and multi-modality detectors and mainly builds upon PyTorch and MMDetection[5]. PyTorch is an optimized tensor library that is commonly used for all kinds of deep learning. The variety of models implemented in the MMdetection3D framework opens up for simpler comparison of new concepts and models with older SOTA models.

4.1.2 CenterPoint

First off, we train the *Baseline* of our models, which is the one from section 3.2.2 utilizing the completely unmodified GTDS.
We then move on to the second method described in the Methods chapter, RFD. This is the only method for which downsampling using FPS is evaluated. Hence there are two different models trained for this method. The tuning parameters for the downsampling rate R, the downsampling factor S and minimum amount of points in a sampled object are the same for both runs (Random downsampling and FPS downsampling) and are presented in table 4.1. Although the minimum points filtering might seem aggressive, we find it necessary in order to ensure that objects of high enough quality and within reasonable range are sampled. Local xy-flip is utilized for both runs.

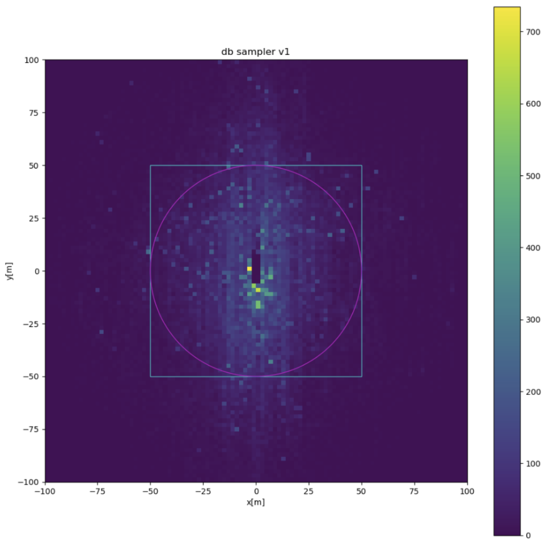
These parameters are tuned mostly using 2D histograms like the ones in Figure 3.4 along with 1D histograms showing BEV range from sensor distributions. The tuning goal for these runs is to produce a more even distribution in regards to 2D

Table 4.1: Tuning parameters for training runs with RFD.

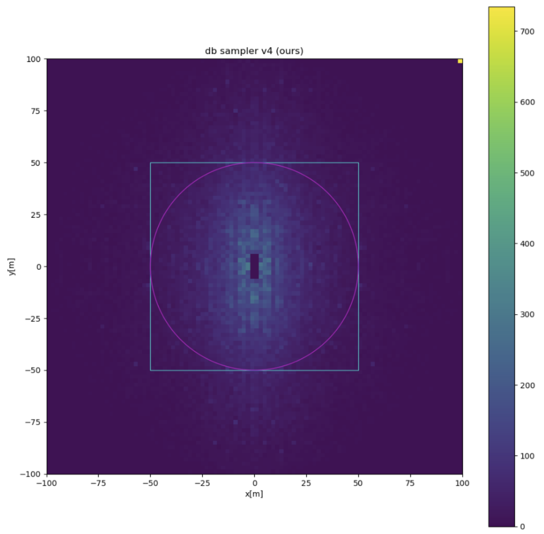
Class	R	S	Min. points
Car	0.9	[1.7, 2.2]	100
Truck	0.5	[1.4, 1.7]	50
Bus	0.5	[1.4, 1.6]	100
Trailer	0.5	[1.5, 1.5]	100
C.V.	0.5	[1.3, 1.8]	100
Ped.	0.5	[1.5, 1.8]	20
Mot.	0.5	[1.4, 1.6]	15
Bicycle	0.5	[1.4, 1.6]	15
T.C.	0.5	[1.5, 1.5]	10
Barrier	0.5	[1.3, 1.5]	15

BEV coordinates of objects within their respective detection ranges, i.e., removing positional bias. Comparisons between the original GTDS and the results of tuning RFD are shown for Construction Vehicle (Figure 4.1) and Truck (Figure 4.2). The total number of sampled objects is the same in both subfigures, but the distribution is much more even in terms of 2D BEV location for RFD.

An important note here is that the final distribution of all annotated objects used to train will not be represented by Figure 4.1b alone, but rather a weighted combination of both of the subfigure distributions. This is because the distribution before sampling additional objects in the point clouds is still approximately represented by Figure 4.1a. The same is true for Figures 4.2a and 4.2b. The final distribution will also be influenced by how many objects of each class are already natively present in the point clouds versus how many are sampled into each point cloud, meaning that for Construction Vehicle it will be more biased towards the representation in Figure 4.1b.

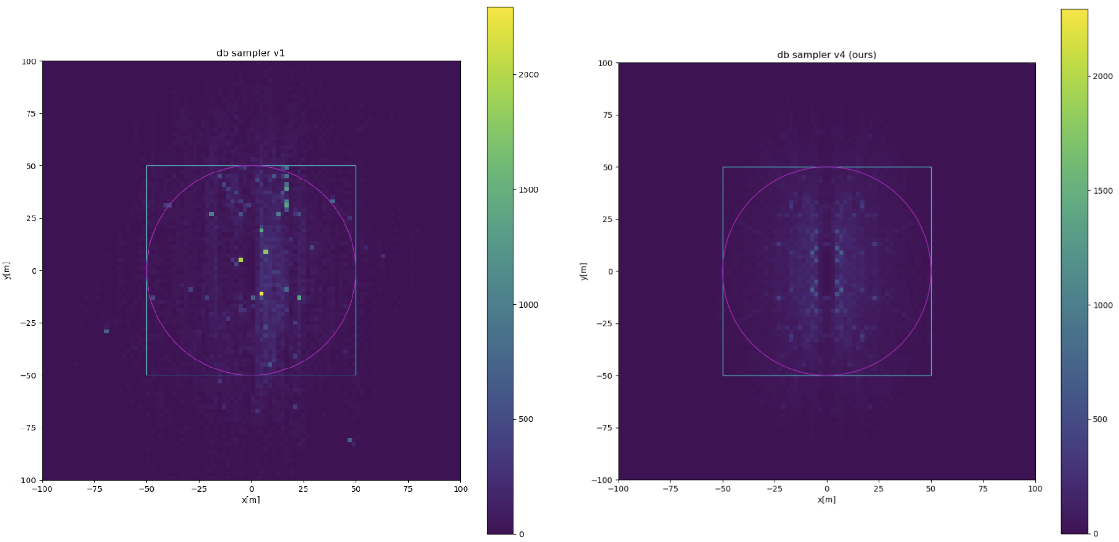


(a) Heatmap original GTDS.



(b) Heatmap RFD tuned.

Figure 4.2: Distributions of Truck with GTDS and RFD. Same total number of sampled objects for both versions with color scale normalized.



(a) Heatmap original GTDS. **(b)** Heatmap RFD tuned.

Figure 4.1: Distributions of Construction Vehicle with GTDS and RFD. Same number of sampled objects for both versions with color scale normalized.

The next method used for training is RRD. This version aims to introduce more bias towards training on long range and is trained using two different tuning parameter setups. One where half of the sampled objects are chosen to be moved away and downsampled (RRDv1), and one where all of them are (RRDv2). The ranges that objects are moved into for both of these training runs lie within the *long* range intervals for each class that were previously presented in Table 3.2. As an extra addition, an online number of points-check is implemented in order to ensure that objects contain at least five points after being downsampled. In the case where they do not, the sampled object is used as-is without moving or downsampling it. A typical class distribution for one of these cases is shown in Figure 4.3.

Finally, CenterPoint with HRFiS is trained using two different tuning configurations. One where for all classes, only objects that are originally within what we define as long range are sampled (HRFiSv1). This is done in order to evaluate whether our downsampling methods can possibly have a negative impact due to how they affect the point distribution within objects. The other configuration (HRFiSv2) only applies this to classes that are more frequent in the original nuScenes data; Cars, Trucks, Buses, and Pedestrians. It also adds local xy-flip of sampled objects which is not present in HRFiSv1. We propose that using HRFiSv2 for more uncommon classes like Construction Vehicles limits the available objects to be sampled. Filtering on the interval 2/3 to 3/3 of max range for Construction Vehicle results in only being able to sample about 42% of all the Construction Vehicles originally available in the database, which could potentially have a big impact on training data diversity for this class. This could in turn have a big negative impact on the final results, a hypothesis that is further confirmed while later looking at the results of

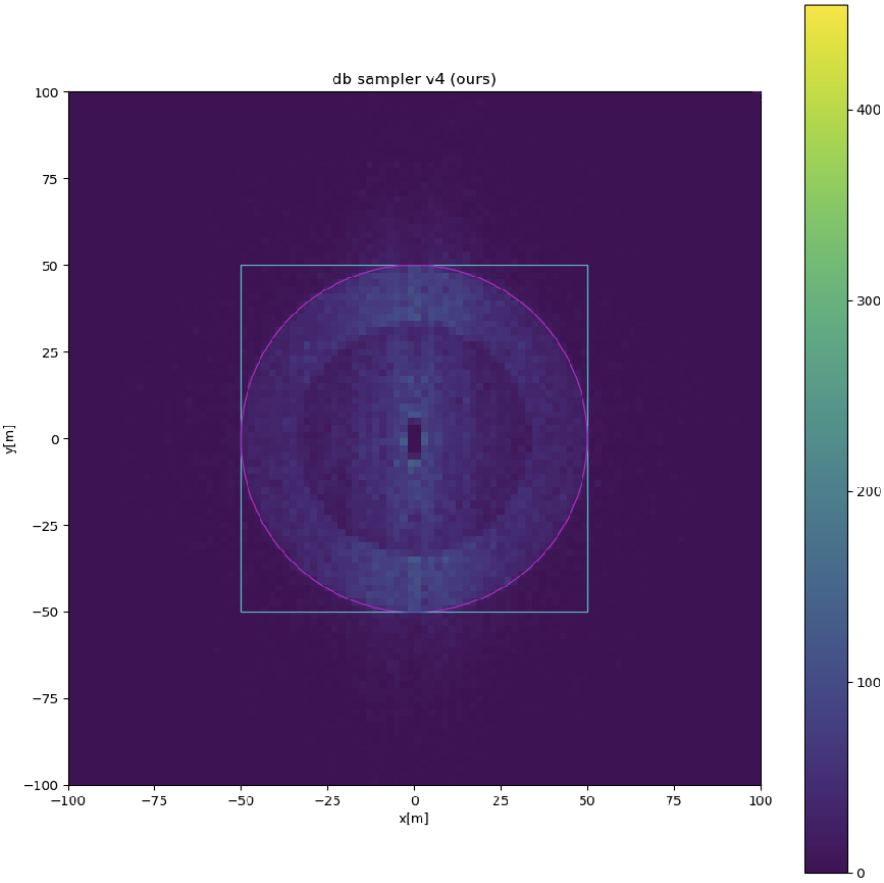


Figure 4.3: 2D heatmap for Class car. RRD with downsampling rate 0.5.

HRFiSv1. As for Barrier and Traffic cone, these are not as poorly represented as Construction Vehicle but are very locationally biased towards short range and are hence also excluded when filtering by range in HRFiSv2.

4.1.3 PointPillars

We also perform experiments for some of the configurations on PointPillars in order to see if the results are consistent across different models. The configurations used for training on PointPillars are identical to their counterparts in section 4.1.2 and the following are used:

- Without GTDS
- With Baseline GTDS
- With RFD using Random downsampling
- With HRFiSv2

4.2 Evaluation on nuScenes validation set

Due to the fact that evaluation on the nuScenes test set requires a formal model submission, we perform all evaluations of our methods exclusively on the validation set. This section presents the results of our experiments on performance variance between training runs and experiments on the original implementations of GTDS and Fade.

4.2.1 Estimating variation limits between training runs

In order to provide some context in regards of performance differences between different training runs using the exact same model configuration, we start by performing the experiments described in section 3.6. The numerical results are provided in Table 4.2. For class-specific AP metrics along with mAP and NDS, the biggest difference per metric *across both model configurations* is presented. This is meant to represent the biggest difference we can expect to encounter when training with our different configurations. It is not done for the other class-specific TP-metrics since we find that there is big variance in a lot of them and they each contribute relatively little to the NDS. For a more detailed Table of these results, see Appendix B.

For the version of HRFiS used in these experiments (HRFiSv2), we apply hard range filtering to the classes Car, Truck, Bus and Trailer with a range from sensor interval of $[20, 50]$ in meters, along with Pedestrian on the interval $[15, 40]$. All sampled objects are also filtered by a minimum amount of points inside the bounding box of 5. Class acronyms used in the tables include C.V. for Construction Vehicle, Ped. for Pedestrian, Mot. for Motorcycle and T.C. for Traffic Cone.

These results should be kept in mind while reading the rest of this chapter, and we recommend going back to them when examining results that are similar between different methods.

4.2.2 Baseline GTDS and Fade augmentation experiments

In order to evaluate the performance gains from utilizing the Fade augmentation, we apply training with Fade to all configurations of both the CenterPoint and PointPillars Baseline models and compare them to their non-Faded counterparts. We also train models entirely without GTDS and compare these to the baseline models with and without Fade. The results of these experiments are presented in Tables 4.3 and 4.4.

Metric	MED
Car AP	0.002
Truck AP	0.013
Bus AP	0.032
Trailer AP	0.013
C.V. AP	0.016
Ped. AP	0.011
Mot. AP	0.014
Bicycle AP	0.041
T.C. AP	0.020
Barrier AP	0.022
mATE	0.008
mASE	0.002
mAOE	0.046
mAVE	0.027
mAAE	0.008
mAP	0.007
NDS	0.004

Table 4.2: Maximum Expected Difference (MED) for all class-specific APs, all mean metrics and NDS.

Table 4.3: Full range results for CenterPoint without GTDS and CenterPoint Baseline with and without Fade. Green numbers represent better performance scores compared to No GTDS while red ones represent worse performance. Acronyms include C.V. for Construction Vehicle, Ped. for Pedestrian, Mot. for Motorcycle and T.C. for Traffic Cone.

Class	**CenterPoint AP (Full range)**		
	No GTDS	GTDS	GTDS Fade
Car	0.817	0.840	0.839
Truck	0.479	0.500	0.513
Bus	0.579	0.618	0.640
Trailer	0.333	0.329	0.366
C.V.	0.142	0.127	0.203
Ped.	0.753	0.778	0.777
Mot.	0.499	0.426	0.585
Bicycle	0.199	0.157	0.305
T.C.	0.516	0.539	0.552
Barrier	0.552	0.608	0.598
mAP	0.487	0.492	0.538
NDS	0.597	0.597	0.628

Table 4.4: Full range results for PointPillars without GTDS and PointPillars Baseline with and without Fade.

Class	**PointPillars AP (Full range)**		
	No GTDS	GTDS	GTDS Fade
Car	0.806	0.801	0.810
Truck	0.370	0.372	0.413
Bus	0.447	0.468	0.497
Trailer	0.280	0.257	0.307
C.V.	0.056	0.033	0.098
Ped.	0.735	0.717	0.738
Mot.	0.391	0.215	0.399
Bicycle	0.122	0.023	0.129
T.C.	0.340	0.236	0.345
Barrier	0.530	0.524	0.570
mAP	0.408	0.365	0.431
NDS	0.540	0.525	0.563

From the results presented in Table 4.4 it can be seen that when used for CenterPoint, GTDS can have a negative impact on the AP for some classes even if the overall performance in terms of mAP and NDS increases. The affected classes are less frequent ones that are sampled in larger quantities compared to the more common classes. The same trend can be observed when looking at the PointPillars results in Table 4.4. Although even more of the classes are negatively impacted here, it is still most apparent for the more unusual ones.

The main reason for this seems to be connected to the number of predictions generated for these classes. Using GTDS without Fade teaches the model that the uncommon classes are more common than they really are in the dataset, which increases the amount of FP predictions. Comparing Tables 4.3 and 4.5 it can be noticed that for the classes where AP is reduced the most when applying GTDS, the number of FPs for GTDS is significantly larger compared to *No GTDS* and *GTDS Fade* which ultimately affects the AP for these classes. A qualitative example of the discrepancy in predicted objects is shown in Figures 4.4 and 4.5. Only prediction and ground truths for the class Bicycle are shown since it is the class with the highest proportional difference between FPs and TPs. Notice that while the model trained without Fade manages to predict bicycles for both available ground truths, it also produces many more false positives. The one trained with Fade (Figure 4.5) misses one of the bicycles but has no false positives at all.
It is also interesting to look at how Fade improves long-range results, which is presented in Table 4.7. While looking at the results for HRFiSv1 in this table, we can conclude that even when the model that the Fade training is resumed from (before Fade) produces significantly worse results than its counterparts, the Fade augmentation is able to make up for this shortcoming fairly effectively, closing the gap between good and bad configurations.

The results for PointPillars in Table 4.4 show an overall loss of performance, in terms of both mAP and NDS, from using GTDS. This is not consistent with the results presented in [18] when GTDS was applied to the KITTI dataset. Since the experiments were carried out using a GTDS version tuned as in [35], this shows that the results can be connected to tuning of the number of sampled objects in GTDS when training Pointpillars on nuScenes. For comparison, the GTDS configuration of [18] on the KITTI dataset used a sampler tuning of 15, 0 and 8 for Car, Pedestrian and Cyclist respectively. This is a totally different ratio between how commonly each of the classes are already present in the dataset.

In terms of the ratio between FPs and TPs, PointPillars is much worse than CenterPoint. These results are presented in Table 4.6. This seems to be mostly due to the sheer amount of FPs produced by the model but we once again see a pattern of reducing FPs drastically by applying Fade.

Table 4.5: Comparison between the number of FP and FN for CenterPoint evaluated on the nuScenes validation set. The matching is done on the shortest distance threshold $d = 0.5$ m. Red values represent the worst value for each specific class and metric and green represent the best value

CenterPoint	**FP (Full range)**									
	Car	Truck	Bus	Trailer	C.V	Ped.	Mot.	Bicycle	T.C	Barrier
No GTDS	101,696	22,654	1,535	2,824	3,264	96,231	1,599	2,132	51,297	26,849
Baseline GTDS	84,530	36,453	5,962	23,798	39,274	94,380	43,562	125,576	85,470	50,105
Baseline GTDS + Fade	81,530	24,419	1,688	3,559	4,515	86,692	3,995	7,691	49,468	23,976
	TP (Full range)									
No GTDS	42,869	5,334	1,095	684	228	21,250	1,041	590	8,434	10,621
Baseline GTDS	43,967	5,613	1,197	821	362	21,584	1,242	1,264	8,942	11,883
Baseline GTDS + Fade	43,813	5,624	1,237	764	338	21,572	1,228	991	8,664	11,547
	# GT objects available									
	53,043	9,260	2,062	2,331	1,513	23,500	1,855	1,824	10,570	15,903

Table 4.6: Comparison between the number of FP and FN for PointPillars evaluated the nuScenes validation set. The matching is done on the shortest distance threshold $d = 0.5$ m.

PointPillars	**FP (Full range)**									
	Car	Truck	Bus	Trailer	C.V	Ped.	Mot.	Bicycle	T.C	Barrier
No GTDS	196,106	108,479	20,608	44,388	43,512	174,021	30,077	47,901	123,880	83,756
Baseline GTDS	224,010	167,262	54,493	185,698	207,479	243,038	207,910	272,177	201,274	142,665
Baseline GTDS + Fade	194,716	109,216	17,344	40,608	36,626	196,908	35,193	59,612	138,989	83,219
	TP (Full range)									
No GTDS	41,916	4,819	943	497	295	21,722	1,372	1,084	8,669	10,924
Baseline GTDS	41,622	5,060	1,010	580	397	22,055	1,428	1,391	8,840	11,303
Baseline GTDS + Fade	42,137	5,220	1,016	639	370	22,038	1,374	1,263	9,040	11,351
	# GT objects available									
	53,043	9,260	2,062	2,331	1,513	23,500	1,855	1,824	10,570	15,903

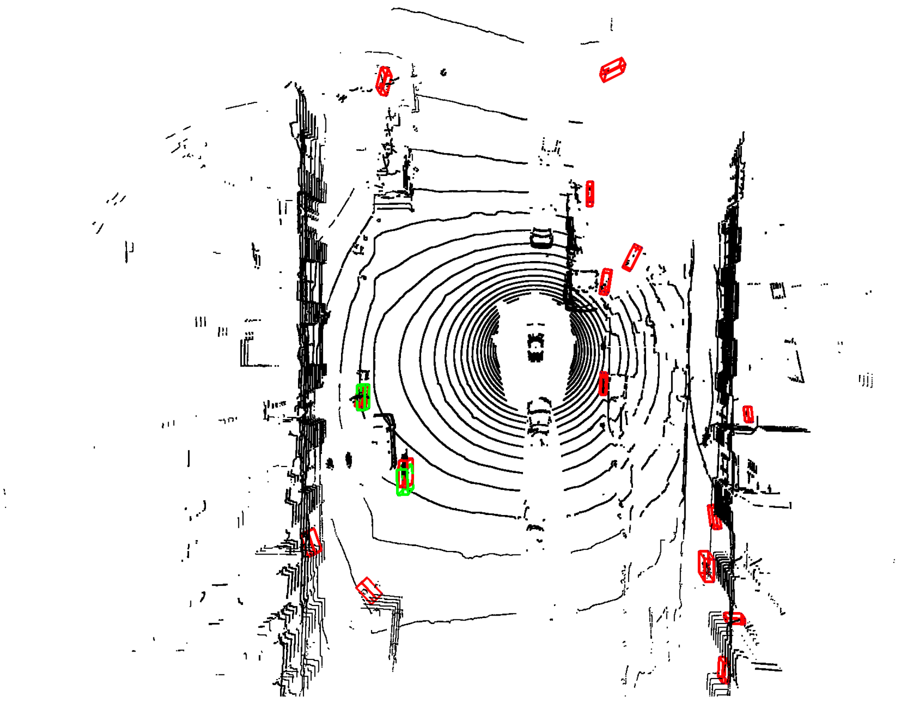


Figure 4.4: Bicycle GTs (green) and predictions (red) generated by CenterPoint Baseline without Fade.

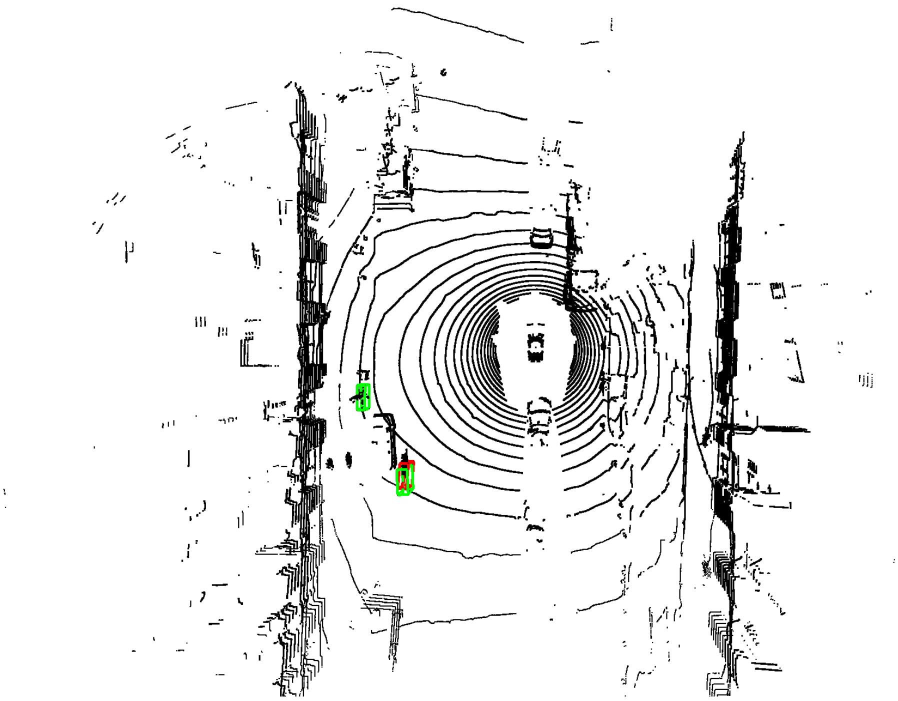


Figure 4.5: Bicycle GTs (green) and predictions (red) generated by CenterPoint Baseline + Fade.

The main takeaway from these experiments is that GTDS can turn out to have a quite large negative impact on the detection results before applying Fade, especially for classes that are more uncommon in the unmodified data and are hence sampled more often. These are however also the ones that benefit the most from applying Fade, compared to their no GTDS-counterparts. The results suggest that no matter which variant of GTDS is used, it must be done with careful tuning and the results of doing so may vary a lot between different models, even on the same dataset.

4.3 Modifed GTDS long-range detection results

As stated in section 3.3, we also find it interesting to report results for different ranges. As the modifications of GTDS presented in this thesis aim to focus on long-range detections, only metrics for what we previously defined as long range will be presented here. Once again, all results are evaluated with the nuScenes validation split as a base.

4.3.1 CenterPoint results

In order to closely examine the results of CenterPoint on longer ranges, we present tables of all nuScenes mean-metrics along with tables of class-specific AP for all available classes and configurations.

Table 4.7: CenterPoint long-range results for different versions of the GTDS with and without Fade. The best results for each metric are marked with green text while the worst one are marked with red. Methods marked with * are heavily tuned towards long range.

Method	**Metric (long range)**						
	mAP	mATE	mASE	mAOE	mAVE	mAAE	NDS
No GTDS	0.334	0.418	**0.289**	0.454	0.372	**0.161**	0.498
Baseline GTDS	0.328	0.420	0.286	0.468	0.372	0.179	0.491
Baseline GTDS + Fade	**0.373**	0.417	0.281	**0.417**	**0.348**	0.168	**0.523**
RFD Random	0.332	0.415	0.284	0.471	0.399	0.191	0.490
RFD Random + Fade	0.370	0.414	0.279	0.426	0.366	0.182	0.519
RFD FPS	0.335	**0.423**	0.285	0.458	0.412	0.183	0.492
RFD FPS + Fade	0.367	0.405	**0.278**	0.422	0.379	0.177	0.518
RRDv1	0.317	0.418	0.287	0.469	0.414	0.186	0.481
RRDv1 + Fade	0.366	0.418	0.282	0.416	0.385	0.172	0.516
RRDv2*	0.320	0.410	0.283	0.513	0.416	0.184	0.480
RRDv2 + Fade*	0.370	**0.404**	0.281	0.482	0.389	0.177	0.511
HRFiSv1*	**0.287**	0.422	**0.289**	**0.508**	0.393	**0.205**	**0.462**
HRFiSv1 + Fade*	0.357	0.414	0.286	0.447	0.363	0.186	0.509
HRFiSv2	0.324	0.417	0.285	0.471	**0.419**	0.190	0.484
HRFiSv2 + Fade	0.368	**0.404**	0.279	0.432	0.383	0.176	0.517

Table 4.8: CenterPoint class-specific long-range results for different versions of the GTDS with and without Fade. The best results for each class are marked with green text while the worst one are marked with red. Methods marked with * are heavily tuned towards long range. For mAP/NDS, see Table 4.7.

Method	**Class AP (long range)**									
	Car	Truck	Bus	Trailer	C.V	Ped.	Mot.	Bicycle	T.C	Barrier
No GTDS	**0.580**	0.326	0.375	0.209	0.033	**0.653**	0.283	0.108	0.306	0.475
Baseline GTDS	0.618	0.322	0.401	0.212	0.017	0.689	0.150	0.011	0.340	0.517
Baseline GTDS +Fade	0.617	**0.353**	0.441	0.238	0.074	0.688	0.348	0.096	**0.353**	0.520
RFD Random	**0.625**	0.327	0.394	0.225	0.022	0.684	0.193	0.012	0.318	**0.523**
RFD Random + Fade	0.620	0.337	0.425	0.245	0.063	0.692	0.339	**0.114**	0.345	**0.523**
RFD FPS	0.620	0.339	0.393	0.242	0.044	**0.693**	0.172	0.018	0.327	0.507
RFD FPS + Fade	0.615	0.339	0.429	0.266	0.049	**0.693**	**0.350**	0.092	0.345	0.496
RRDv1	0.612	0.308	0.381	0.219	0.013	0.679	0.125	0.007	0.316	0.506
RRDv1 + Fade	0.612	0.334	0.408	0.269	0.071	0.684	0.341	0.093	0.352	0.499
RRDv2*	0.622	0.325	0.408	0.236	0.009	0.684	0.130	0.008	0.311	**0.471**
RRDv2 + Fade*	0.619	**0.353**	**0.445**	**0.275**	0.050	0.689	0.332	0.092	0.344	0.497
HRFiSv1*	0.601	**0.278**	**0.334**	**0.174**	**0.002**	0.670	**0.059**	**0.002**	**0.277**	0.476
HRFiSv1 + Fade*	0.609	0.334	0.394	0.227	0.031	0.683	0.317	0.112	0.351	0.509
HRFiSv2	0.619	0.309	0.393	0.201	0.024	0.684	0.136	0.020	0.347	0.510
HRFiSv2 + Fade	0.619	0.338	0.424	0.252	**0.078**	**0.693**	0.315	0.113	0.338	0.514

4.3.2 CenterPoint discussion

One of the most interesting takeaways when looking at table 4.7 is that HRFiSv1 results in the worst performance for four out of six metrics and also NDS which is a weighted average of the other ones. This method was trained with the sole purpose of trying to increase long-range detections by biasing towards them when utilizing GTDS. The idea was that this might result in better performance than RRDv2, due to the fact that it does not alter the point distributions within objects since no moving of sampled objects or downsampling is applied. Ultimately, HRFiSv1 still fails at producing any performance gains for long range, which we find interesting to discuss.

A possible reason for why this happens is that we aggressively limit the number of different objects to be sampled, as described for Construction Vehicle in section 4.1.2. This is increasingly obvious when looking at the class-specific long-range results in Table 4.8. The most uncommon classes (C.V., Motorcycle, Bicycle) exhibit the most significant drop in performance from using this method. Although the Trailer and Bus classes are almost as uncommon, a larger portion of their objects that are available for sampling are still present when filtering by long range, since they appear more frequently at longer ranges. We also argue that these are easier for the network to learn to detect, both based on training results and their general geometry and size. The fact that HRFiSv1 limits the number of objects that are available for sampling drastically is more or less true for all of the classes and could likely have harmful effects on the final performance by greatly limiting data diversity. Also, since there is no moving of sampled objects done online for this configuration, the models could end up with even more bias in regards of where objects are predicted within each point cloud sample. Since we saw a clear local bias for some classes already in the unfiltered 2D heatmaps 3.4b, we can expect this locational bias to be even more prominent when aggressively filtering out a big portion of the objects available for sampling. An interesting way to tackle this problem would be to make more use of the local augmentations described in section 2.7.2. This is what we would proposedly do if we had more time to further evaluate this approach. Both in terms of mAP, which is the main contributor to the NDS, and actual NDS, this configuration is the worst one while significantly exceeding all of the expected differences between training runs presented in section 4.2.1.

Continuing on the trail of trying to force long-range bias for the models, RRDv2 exhibits better performance than HRFiSv1. Still, in terms of mAP and NDS, it performs slightly worse than the baseline for long-range detections, both with and without Fade. This leads us to believe that there might be some issues with the techniques we use to downsample objects. If the downsampled objects did in fact contain good point distributions that conform to how objects look at longer ranges in the real data, it is hard to see how RRDv2 would not produce better results at longer ranges than the baseline even if this would come at the cost of worse results for short- and medium-range. A solution like this could be interesting in a scenario where we run two networks in parallel on a vehicle, one that is biased towards long range and one that is biased towards short and medium. However, this is not a result we managed to achieve in this thesis and we propose to look further into this in the future.

The configuration that produces results that are closest to the baseline for long-range detections is RFD Random with Fade. Although it is slightly worse than the Baseline with Fade, these values lie within the expected differences between model trainings both for mAP and NDS. It is reasonable to consider these two methods equal until they have been trained more times and an average of these training runs can be made to get a more certain result. These results are possibly an effect of RFD Random being a bit more conservative when it comes to moving objects further away. Since it never moves a sampled object more than 2.2 times further away from the sensor than where it was sampled, the choice of a subpar downsampling method does not have the same risk of producing a bad point distribution within an object. An object that is only moved twice as far away will still conform better to the general point distribution in terms of the number of scan lines hitting the object than one that is e.g., moved 10 times further away than where it was originally found (which is possible in RRD). Hence, we propose that if using another technique to downsample objects, RRD could possibly produce better results than other methods.

4.3.3 PointPillars results

The long range results for PointPillars will be presented in this section. Since the experiments on Pointpillars was not as extensive as those for CenterPoint, only some versions of the modified GTDS sampler are represented here. The overall metrics can be seen in Table 4.9.

Table 4.9: PointPillars long-range results for different versions of the GTDS with and without Fade. The best results for each metric are marked with green text while the worst ones are marked with red.

Method	Metric (long range)						
	mAP	mATE	mASE	mAOE	mAVE	mAAE	NDS
No GTDS	0.258	**0.511**	**0.301**	0.530	0.371	**0.149**	0.443
Baseline GTDS	**0.223**	0.488	0.297	0.541	0.377	0.183	0.423
Baseline GTDS + Fade	0.266	**0.472**	**0.289**	**0.514**	0.369	0.165	**0.452**
RFD Random	0.232	0.484	0.295	0.565	**0.412**	0.184	**0.422**
RFD Random + Fade	**0.274**	0.477	0.290	**0.567**	0.373	0.162	0.450
HRFiSv2	0.225	0.477	0.300	0.543	0.367	**0.189**	0.425
HRFiSv2 + Fade	0.267	0.482	0.293	0.518	**0.360**	0.164	**0.452**

4.3.4 PointPillars discussion

One interesting takeaway from Table 4.9 is the similarity of results between Baseline GTDS + Fade and HRFiSv2 + Fade which suggest that the closely sampled objects do not affect the long-range performance of PointPillars as it does for CenterPoint. The loss of mAP for Baseline GTDS mentioned in 4.2.2 can also be seen for the long-range results. This is not surprising since no additional techniques are used to improve long-range performance. RFD Random + Fade and Baseline + Fade are performing similarly in relation to each other as they do for CenterPoint. The same reasoning as for the CenterPoint versions applies in this case. It is interesting that the main contributor to the lower NDS for RFD Random + Fade is the mAOE. If the difference between training runs is similar to those of CenterPoint, the mAOE lies within the expected difference but the mAP does not. This could indicate that a better result from RFD Random can be expected if more trainings were made and an average of these runs was used instead.

4.3.5 Effect of Random sampling vs. FPS

When examining the differences between using FPS and Random choice to downsample objects, the long-range results are quite similar. RFD FPS performs slightly better than RFD Random without Fade, while when adding Fade to both RFD Random pulls ahead slightly. For all of the metrics, the difference between these two methods are within or very close to the usual training variance differences presented in Tables B.1 and B.2. This is one of the main reasons we chose not to proceed with using FPS for downsampling objects in RRDv1 and RRDv2.

Another perhaps more important reason is that using FPS for downsampling increase the time it takes to train a model. When training RFD with the configuration we used for these experiments and Random downsampling, the training times on four NVIDIA A100 GPUs were about 36 hours. When using FPS for downsampling instead, the training time ended up being about 62 hours. This corresponds to an increased training time of more than 70% while not even accounting for the time used to validate the models between epochs (this is done between every epoch for these training runs), which is unaffected by the choice of downsampling method.

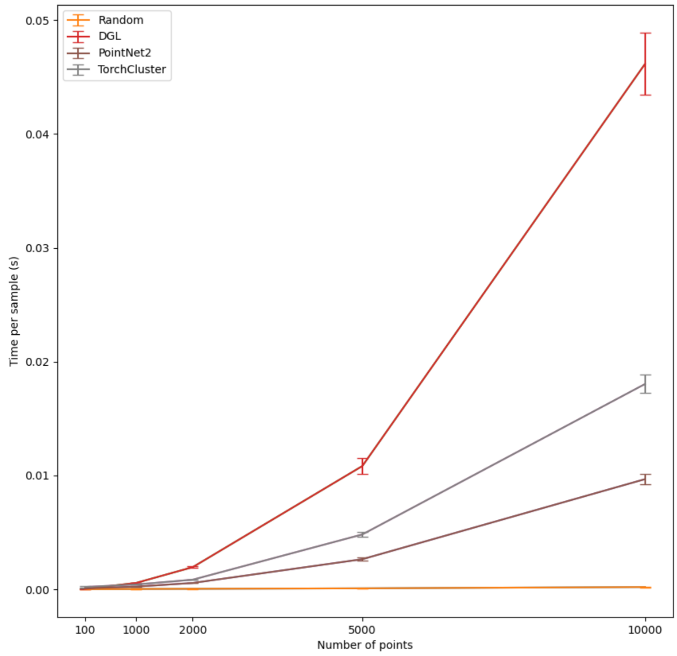


Figure 4.6: Comparison of FPS implementations for numbers of points to sample [100, 1000, 2000, 10000]. The maximum amount of points (10000) is roughly the same as the maximum amount of points present in a Car ground truth as per Figure 3.3. Data transfer time between CPU and GPU tensors is taken into account.

The above experiment is however done with an implementation of FPS that runs entirely on a CPU. As FPS is an algorithm that should in theory benefit a lot from extensive multi-threading on a GPU, we perform some minor experiments on different implementations. The implementation used for training is the one provided by Deep Graph Library (DGL) [32] and we compare it to one provided by PyTorch Cluster [10] and one by PointNet2 [33]. Both of the latter utilize cuda code to implement the algorithms to run on a GPU and are run the experiments are run on a NVIDIA GTX 1650Ti GPU. Since this is a far inferior GPU compared to the A100s used while training, we can at the very least expect the results of the GPU-based variants to be on par with these. The results are obtained using Monte Carlo Simulations on random sets of LiDAR points and are presented in Figure 4.6. They show that using the PointNet2 implementation of FPS can reduce the time spent calculating the set of points for FPS by up to 85%.

In conclusion, the benefits of using FPS to downsample objects instead of Random downsamplings are close to none, and we proceed to not further evaluate this technique for RRDv1 and RRDv2. This is, judging from the overall results in Table 4.7, possibly a side effect of both FPS and Random sampling both being subpar choices of downsampling methods.

4.4 Results for all range intervals and comparison with long-range results

Finally, with long-range results already in mind, we present the results for short- and medium-range as well. Even though the work in this thesis is focused on long-range detections, our work might have unintended effects on the detections for other ranges. Hence, class-specific metrics for CenterPoint on all- and medium- and short-range are presented in Tables 4.10, 4.11 and 4.12 respectively.
Regarding the more common classes (Car, Truck, Pedestrian), the results are very similar across all methods except No GTDS for the full detection range. This confirms our previous conclusion about the different variations of GTDS not having a very big impact on these classes. However, there is one outlier; RFD Random's performance *without Fade* on the Car class. The all-range AP-increase (Table 4.10) of 0.003 compared to the second best model in the category is slightly higher than the approximated maximum difference presented in section 4.2.1. The ranges that seem to contribute most to these results are also long and medium. Keep in mind that for this configuration, 90% of the sampled cars were chosen for downsampling with a downsampling factor ranging from 1.7 to 2.2 as presented in Table 4.1. The overall quality of sampled cars for this configuration was also fairly high due to the minimum point-filtering also presented in the same table. This leads us to believe that for classes that are well represented and contain diverse enough data, more aggressive minimum point-filtering along with random downsampling can result in a positive impact on final performance. A better downsampling technique would likely have an even better effect though.

The method that performs the worst in terms of all-range-mAP is once again HRFiSv1. An interesting thing to note is that most of this performance degradation seems to originate from its long-range predictions. For many of the classes, it does not perform significantly worse when it comes to short- and medium-range. Our intuition about why this happens is that we repeatedly sample from a much less diverse subset of the data (since this method samples objects at long range exclusively, without any moving and downsampling) and hence introduces a great amount of locational bias for the longer ranges. This would however have to be confirmed by extensive examination of specific frames, in order to see if it often predicts objects in the same places for long range i.e., having large locational bias.

Regarding the other methods, they usually perform fairly similarly when it comes to short- and medium-range. A clear trend that can be seen from these tables is that even when biasing more towards longer ranges, GTDS results in improvements AP-wise for most classes compared to using No GTDS at all. This is true both before and after applying Fade.

Finally, Table 4.13 show that in terms of mAP, all models utilizing some type of GTDS except RRDv2 and HRFiSv1 exhibit similar performance when we average over all three range intervals. The degradation in performance for RRDv2 seems to mostly be due to low performance on close ranges, while the degradation for the

Table 4.10: CenterPoint class-specific **all-range** results for different versions of the GTDS with and without Fade. The best results for each class are marked with green text while the worst ones are marked with red. For overall AP/NDS, see Table 4.13.

Method	**Class AP (all range)**									
	Car	Truck	Bus	Trailer	C.V	Ped.	Mot.	Bicycle	T.C	Barrier
No GTDS	**0.817**	0.479	0.579	0.333	0.142	**0.753**	0.499	0.199	0.516	**0.552**
Baseline GTDS	0.840	0.500	0.618	0.329	0.127	0.778	0.426	0.157	0.539	0.608
Baseline GTDS + Fade	0.839	0.513	**0.640**	0.366	**0.203**	0.777	0.585	0.305	0.552	0.598
RFD Random	**0.843**	0.507	0.633	0.328	0.139	0.778	0.466	0.154	0.538	**0.615**
RFD Random + Fade	0.839	0.514	**0.640**	0.336	0.195	0.785	**0.586**	0.299	0.554	0.612
RFD FPS	0.840	0.505	0.620	0.348	0.126	0.782	0.444	0.134	0.550	0.595
RFD FPS + Fade	0.838	0.511	**0.640**	0.376	0.180	0.785	**0.586**	0.280	0.557	0.592
RRDv1	0.838	0.493	0.607	0.343	0.130	0.779	0.425	0.121	0.551	0.604
RRDv1 + Fade	0.836	0.503	0.627	0.383	0.201	0.779	0.581	0.281	**0.561**	0.586
RRDv2*	0.838	0.495	0.612	0.341	0.106	0.777	0.395	0.112	0.528	0.578
RRDv2 + Fade*	0.838	0.516	0.633	**0.391**	0.166	0.784	0.563	0.258	0.539	0.587
HRFiSv1*	0.833	**0.474**	**0.566**	0.292	**0.049**	0.768	**0.317**	**0.085**	**0.512**	0.577
HRFiSv1 + Fade*	0.836	**0.521**	0.611	0.343	0.143	0.775	0.538	0.260	0.551	0.590
HRFiSv2	0.839	0.483	0.609	**0.306**	0.120	0.780	0.431	0.158	0.537	0.597
HRFiSv2 + Fade	0.838	0.508	0.636	0.358	0.196	**0.787**	0.576	**0.319**	0.543	0.599

latter is more related to its medium- and long-range performance.

Table 4.11: CenterPoint class-specific **medium-range** results for different versions of the GTDS with and without Fade. The best results for each class are marked with green text while the worst one are marked with red. For overall AP/NDS, see Table 4.13. Methods marked with * were heavily tuned towards long-range bias

Method	Class AP (medium range)									
	Car	Truck	Bus	Trailer	C.V	Ped.	Mot.	Bicycle	T.C	Barrier
No GTDS	**0.828**	0.508	**0.649**	0.362	0.221	0.748	0.520	0.196	**0.510**	**0.492**
Baseline GTDS	**0.856**	0.526	0.696	0.368	0.216	0.762	**0.435**	0.180	0.541	0.565
Baseline GTDS + Fade	0.855	0.520	0.699	0.401	**0.277**	**0.760**	0.594	0.306	0.554	0.542
RFD Random	**0.856**	0.536	**0.731**	0.349	0.214	0.765	0.477	0.160	0.545	**0.572**
RFD Random + Fade	0.853	0.534	0.719	0.339	0.255	0.770	0.602	0.303	**0.558**	0.567
RFD FPS	0.854	0.532	0.708	0.372	0.152	0.771	0.448	0.146	0.547	0.545
RFD FPS + Fade	0.850	0.530	0.715	0.401	0.234	0.777	0.583	0.276	0.542	0.549
RRDv1	0.855	0.523	0.696	0.386	0.219	0.769	0.463	**0.125**	0.555	0.562
RRDv1 + Fade	0.851	0.512	0.707	0.402	0.270	0.762	**0.605**	0.289	**0.558**	0.543
RRDv2*	0.847	0.516	0.678	0.383	0.221	0.767	0.456	0.149	0.544	0.546
RRDv2 + Fade*	0.849	0.530	0.679	**0.428**	0.233	0.771	0.578	0.253	0.537	0.539
HRFiSv1*	0.848	0.526	0.693	0.340	**0.154**	0.762	0.442	0.209	0.517	0.534
HRFiSv1 + Fade*	0.850	**0.557**	0.695	0.356	0.206	0.767	0.533	0.270	0.536	0.545
HRFiSv2	0.851	**0.503**	0.696	**0.334**	0.172	0.777	0.453	0.196	0.533	0.550
HRFiSv2 + Fade	0.850	0.524	0.706	0.377	0.225	**0.778**	0.595	**0.342**	0.545	0.549

Table 4.12: CenterPoint class-specific **short-range** results for different versions of the GTDS with and without Fade. The best results for each class are marked with green text while the worst one are marked with red. For overall AP/NDS, see Table 4.13. Methods marked with * were heavily tuned towards long-range bias.

Method	Class AP (short range)									
	Car	Truck	Bus	Trailer	C.V	Ped.	Mot.	Bicycle	T.C	Barrier
No GTDS	**0.957**	**0.672**	**0.782**	0.627	**0.229**	**0.844**	0.703	0.285	0.651	**0.750**
Baseline GTDS	0.960	0.723	0.813	0.573	0.284	0.866	0.762	0.275	0.648	0.797
Baseline GTDS + Fade	0.960	0.728	0.861	**0.719**	0.348	0.868	0.824	**0.504**	0.663	0.799
RFD Random	0.960	0.732	0.833	0.614	0.298	0.869	0.777	0.324	0.658	0.793
RFD Random + Fade	0.959	**0.753**	0.857	0.652	0.359	**0.878**	0.832	0.453	0.677	0.793
RFD FPS	**0.961**	0.706	0.831	0.629	0.261	0.867	0.767	0.259	0.684	0.798
RFD FPS + Fade	0.960	0.736	**0.866**	0.665	0.345	0.870	0.827	0.472	**0.695**	0.796
RRDv1	0.959	0.721	0.818	0.572	0.334	0.872	0.768	0.282	0.686	0.792
RRDv1 + Fade	0.959	0.747	0.849	0.689	0.354	0.874	0.800	0.459	0.688	0.772
RRDv2*	0.958	0.703	0.847	**0.544**	0.256	0.861	**0.671**	**0.244**	**0.624**	0.780
RRDv2 + Fade*	0.958	0.723	0.853	0.581	0.260	0.872	0.797	0.423	0.650	0.784
HRFiSv1*	**0.961**	0.711	0.838	0.602	0.259	0.854	0.735	0.292	0.672	0.789
HRFiSv1 + Fade*	0.960	0.731	0.855	0.687	0.248	0.859	0.780	0.365	0.680	0.784
HRFiSv2	0.960	0.725	0.813	0.576	0.262	0.859	0.769	0.254	0.645	0.801
HRFiSv2 + Fade	0.959	**0.753**	0.861	0.656	**0.363**	0.871	**0.839**	0.476	0.658	**0.809**

Method	Range interval mAP			
	short	medium	long	all
No GTDS	0.650	0.503	0.335	0.487
Baseline GTDS	0.670	0.515	0.328	0.492
Baseline GTDS + Fade	**0.727**	**0.551**	**0.373**	**0.538**
RFD Random	0.686	0.520	0.332	0.500
RFD Random + Fade	0.721	0.550	0.370	0.536
RFD FPS	0.676	0.507	0.335	0.494
RFD FPS + Fade	0.723	0.546	0.367	0.535
RRDv1	0.680	0.515	0.316	0.489
RRDv1 + Fade	0.719	0.550	0.366	0.534
RRDv2*	**0.649**	0.511	0.320	0.478
RRDv2 + Fade*	0.690	0.540	0.370	0.527
HRFiSv1*	0.671	**0.502**	**0.287**	**0.447**
HRFiSv1 + Fade*	0.695	0.531	0.357	0.517
HRFiSv2	0.666	0.507	0.324	0.486
HRFiSv2 + Fade	0.724	0.549	0.368	0.536

Table 4.13: CenterPoint mAP for all range intervals. Methods marked with * were heavily tuned towards long-range bias.

5
Conclusion

Through extensive experiments on two different models and the NuScenes dataset, this thesis shows the importance of utilizing GTDS carefully and correctly. This foremost includes the need for tuning the number of sampled objects for the specific application and dataset.

Our experiments using GTDS and Fade suggest that while GTDS was originally designed to smoothen out the discrepancy between background and foreground points along with tackling class imbalance, it can potentially hurt the performance of the more uncommon classes a lot if not tuned correctly. The Fade augmentation is partially able to make up for the shortcomings of subpar GTDS tuning but can enhance results even further if the original GTDS tuning is good.

Although our proposed methods ultimately fail when it comes to increasing long-range detection performance, a detailed discussion about why this is the case is provided along with recommendations about what we think should be done differently. We also show that long-range accuracy seems to be more connected to the quality of sampled objects and diversity in the data. This leads us to believe that there is still untapped potential when it comes to utilizing GTDS as effectively as possible.

5.1 Proposed further work

As a consequence of the fact that some of our GTDS configurations that were tuned to be biased towards longer ranges had a negative impact on the long-range detection performance, this leads us to believe that the choice of downsampling method might be a problem. Hence, our first proposal when it comes to future work is to try out other downsampling methods. A good starting point for this would likely be an adaption of [16] towards the nuScenes data format. Even though we've previously mentioned that this might be tricky due to noise originating from sweep concatenation used in nuScenes, it might be possible to do sufficiently well by e.g., dividing the scan angles into fixed bins. It would also be interesting to treat the relation between the number of points in an object and its distance from the sensor by modeling occlusion in objects instead choosing the points to sample cubically proportional to the factor S by which we move objects further away. The reason for this is that the number of points should actually be quadratically proportional to the range of the detections.

One subject not touched upon regarding downsampling and translation of GT objects is that of point intensity. It could be that the intensity of the downsampled objects points need to be adjusted accordingly to properly represent objects further away.

Lastly, since models using voxel-based backbones were not evaluated in this thesis it could be of value to investigate the effect of our techniques on these. The reason for this is that more spatial information is preserved by using a voxelized point cloud rather than pillarized one.

5.2 Ethical aspects

When conducting new research, it is important to take all possible use cases of the research into account. In the bigger picture, the question is if the rapid advances in object detection will be of positive significance or if it also comes with new hazards to take into account. Since most of the object detection research is done to accelerate the realization of AD, some aspects need to be taken into consideration. Can you trust an autonomous vehicle to properly handle all edge cases that the average human would be able to in a traffic situation, and who is to blame if it fails? There are many of these different ethical and moral dilemmas that might come to mind when dealing with AD. One common dilemma that usually surfaces in this discussion is the *Trolley Problem*, where a bystander has the opportunity to divert a runaway train heading towards a group of people, but in doing so will make the train head for a single person. What is the morally correct thing to do in this case? Such questions are difficult even for human experts to answer, and can we trust a machine or an algorithm to make a good decision? No matter the outcome of such a dilemma, one question still remains, and that is who should ultimately be held responsible for the decision made.

Better and more precise object detection could potentially be implemented for military usage in autonomous weapons. One might argue that more precise object detection could reduce the risk of collateral damage and civilian casualties, but it could as likely be used to target power or healthcare infrastructure and people with a specific ethnical background. Weaponizing object detection and deep learning algorithms is something that has already been done, like the *STM Kargu* which according to the UN security council's panel of experts was utilized in Libya 2020 to hunt down and kill a human target.

On the other hand, improved object detection has the possibility to save countless lives, not only when used for full AD in the future, but when used for active safety systems in today's implementations. As more and more new cars are being equipped with advanced LiDAR sensors [14], this is of increasing importance.

Bibliography

[1] Xuyang Bai, Zeyu Hu, Xinge Zhu, Qingqiu Huang, Yilun Chen, Hangbo Fu, and Chiew-Lan Tai. Transfusion: Robust lidar-camera fusion for 3d object detection with transformers. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1080–1089, 2022.

[2] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020.

[3] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11618–11628, 2020.

[4] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 213–229. Springer, 2020.

[5] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. Mmdetection: Open mmlab detection toolbox and benchmark, 2019.

[6] Qi Chen, Lin Sun, Zhixin Wang, Kui Jia, and Alan Yuille. Object as hotspots: An anchor-free 3d object detection approach via firing of hotspots. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXI 16*, pages 68–84. Springer, 2020.

[7] Shuyang Cheng, Zhaoqi Leng, Ekin Dogus Cubuk, Barret Zoph, Chunyan Bai, Jiquan Ngiam, Yang Song, Benjamin Caine, Vijay Vasudevan, Congcong Li, et al. Improving 3d object detection through progressive population based augmentation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXI 16*, pages 279–294. Springer, 2020.

[8] MMDetection3D Contributors. MMDetection3D: OpenMMLab next-generation platform for general 3D object detection. `https://github.com/open-mmlab/mmdetection3d`, 2020.

[9] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–308, 2009.

[10] Matthias Fey. Rusty1s/pytorch_cluster: Pytorch extension library of optimized graph cluster algorithms, Mar 2023.

[11] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.

[12] Benjamin Graham. Spatially-sparse convolutional neural networks, 2014.

[13] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks, 2017.

[14] Volvo Car Group. *Next generation pure electric Volvo comes with LiDAR technology and AI-driven super computer as standard to help save lives*, Jun 2021.

[15] Martin Hahner, Dengxin Dai, Alexander Liniger, and Luc Van Gool. Quantifying data augmentation for lidar based 3d object detection, 2022.

[16] Jordan SK Hu and Steven L Waslander. Pattern-aware data augmentation for lidar 3d object detection. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 2703–2710. IEEE, 2021.

[17] Xuzhong Hu, Zaipeng Duan, and Jie Ma. Context-aware data augmentation for lidar 3d object detection, 2022.

[18] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12697–12705, 2019.

[19] Daeun Lee and Jinkyu Kim. Resolving class imbalance for lidar-based object detector by dynamic weight average and contextual ground truth sampling. In *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 682–691, 2023.

[20] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[21] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[22] Zhijian Liu, Haotian Tang, Alexander Amini, Xingyu Yang, Huizi Mao, Daniela Rus, and Song Han. Bevfusion: Multi-task multi-sensor fusion with unified bird's-eye view representation. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

[23] Jiageng Mao, Minzhe Niu, Chenhan Jiang, Hanxue Liang, Jingheng Chen, Xiaodan Liang, Yamin Li, Chaoqiang Ye, Wei Zhang, Zhenguo Li, Jie Yu, Hang Xu, and Chunjing Xu. One million scenes for autonomous driving: Once dataset, 2021.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance

deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[25] 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*, pages 652–660, 2017.

[26] Matthias Reuse, Martin Simon, and Bernhard Sick. About the ambiguity of data augmentation for 3d object detection in autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 979–987, 2021.

[27] Danila Rukhovich, Anna Vorontsova, and Anton Konushin. Imvoxelnet: Image to voxels projection for monocular and multi-view general-purpose 3d object detection. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2397–2406, 2022.

[28] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[29] Petr Šebek, Šimon Pokornỳ, Patrik Vacek, and Tomáš Svoboda. Real3d-aug: Point cloud augmentation by placing real objects with occlusion handling for 3d detection and segmentation. *arXiv preprint arXiv:2206.07634*, 2022.

[30] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020.

[31] Chunwei Wang, Chao Ma, Ming Zhu, and Xiaokang Yang. Pointaugmenting: Cross-modal augmentation for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11794–11803, 2021.

[32] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. Deep graph library: A graph-centric, highly-performant package for graph neural networks. *arXiv preprint arXiv:1909.01315*, 2019.

[33] Erik Wijmans. Pointnet++ pytorch. *https://github.com/erikwijmans/Pointnet2_PyTorch*, 2018.

[34] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018.

[35] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11784–11793, 2021.

[36] Wu Zheng, Weiliang Tang, Li Jiang, and Chi-Wing Fu. Se-ssd: Self-ensembling single-stage object detector from point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14494–14503, 2021.

[37] 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.

[38] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. Class-balanced grouping and sampling for point cloud 3d object detection, 2019.

A

Appendix A

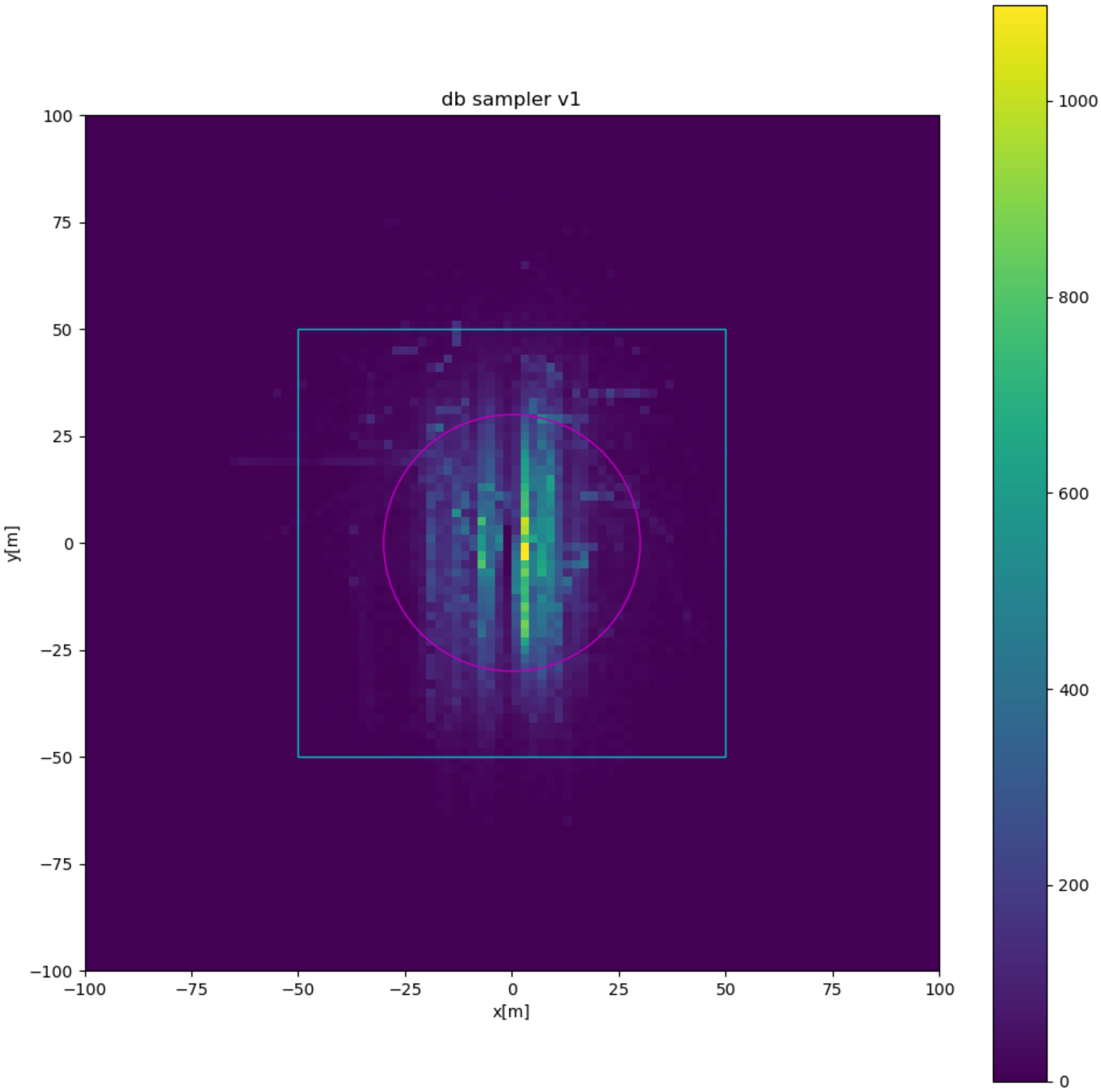


Figure A.1: Barrier original distribution.

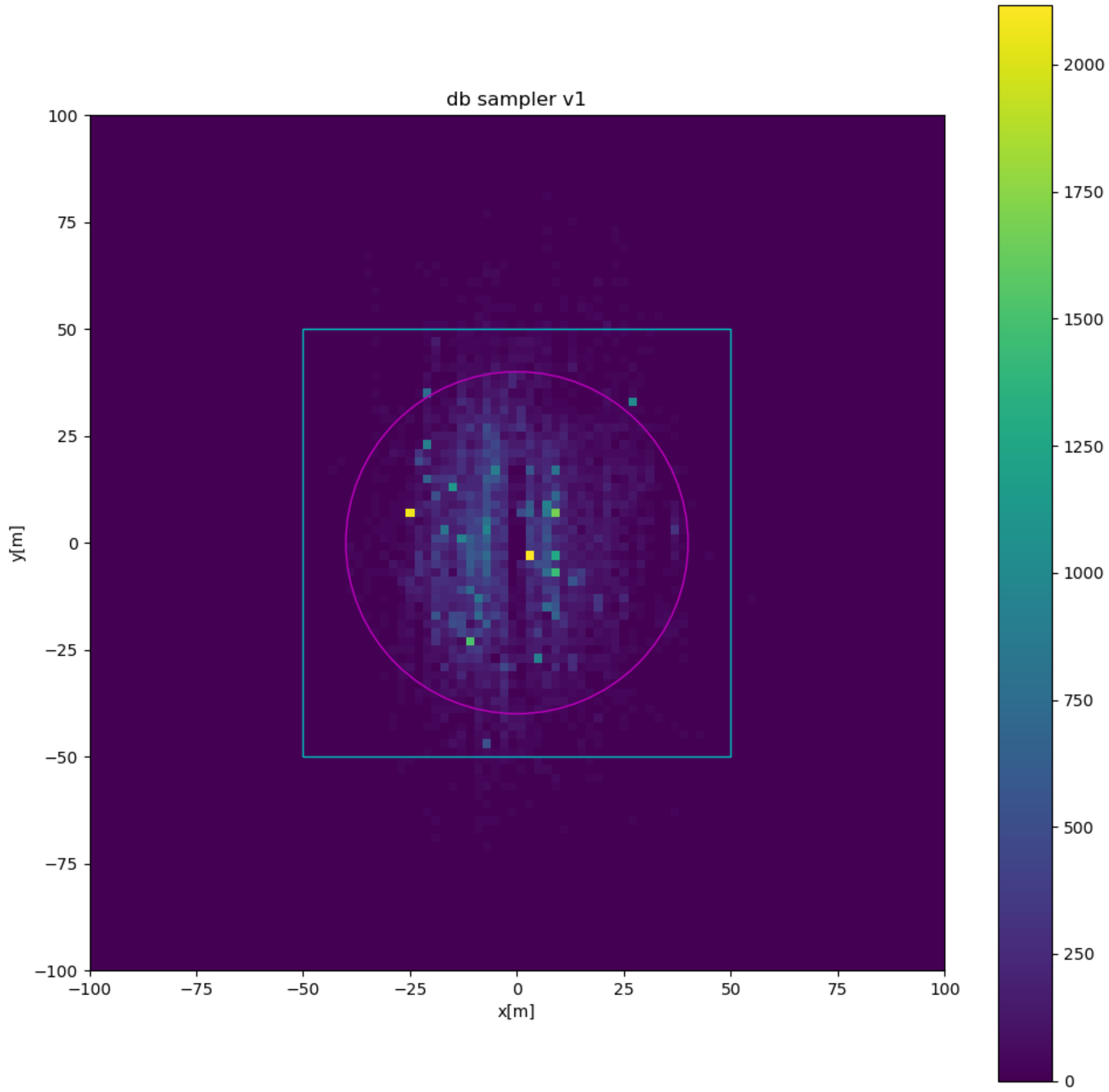


Figure A.2: Bicycle original distribution.

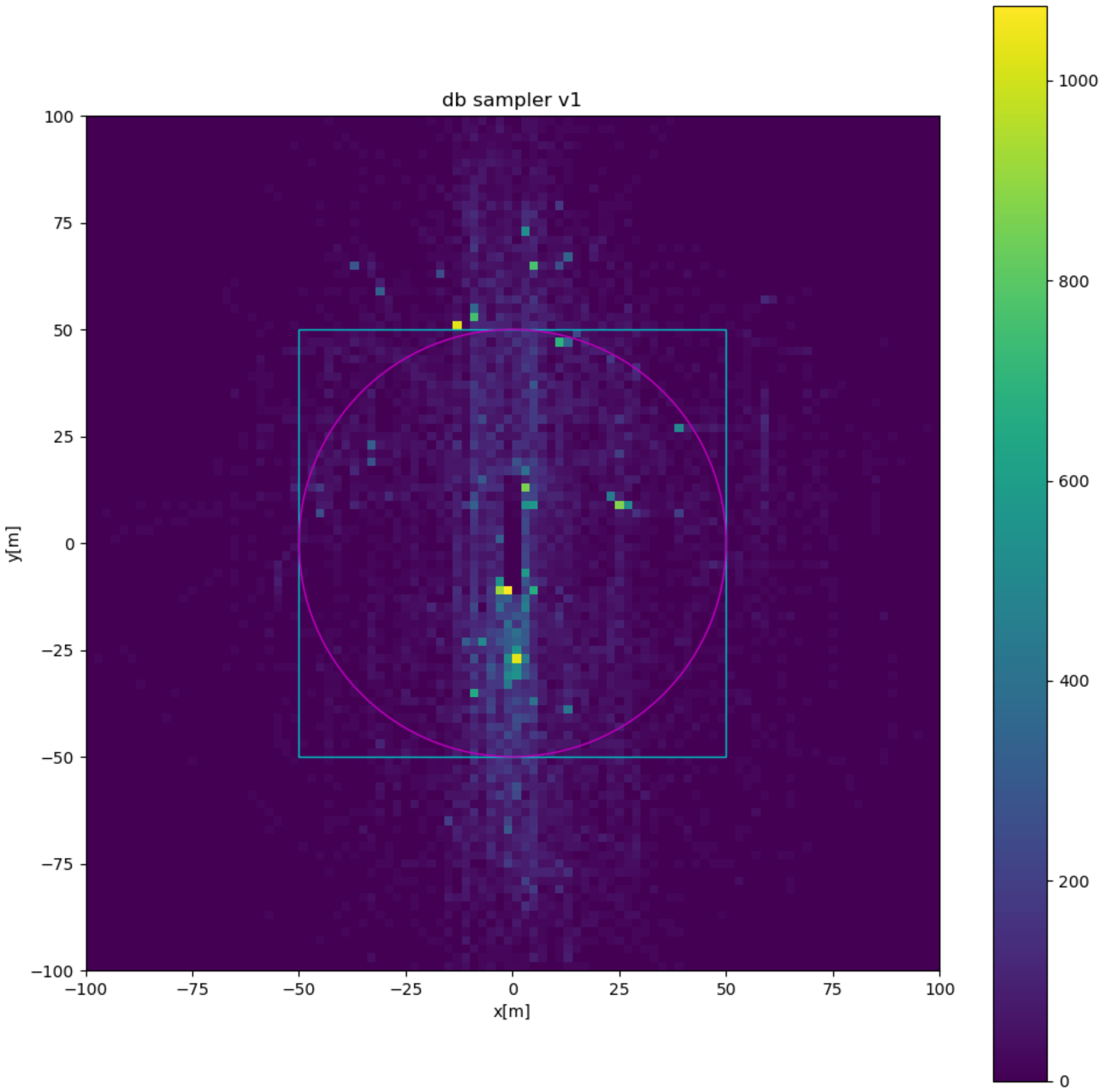


Figure A.3: Bus original distribution.

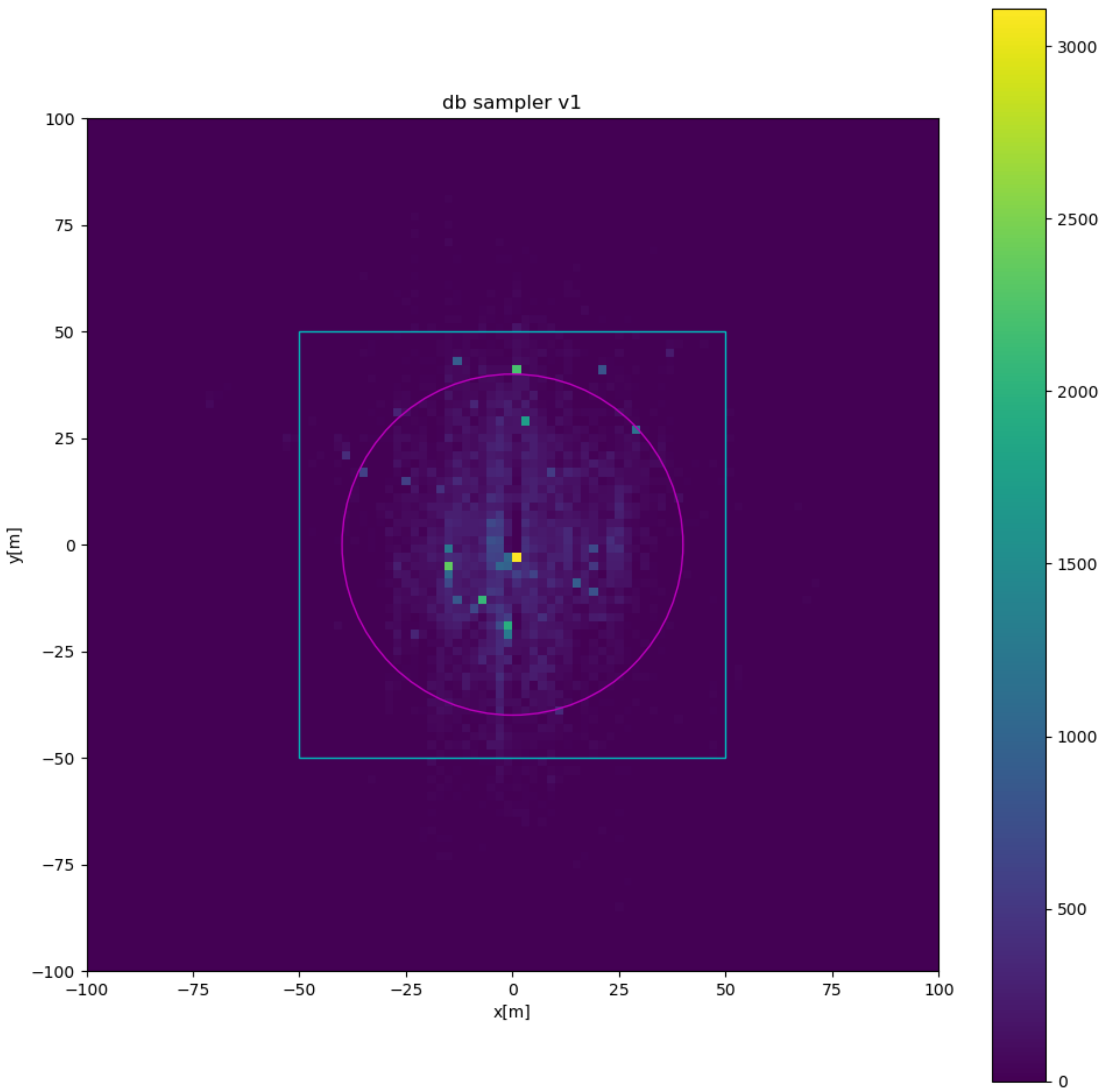


Figure A.4: Motorcycle original distribution.

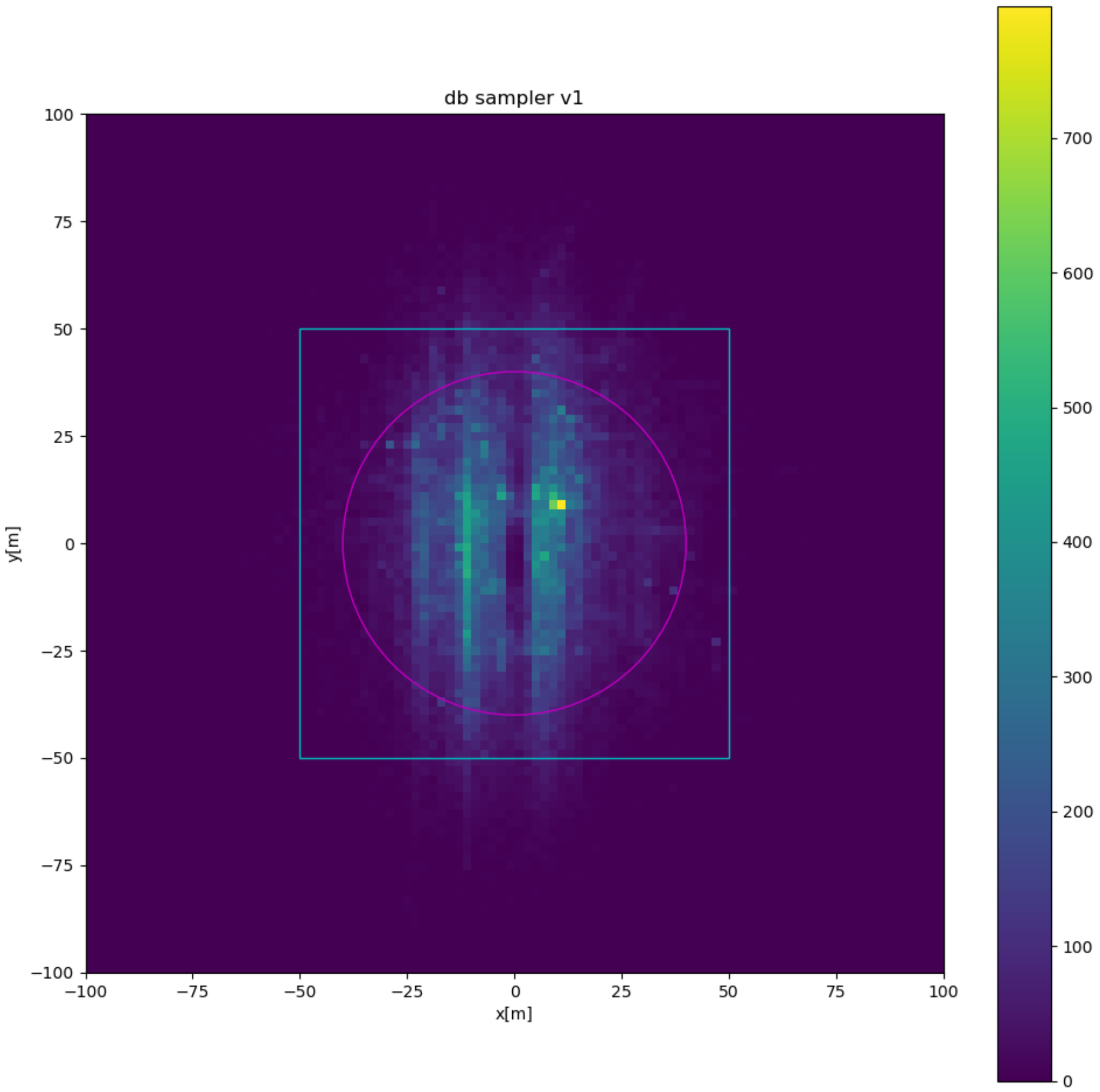


Figure A.5: Pedestrian original distribution.

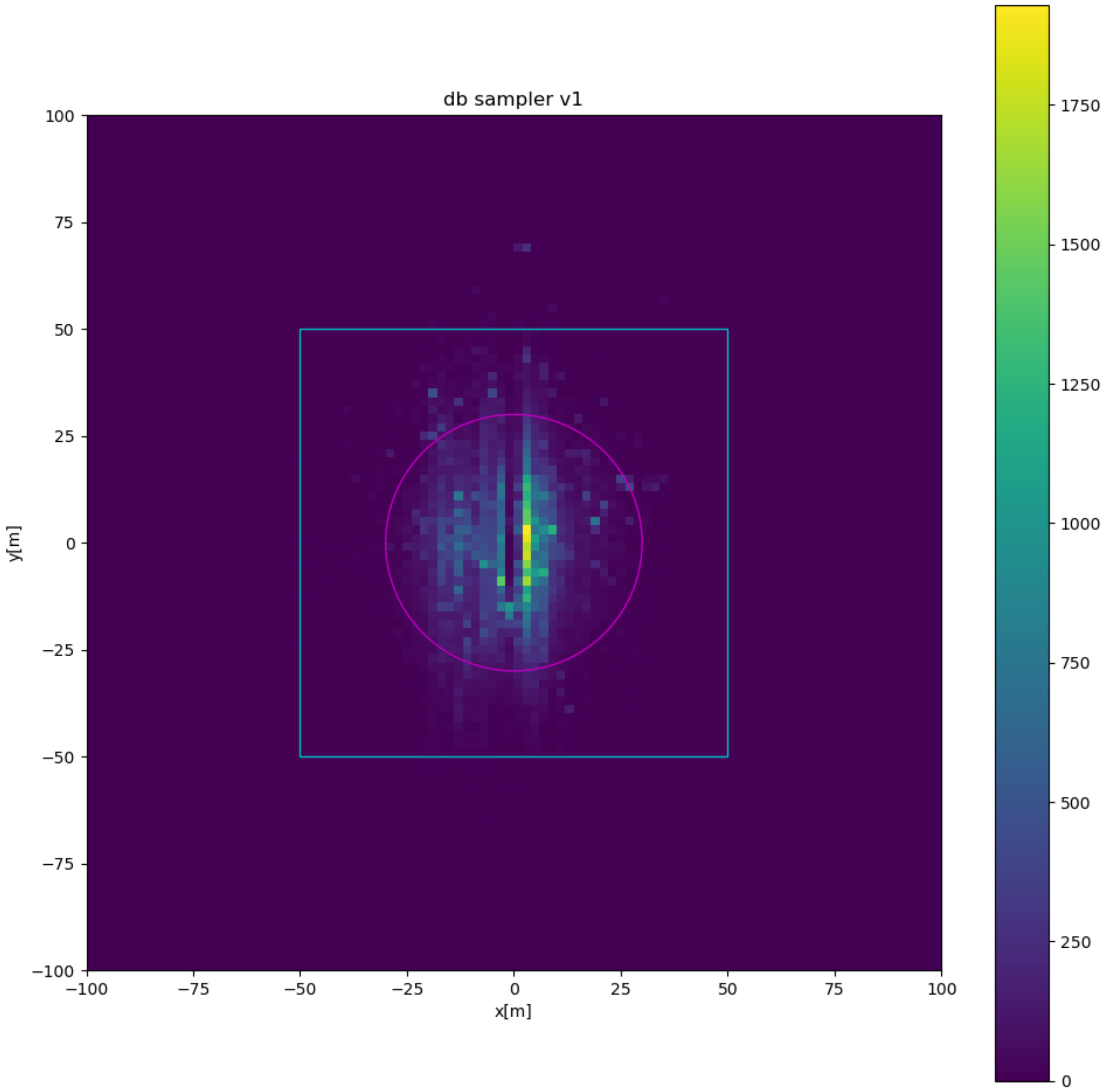


Figure A.6: Traffic Cone original distribution.

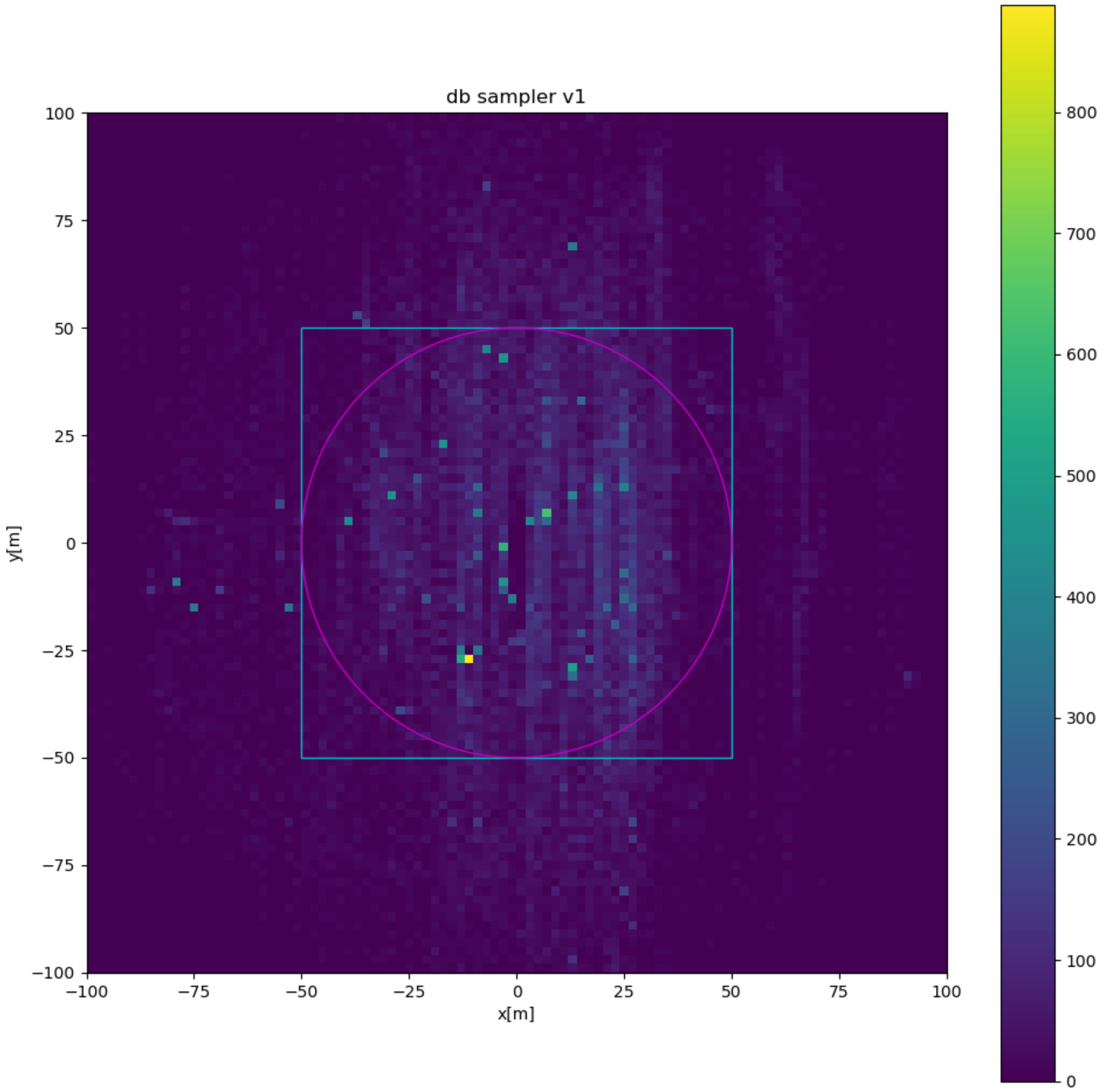


Figure A.7: Trailer original distribution.

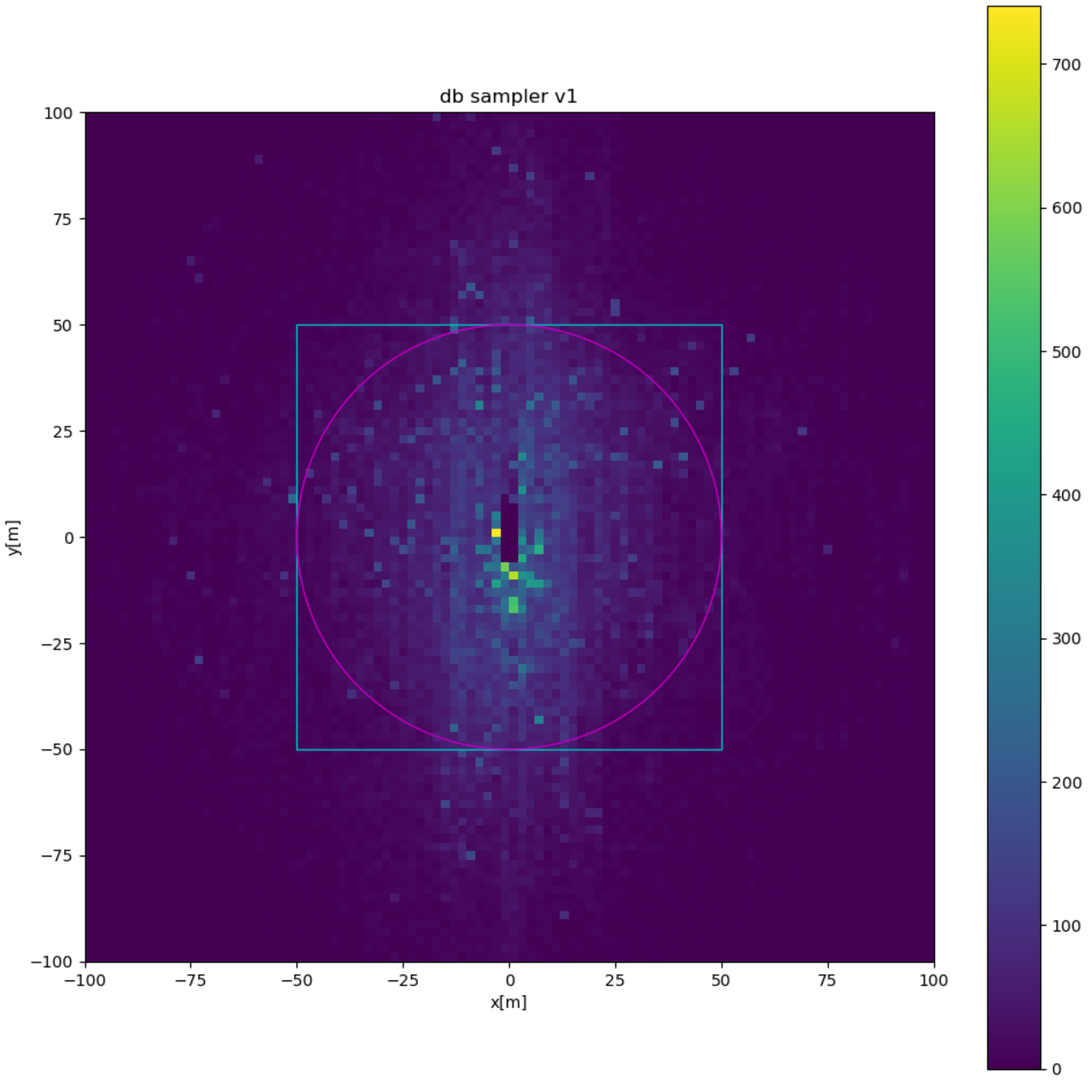


Figure A.8: Truck original distribution.

B

Appendix B

Table B.1: CenterPoint Baseline differences between runs. Results presented before Fade (b.F.) and after Fade.

Class	**Metric**													
	AP		ATE		ASE		AOE		AVE		AAE			
	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade		
Car	**0.002**	0.001	0.002	0.001	0.001	0.002	0.005	0.003	0.004	0.014	0.004	0.003		
Truck	**0.013**	0.013	0.019	0.021	0.006	0.008	0.028	0.030	0.010	0.035	0.014	0.006		
Bus	0.009	0.024	0.013	0.013	0.010	0.007	0.064	0.018	0.044	0.030	0.005	0.006		
Trailer	0.003	0.009	0.020	0.018	0.009	0.013	0.049	0.042	0.013	0.031	0.013	0.019		
C.V.	0.007	**0.016**	0.030	0.046	0.012	0.010	0.157	0.226	0.009	0.003	0.018	0.047		
Ped.	0.007	**0.011**	0.000	0.001	0.003	0.005	0.012	0.021	0.007	0.004	0.005	0.006		
Mot.	0.010	0.012	0.004	0.002	0.003	0.005	0.052	0.041	0.144	0.033	0.009	0.023		
Bicycle	0.014	0.035	0.007	0.003	0.003	0.003	0.147	0.138	0.006	0.014	0.006	0.002		
T.C.	0.010	0.010	0.011	0.003	0.009	0.012	n/a	n/a	n/a	n/a	n/a	n/a		
Barrier	0.008	**0.022**	0.002	0.005	0.003	0.001	0.009	0.008	n/a	n/a	n/a	n/a		
	mAP		mATE		mASE		mAOE		mAVE		mAAE		NDS	
	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade
Overall	0.001	0.003	0.001	0.007	0.002	0.001	0.035	**0.046**	0.021	0.005	0.004	**0.008**	0.004	0.003

Table B.2: CenterPoint HRFiSv2 differences between runs. Results presented before Fade (b.F.) and after Fade.

Class	**Metric**													
	AP		ATE		ASE		AOE		AVE		AAE			
	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade		
Car	0.001	0.001	0.002	0.002	0.001	0.000	0.004	0.003	0.002	0.003	0.006	0.009		
Truck	0.012	0.011	0.004	0.013	0.003	0.006	0.033	0.012	0.019	0.005	0.019	0.009		
Bus	**0.032**	0.040	0.034	0.035	0.008	0.012	0.013	0.014	0.033	0.049	0.012	0.017		
Trailer	**0.013**	0.006	0.025	0.025	0.004	0.005	0.017	0.038	0.045	0.042	0.005	0.025		
C.V.	0.013	0.013	0.010	0.050	0.017	0.016	0.120	0.093	0.007	0.014	0.033	0.036		
Ped.	0.010	0.008	0.003	0.002	0.001	0.001	0.017	0.027	0.003	0.004	0.003	0.005		
Mot.	**0.014**	0.003	0.007	0.014	0.004	0.006	0.019	0.022	0.184	0.049	0.030	0.034		
Bicycle	**0.041**	0.004	0.003	0.006	0.006	0.006	0.033	0.036	0.046	0.037	0.007	0.002		
T.C.	0.010	**0.020**	0.011	0.008	0.002	0.004	nan	nan	nan	nan	nan	nan		
Barrier	0.011	0.002	0.000	0.008	0.005	0.003	0.011	0.009	nan	nan	nan	nan		
	mAP		mATE		mASE		mAOE		mAVE		mAAE		NDS	
	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade	b.F.	Fade
Overall	**0.007**	0.005	**0.008**	0.008	0.001	**0.002**	0.005	0.011	**0.027**	0.009	0.002	0.004	**0.004**	0.004

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se