RWTH Aachen University
Faculty of Electrical Engineering

Institute of High Frequency Technology Engineering
Chair of Radar Systems Engineering
Univ.-Prof. Dr.-Ing. Peter Knott

Master's Thesis

Analysis of Uncertainty Estimates for Transformer-based Radar-, Camera-, and LiDAR-driven 3D Object Detectors

The present work was submitted to Institute of High Frequency Technology Engineering and Institute for Automotive Engineering
by:

Mr. B.Sc. Stefan-Daniel Vilceanu, Matr.-No.: 407531

Supervisor:

Till Beemelmanns, M.Sc.

First Examiner:
Univ.-Prof. Dr.-Ing. Peter Knott

Second Examiner:
Univ. -Prof. Dr.-Ing. Lutz Eckstein

Aachen, November 2024

Contents

1 Introduction

Artificial Intelligence (AI) has become a cornerstone of modern technology, making its marks into numerous and diverse fields, such as healthcare, transportation or finance. At its core, AI aims to create systems that can simulate human intelligence by analyzing data and learning from identified patterns in order to be able to make decisions on their own or with minimal human assistance. Deep learning, a major subset of AI, has boosted the advancements further by enabling machines to learn complex data features from vast datasets. Leveraging neural networks and complex architectures, deep learning models have achieved remarkable performances for tasks such as image classification and object detection becoming an interesting prospect for fields like the autonomous driving sector.

Object detection is a fundamental task in computer vision that involves identification, classification and precise localization of objects, normally computed through bounding boxes. This capability is crucial especially for autonomous driving applications, since being aware of the exact position, scale and type of an object can aid in making the best possible decision for every scenario. Extending this to 3D Object Detection introduces further complexity, by having to estimate an additional third dimension, the depth of an object, together with its orientation in space, which, in the context of autonomous driving, should yield even better decision-making. By integrating LiDAR, radar and camera sensors, machines are able to understand their surroundings with certain confidence.

Particularly for safety-critical application, such as in the case of autonomous driving, it is crucial that predictions are accurate and trustworthy. Therefore, estimating uncertainty is an essential component in the process of developing robust and reliable AI models. Reliability describes the rapport between the model's predictions and confidence assessments, while robustness refers to a model's ability to perform well under challenging conditions. Fig. 1-1 illustrates a camera image of a moose on the road that is being misclassified as a pedestrian with a high confidence score of 87%. This is an example of epistemic uncertainty, which appears when the model has not seen enough training data of a specific object. In the context of 3D Object Detection, where the goal is to detect and localize an object within a three-dimensional space, uncertainty quantification can help assess the model's confidence in its prediction, this being crucial when making informed decisions, as understanding when a model might be uncertain under more ambiguous scenarios could enable the system to act more cautiously.

Problem Statement

Despite recent advancements in 3D Object Detection, there is a substantial gap in the literature regarding uncertainty quantification for 3D state-of-the-art transformer-based architectures. Most of the studies on uncertainty focus primarily on 2D data or limit their scope to single type of sensor input, overlooking research that analyzes all three main types of sensors, camera, radar and LiDAR. However, for better applications in safety-critical environments like autonomous driving, understanding and accurately estimating uncertainty for 3D predictions is crucial. This thesis aims to address and fill some of the current literature gaps by developing an analysis of uncertainty estimates for

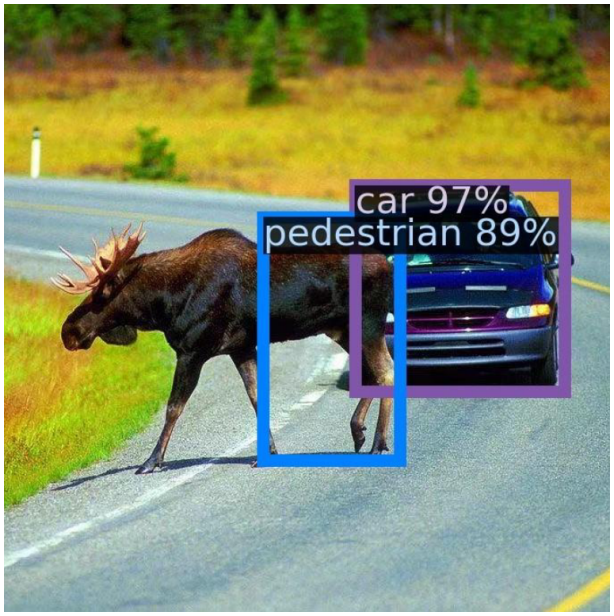


Fig. 1-1: Misclassification of a moose as a pedestrian in a safety-critical environment [DU22].

performant 3D transformer-based architectures and to contribute with a more robust and reliable approach for calibrating 3D object detectors.

Objectives

This thesis will focus on achieving the following objectives:

1. Study of the decoder side of two transformer-based architectures. The first one, Positional Embedding Transformation (PETR), will be trained on the camera data of the NuScenes dataset, while the other architecture, PillarFormer, is to be trained using radar and LiDAR point clouds of the View-of-Delft dataset.

2. Implementation of an uncertainty quantification baseline for both architectures, including popular methods such as Monte-Carlo Dropout, Deep Ensembles and Deep Sub-Ensembles. These will serve as a comparison tool for further calibration techniques used in this thesis.

3. Implementation of uncertainty calibration methods designed to improve reliability of models without requiring large amounts of computational resources such as in the case of the baseline methods.

4. Present a comprehensive analysis between the baseline and the calibration techniques based on various metrics, such as Expected Calibration Error, Negative Log Likelihood adjusted for both the classification and regression tasks, and Mahalanobis Distance.

Thesis Structure

The structure of this work is organized into five main chapters, following the introductory chapter, each building on the foundations of the previous one in order to create a clear progression from the theory and until the final conclusion. The theoretical chapter provides an in depth explanation on the necessary background information needed to understand the core concepts of this thesis,

starting with a short introduction to Machine Learning, followed by Deep Learning concepts and several overviews on 3D Object Detection, sensor types, uncertainty quantification baselines and metrics used. The next chapter, Related Work, examines existing studies and resources, such as the two architectures used, PETR and PillarFormer, the datasets of NuScenes and View-of-Delft and recent uncertainty estimates researches for object detection. The fourth chapter outlines the taken methodology by describing the general process, the proposed calibration techniques, the clustering of multiple predictions and finally, the flow of the experiments. Furthermore, the fifth chapter presents the gathered results that differentiate based on calibration technique, architecture, dataset or sensor type, as well as a baseline to compare these results to. Finally, the last chapter concludes this thesis by summarizing key findings and proposing future research.

2 Theoretical Background

This chapter aims to bring a broader theoretical understanding of the subjects that are going to be tackled in this work. Its structure consists of a short introduction to Machine Learning (ML), a pioneering field of Artificial Intelligence, followed by a more advanced continuation of ML, Deep Learning (DL). Different concepts of DL as well as its applications are introduced, focusing mostly on the transformer architecture in the context of 3D Object Detection, which build the foundation of this thesis. Moreover, three types of sensors for data acquisition are described, followed by an introduction to the concept of uncertainty and to the approaches for quantifying uncertainty. Finally, this chapter ends by presenting evaluation metrics and plot for uncertainty estimates.

2.1 Artificial Intelligence and Machine Learning

Artificial Intelligence is a broad field in computer science that tries to develop models or systems that are capable of executing tasks that would usually require human intelligence or interaction. The main idea is to simulate intelligent behavior using machines [RUS16].

Machine Learning is a subfield of AI that concentrates on utilizing colloquial data to enable machines to learn patterns or relationships within it. Therefore, the machines are able to make decisions, improve from these, and then make newer, better decisions. This concept defines the training of a ML model. Another essential characteristic of a ML system is the ability to generalize to unseen data. Using the learnt parameters from the training step of the model, new decisions have to be taken on inputs that the model is unfamiliar with, which describes the concept of testing or validating. It tests or validates how well the model generalizes to unseen data [CHO21]. There are multiple methodologies of ML, but the most common ones are:

- **Supervised Learning** - the ML model is trained on an already labeled dataset. In this approach, each input data has one or multiple corresponding target outputs and the job of the model is to map the inputs to the labeled targets. The main goal is then to make predictions on unseen data based on what it has already learned using the labeled data. The first of two primary tasks of the supervised learning is to do classifications, attributing one or multiple discrete labels to a certain input. For example, labeling particular objects in an image to predefined classes. The second task of this ML type is performing regression, which is the process of predicting continuous variables based on a certain input. An example of a regression task is the predicted length and width of a bounding box from an object of the input image. The work of this thesis makes mostly use of the supervised learning type in the context of 3D Object Detection by classifying objects and making prediction for the 3D bounding boxes of these.
- **Unsupervised Learning** - a type of ML where the model trains on unlabeled data. This means that for a certain input there is not a predefined target output. Therefore, the job of an unsupervised learning model is to find relationships or patterns within the data, without being sure of what is correct. This ML type is generally used for different tasks compared to the supervised learning. The most common task that unsupervised learning performs is clustering, which describes the technique of grouping similar data points together based on

their features. In this thesis, clustering techniques like DBSCAN are performed for merging close objects together.

- **Reinforcement Learning** - an ML technique that tries to achieve the most optimal result by mimicking the trial-and-error learning process similar to how humans behave to achieve their goals. The reinforcement learning algorithms use a reward-and-punishment method during data processing. Actions that work towards the goal are reinforced or rewarded, while actions that lead away from it are ignored or punished.

2.2 Deep Learning

Deep learning is a subset of machine learning that has its focus on more sophisticated approaches, different from the classical machine learning methods, such as Artificial Neural Networks (ANN). DL involves training these more advanced networks on large amounts of data so that they can be used for solving complex tasks. The idea behind these ANN is to partially mimic the functionality of the human brain. Therefore, these neural networks are composed of layers of interconnected nodes that act in a similar way as synapses and neurons function in a biological brain. The word "deep" in deep learning refers to the use of multiple layers in a network, which enables the model to solve increasingly sophisticated problems, but at the same time it raises the complexity of the network. An example of a deep neural network can be seen in Fig. 2-1 where a multi-layer network is shown. The input layer is the one that receives the raw data, while the L hidden layers are responsible to perform transformations on the input from the previous layer. At last, the output layer gives the final result, which can either be probabilities for multiple classes in classification tasks or a continuous variable in the case of regression problems [BEN17].

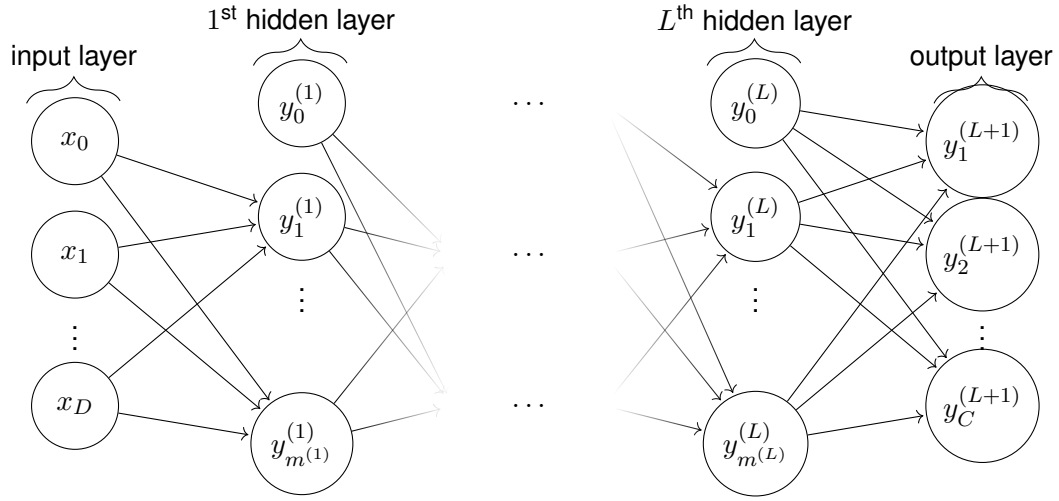


Fig. 2-1: Network graph of a $(L+1)$-layer perceptron with D input units and C output units. The l^{th} hidden layer contains $m^{(l)}$ hidden units.

2.2.1 Artificial Neural Networks

Artificial Neural Networks, simpler called neural networks (NN), represent the base of deep learning. As previously mentioned, NNs are made up of multiple layers that consist of, usually, lots of neurons

that are connected between them. The neuron, sometimes called node, is a computational unit that takes an input, processes it and can produce an output. It is analogous to a biological neuron. Besides neurons, input, output and hidden layers, ANNs contain two more important parameters, which influence the connections between the nodes. The first parameter is a multiplicative factor for a node, called a weight, that serves as its scaler. The weight of a node specifies how important a certain input is for that node. During the training process, weights are permanently readjusted for every new input in order to minimize the error in the model's prediction. The second parameter is represented by the bias. This is an additive term, added to the product of inputs and weights. Its role is to positively or negatively offset the result at every single neuron before being put through an activation function. The activation function has a crucial role in neural networks theory, these allowing the model to learn complex relationship by introducing non-linearity into the network. Some popular examples of activation functions are sigmoid:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \qquad \text{Eq. 2-1}$$

which was one of the first non-linear activations used, ideal for binary classification tasks because of producing an output between 0 and 1. Other examples are ReLU (Rectified Linear Unit):

$$\textit{ReLU}(z) = \max(0, z), \qquad \text{Eq. 2-2}$$

which is still highly used due to its simplicity and effectiveness, and Softmax:

$$\textit{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}, \qquad \text{Eq. 2-3}$$

which is used for scenarios of multi-class classification because it manages to convert an usual multi-node output into a probability distribution [KRE11].

In order to process an output, the network does a so called forward propagation. In this step, the input data propagates through the network layer by layer, where each neuron from every layer takes the input from the previous layer in order to compute a weighted sum and apply an activation function to it. This result will eventually be used in the same calculation process of the next layer and so on, until the output layer and final result is reached.

$$z_j = \sigma(\sum_{i \in I} x_i w_i + b) \qquad \text{Eq. 2-4}$$

shows the basic calculation of a node from current layer j. The results from all previous nodes, x, are multiplied by their corresponding weight w and a bias b of the current node is added. The result is put through the activation function σ that introduces the desired non-linearity.

After a first forward pass through the network, the result is in most cases not yet right. Therefore, the network needs to undergo a training phase in order to adjust the weights and biases of the nodes. This adjustment can be done by tracking the prediction error and aiming at minimizing it through

a loss function. The loss function works by measuring how far apart the predicted output is to the target value. Two very popular loss functions are the Mean Squared Error (MSE) for regression tasks and the Cross-Entropy Loss for classification tasks. The MSE function

$$\textit{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \qquad \text{Eq. 2-5}$$

calculates the sum of squared differences between the true value y_i and the predicted value $\hat{y}_i$. The Cross-Entropy Loss

$$\textit{Cross-Entropy Loss} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i), \qquad \text{Eq. 2-6}$$

is the negated sum of the product between the true value y_i and the logarithm of the predicted value $\hat{y}_i$, with N being the number of classes.

Only after the prediction error has been assessed, the network can enter its learning phase by using backpropagation to adjust the weights. This technique works by calculating the gradient of the loss function with respect to every weight in the network and has a bottom-up approach by beginning at the output layer and going back through the entire network [BEN17]. The gradient calculation happens during backpropagation, and the gradients are adjusted using the gradient descent algorithm, which also has several more performant and complex variations like Stochastic Gradient Descent (SGD), Mini-batch Gradient Descent or the well known Adam Optimizer [KIN14]. This algorithm responsible for adjusting the model weights makes use of a so called learning factor, which is responsible to scale the magnitude of weight updates during their readjustment. It has a very important role, as it impacts the speed of how fast the algorithm converges or, in the case of a poorly chosen value, the algorithm might also diverge, thus the network will not be able to learn properly.

As part of the training process, hyperparameters must be tuned when working with neural networks. Hyperparameters are called as such, because they configure the whole network architecture as well as the training procedure and must be differentiated from the model parameters, which are the internal weights and biases. As mentioned before, learning rate plays a big role in the training mechanism, but additional hyperparameters like batch size, number of epochs, the network architecture, with the number of layers and neurons, activation functions, loss functions and some more are also key to achieving a well trained DL model [YAN20].

2.2.2 Convolutional Neural Networks

Traditional NNs start to struggle when more complex data is applied. High dimensionality data, e.g. pixels of a 2D image or a 3D volume, would make the number of parameters in such a network explode, which would be extremely computationally expensive. Moreover, simple neural networks treat all input features equally and do not take into consideration the spatial relationships between the pixels in an image, which would make tasks like object detection or image segmentation quite

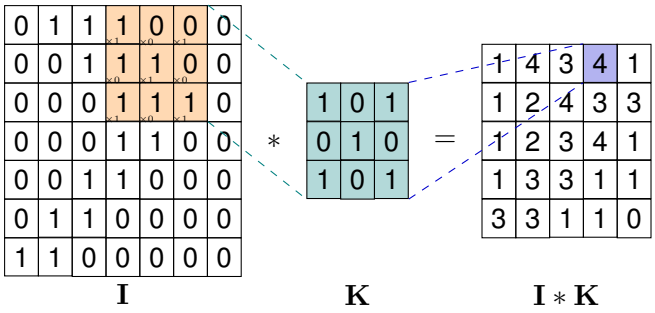


Fig. 2-2: Two dimensional convolution with a 3x3 filter.

difficult. Convolutional Neural Networks (CNNs) were designed to overcome these challenges by introducing some additional innovations to the main architecture.

The first feature added to the CNN are convolutional layers. Instead of having every neuron connected to a pixel of the input image, CNNs ensure only a small, local connection to the input, called receptive field. This connected region is able to detect local features, such as edges or corners, in the image. The main benefit of this approach is the possibility to share the same set of weights across the whole image, usually known as filter or kernel, that would have to the task to identify a specific feature over all the regions of the image. Overall, multiple different filters, each with their own trained weights, are then slid over the input image. For each used filter, a feature map is output. The feature map then indicates if the kernel has found a certain feature. Earlier layers are responsible for finding low-level characteristics, while deeper layers are looking for higher-level patterns. Therefore, by using these techniques, not only that the number of weights is kept much lower than when using a traditional neural network, but the relationship between local pixels is taken into consideration. One more advantage of CNNs is that they exhibit translation invariance, meaning that the network can still detect an object if the image is shifted or translated [OSH15] [GU18].

On the left side of Fig. 2-2, the input image **I** is shown, **K** is 3x3 the kernel, and the third block represents the newly processed feature map. Furthermore, the convolved region is depicted in orange, and it performs a matrix inner product (summation of the element-wise multiplication of the two matrices), the final result being shown in purple.

Additionally, the CNN architecture has further tools to reduce the dimensionality of inputs using pooling layers. These generally reduce the complexity of the network and make it more robust to variations in the output. The idea behind a pooling layer is to downsample feature maps in a specific manner, for example, by taking the maximum of each small region of the feature map, therefore achieving robustness and a lower sensitivity to translations and minor changes in the position of objects in the image.

Although traditional NNs are not well suited to handle high dimensional data, they are not completely ignored when it comes to CNNs. Convolutional and pooling layers are used to process the raw image in order to extract plenty of features, but to the end of the network, most architectures include at least one fully connected layer, as in the traditional NN. These layers process the highly downsampled, high-level feature maps to produce suitable outputs, which are usually either image classifications, object detections or image segmentations.

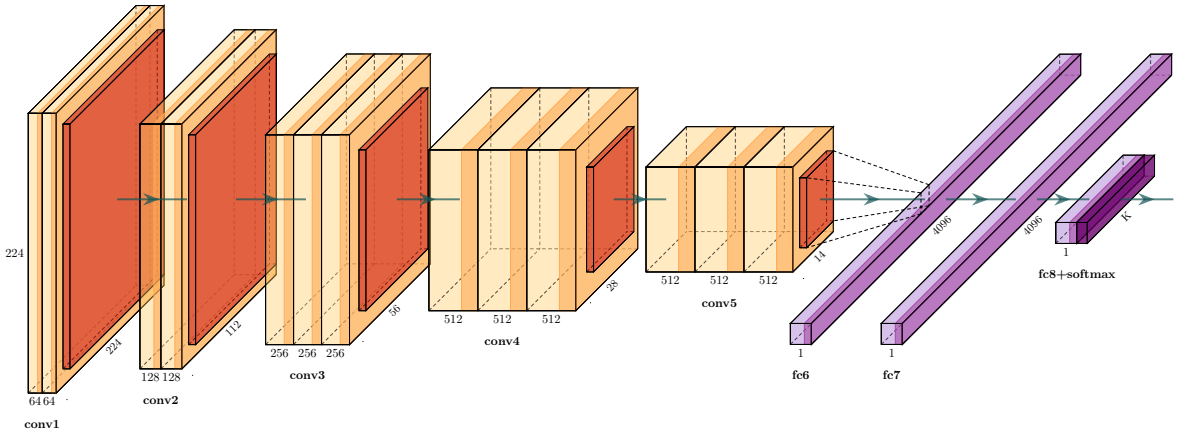


Fig. 2-3: Architecture of the VGGNet.

Fig. 2-3 illustrates a well-known CNN architecture, namely the VGGNet [SIM14]. It consists of five different convolutional layers, and it can be seen that while the number of feature maps grows with every layer, the resolution of these feature maps gets lower and lower. After the last convolutional layer, there are three different fully connected layers that handle the highly processed features. Lastly the output is put through a Softmax activation function that would transform the raw logits into class probabilities.

2.2.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are another main type of neural network [MED01]. RNNs are designed to handle sequential data by maintaining a form of memory about past inputs. This is done by implementing "loops" in the architecture that allow information to recur, greatly improving the order and temporal relationships between important data points. The core innovation that makes RNNs be able to store information are hidden states. These are recursively updated at every step and enable the network to remember previous inputs [GRO13]. The main application field for which RNNs are used is Natural Language Processing (NLP). RNNs also come with downsides, the main one being exactly the sequential processing, which limits parallelization and makes training more resource-intensive. A second flow of the RNNs is the difficulty to remember information over long sequences, due to vanishing gradients during the training process. To overcome these issues, a newer DL approach, the transformer architecture, was developed.

2.2.4 Transformers

The transformer architecture is a type of deep learning designed to also handle sequential data and typically used for NLP. It has revolutionized the advancement of NLP tasks since its introduction in 2017 from the "Attention is all you need" research paper [VAS17], due to its major improvements in terms of efficiency but most importantly, in terms of long-range dependencies over the data points in comparison to the RNN.

The transformer architecture is based on an encoder-decoder structure. The encoder consist of multiple layers and its purpose is to process the input sequence, while the decoder, which also consists

of a stack of layers, aims at utilizing the output from the encoder together with previously generated tokens to output the next token in the sequence. A token is the basic unit of input processed from raw data. In the case of NLP, the tokens can be normal words or sub-words, while in the case of the Vision Transformer, the tokens are usually patches of an image.

The innovation of the transformer architecture, which also forms its foundation, comes from the newly designed method to process the data, called Attention Mechanism. This method allows each input token to learn relationships about other tokens in the sequence. This mechanism computes a weighted sum of values based on attention scores, that calculate the relevance of different tokens to each other. First, all the input tokens are linearly transformed into three vectors, query, key and value, which are obtained by multiplying the input vector with the corresponding learned weight matrix. This procedure is described by the following three equations:

$$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q \quad \text{Eq. 2-7}$$

$$\mathbf{K} = \mathbf{X}\mathbf{W}_K \quad \text{Eq. 2-8}$$

$$\mathbf{V} = \mathbf{X}\mathbf{W}_V \quad \text{Eq. 2-9}$$

where:

- $\mathbf{Q}$ is the query matrix,
- $\mathbf{K}$ is the key matrix,
- $\mathbf{V}$ is the value matrix,
- $\mathbf{X}$ is the input sequence matrix,
- $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ are learned weight matrices for the query, key, and value, respectively.

Afterwards, the attention score between two tokens is calculated by taking the dot product of their corresponding query and key vectors. This product gives the similarity between a token i, which is the query, and a token j, which represents the key. If the attention score is higher, then token i will pay more attention to token j. Additionally, it has been observed that scaling the attention score with the square root of the dimension of the query or key vectors, d, helps stabilize the gradients during training. Moreover, to ensure that the attention scores form a probability distribution, the Softmax function is applied on the product, resulting in the attention weights α_{ij}. These weights give out how much attention token i pays to token j and can be described by the following formula:

$$\alpha_{\mathbf{ij}} = \frac{\exp\left(\frac{\mathbf{Q}_i \cdot \mathbf{K}_j}{\sqrt{d_j}}\right)}{\sum_k \exp\left(\frac{\mathbf{Q}_i \cdot \mathbf{K}_k}{\sqrt{d_k}}\right)} \quad \text{Eq. 2-10}$$

Finally, the last step of the attention calculation is computing the weighted sum over all the tokens of the product between the value vectors and the attention weights for the token i. Eq. 2-11 shows the output for a token i as following:

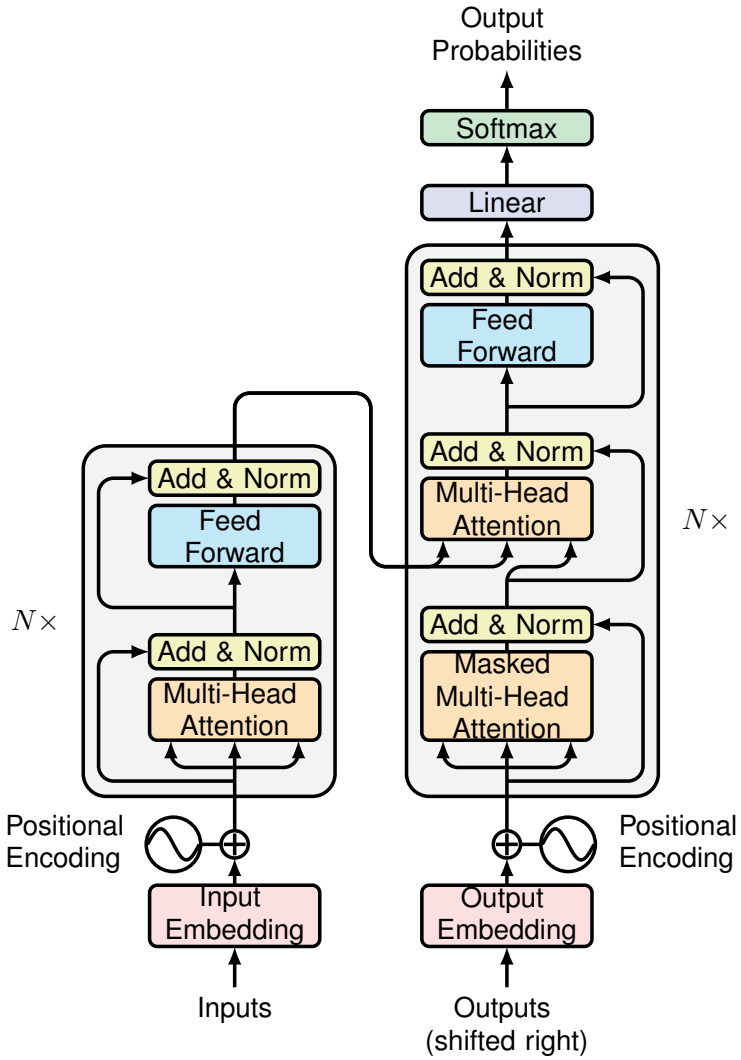


Fig. 2-4: The Transformer architecture, which illustrates the encoder-decoder structure, with both the encoder and the decoder being composed of multiple layers of self-attention and feed-forward networks [VAS17].

$$\text{Output for token } i = \sum_j \alpha_{ij} \mathbf{V}_j \qquad \text{Eq. 2-11}$$

The attention is computed in parallel using multiple attention heads, allowing the model to look at different parts of a sequence at the same time. Therefore, the transformer architecture consists of Multi-Head Attention blocks, that come at different locations throughout the process, as it can be seen in the original architecture of the transformer, shown in Fig. 2-4

2.2.5 Vision Transformer

The Vision Transformer (ViT) is a variation of the vanilla transformer introduced previously and it originally only consisted of the encoder structure. It was designed for computer vision tasks by treating input images as sequences of smaller image patches. Afterwards, all these newly split

patches are flattened into a one dimensional vector and a linear projection is applied to them, in order to map these patches into a higher dimensional space, resulting into so called patch embeddings. A second step of the transformer is to induce a general idea of positioning and spatial relationships between these patches. Therefore, a positional encoding mechanism provides the network with information about the position of every patch embedding from the original input image, which are then served as input to the transformer encoder. The transformer encoder, similarly to the one in the simple transformer, uses Multi-Head Attention blocks to calculate attention scores and to focus on multiple aspects of the image in parallel. Another feature of the ViT is the use of a classification token that carries information from all the patches and at the end of the transformer is put through a fully connected layer to make a final prediction about the input image [DOS20].

The revolutionary ViT architecture is the foundation of many other modern models, each designed for specialized computer vision tasks, from segmentation [STR21], to detection [LI22] or tracking [MEI22]. Therefore, two additional variation of the ViT, relevant to the work done in this thesis, are to be presented: the Detection Transformer (DETR) and the 3DETR (3D Detection Transformer).

DETR

DETR redefines the structure of the original ViT by combining the transformer with a CNN backbone in order to detect objects and making it a encoder-decoder type of architecture [ZHU20]. The CNN backbone is used for feature extraction, outputting a lower-resolution feature map. Before it is processed further by the encoder, positional information needs to be added. Afterwards, the transformer encoder receives the feature map, and as before, using Multi-Head Attention blocks, it allows the model to capture long range dependencies and further image features. The decoder receives a fixed number of object queries, each one being responsible for predicting one object in the image. The way this works is by utilizing a slightly different self-attention mechanism, called cross-attention, where the decoder is able to cross-attend to the encoded image features with its object queries in order to produce the predictions for the bounding boxes and the class labels. The result of this encoder-decoder structure is then put through two neural networks, one, a simple linear layer, tasked with the labeling of the objects, and the second, a feed-forward network, used to predict the 2D bounding box for each predicted object. The flow of the DETR architecture can be seen in Fig. 2-5, and as discussed, it consists of the backbone, positional encoding mechanism, transformer encoder and decoder and fully connected layers. One additional innovation belongs to the way this model learns, DETR introducing bipartite matching to assign predicted bounding boxes to the ground truths during the training process. The assignment step is done using the Hungarian algorithm [MIL07] based on the bipartite matching cost, which is a combination between the classification loss (cross-entropy) and the bounding box loss (L1 loss and generalized IoU) defined in Eq. 2-12.

$$\mathcal{L}_{\text{DETR}} = \sum_{i=1}^{N} \left[\mathcal{L}_{\text{class}}(y_{\hat{\sigma}(i)}, \hat{y}_i) + \mathbb{1}_{\{y_{\hat{\sigma}(i)} \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_{\hat{\sigma}(i)}, \hat{b}_i) \right] \qquad \text{Eq. 2-12}$$

where:

- N: The number of ground truth objects.
- $\hat{\sigma}(i)$: The index of the predicted object assigned to the i-th ground truth object.
- $y_{\hat{\sigma}(i)}$: The class label of the i-th ground truth object.
- $\hat{y}_i$: The predicted class label for the predicted object matched to the i-th ground truth object.
- $\mathcal{L}_{\text{class}}$: The classification loss (e.g., cross-entropy loss).
- $b_{\hat{\sigma}(i)}$: The bounding box coordinates of the i-th ground truth object.
- $\hat{b}_i$: The predicted bounding box coordinates matched to the i-th ground truth object.
- $\mathcal{L}_{\text{box}}$: The bounding box regression loss.
- $\mathbb{1}_{\{y_{\hat{\sigma}(i)} \neq \varnothing\}}$: An indicator function that equals 1 if the ground truth object exists, else 0.

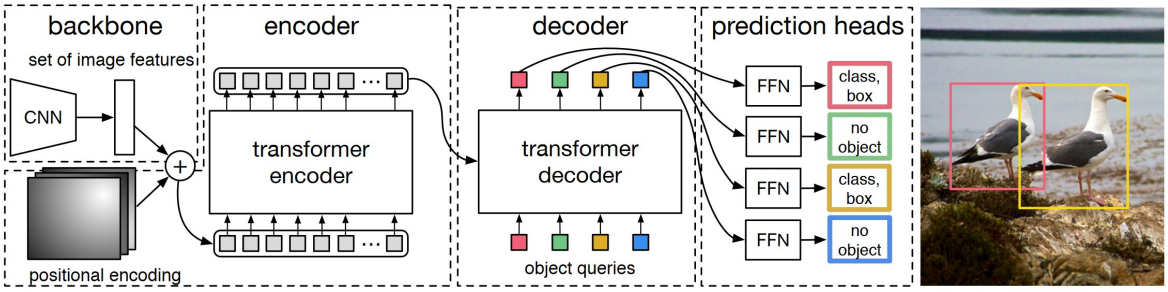


Fig. 2-5: Overview of the Detection Transformer.

3DETR

3DETR is an extension of the DETR architecture designed for the 3D space [MIS21]. It works directly on raw 3D point clouds, which are an unordered set of points, where each point contains three-dimensional XYZ coordinates. Before being input into the encoder, positional embeddings are added to the point cloud to retain spatial information, and additionally, the point cloud is preprocessed in order to extract its features. Then, the encoder-decoder structure is similar as in the DETR, with object queries being used to predict objects in the 3D scene. The prediction heads are tasked once again with class labeling and bounding box prediction, but in this case, the bounding box is determined by seven parameters, the center coordinates (x, y, z), the dimensions (width, height and depth) and the rotation angle of the predicted object. 3DETR uses the same loss function as DETR, slightly adapted for the 3rd additional dimension.

2.3 3D Object Detection Fundamentals

3D Object Detection is the computer vision task that aims to identify, locate and classify objects in a three dimensional space. Unlike the 2D case, where identified objects are parametrized through bounding boxes defined only by width and height, 3D Object Detection deals with trying to understand the complete spatial information of an object, adding the third dimension, the depth, into the

scene. The main importance of this third dimension is that is allows the system to identify the exact position of objects relative to the capturing sensor and their rotation in space [HE19].

As described in the previous chapter, 3D object detectors need to solve both classification and regression tasks. The classification task consists of labeling the detected 3D object, while the regression part needs to identify the parameters of a 3D bounding box, length, width and height, as well as its rotation in space. In 3D Object Detection, the Intersection over Union (IoU) is a crucial metric, used for evaluating the regression predictions. IoU is used to measure the accuracy of a prediction by calculating the overlap between the predicted bounding box of the object and its ground truth bounding box. If the IoU is 0 or a low value, it generally means that the prediction is not correct, while a high value signifies better accuracy. It is defined as following:

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} = \frac{A \cap B}{A \cup B},$$

where A and B represent the two bounding boxes. The IoU metric can be used for both 2D and 3D variants of bounding boxes. The following Fig. 2-6 shows the 2D case of three possible and common outcomes when using this metric. On the left, there is no overlap at all between the predicted and the ground truth bounding box, therefore the IoU value is 0. In the middle example, there is a small overlap, resulting in an IoU of around 0.23. Such a prediction is obviously better than the one in the first example, but still not as accurate as desired. The example on the right shows a very good, desirable match between the two bounding boxes, with an approximative IoU of 0.91.

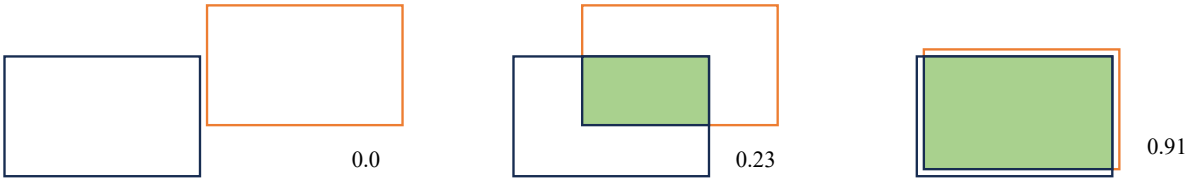


Fig. 2-6: Visualization of three IoU examples.

The scope of 3D Object Detection spans over various fields, but this thesis focuses solely on the autonomous driving sector. Therefore, usual detection examples of 3D objects are vehicles of different sizes, pedestrians, cyclists or other obstacles.

In a highly dynamic, real-time field as the autonomous driving, having object detection of high quality, meaning that detections are as fast as possible and the bounding box predictions as accurate as possible, is crucial in order to obtain general safety for all the involved parties. Unfortunately, doing 3D Object Detection comes with several challenges. During this procedure, a very common obstacle is occlusion, which appears when there are multiple objects overlapping each other in a scene, preventing their full view, the common example would be vehicles occluding pedestrians, cyclists or even other vehicles. Then, another challenge 3D object detectors face is the variation in shape and size of objects, which requires good generalization and model robustness to be overcome. Environmental conditions can also drastically affect the quality of 3D detections, since rain,

fog or high intensity lighting can interfere with the capabilities of the sensors. Moreover, additional training of a model on different kinds of weather can produce a big boost in the quality of detections. Furthermore, the sparsity of 3D data might also be problematic, especially when LiDAR or radar sensors are used, due to the limited number of points available that represent the surface of an object. Utilizing the best available hardware can minimize this issue. Therefore, overcoming such challenges require robust models, highly advanced algorithms as well as efficient and capable sensors to ensure high quality, real-time 3D Object Detection in a safety-critical environment as the autonomous driving [GRO23].

There are several techniques used in 3D Object Detection, each with strengths and trade-offs. The most common methods can be categorized into voxel-based, point-based and camera-based approaches. The voxel-based method processes a 3D point cloud into a structure voxel grid, a voxel being the 3D version of a pixel. By transforming the data in a structured manner, powerful 3D convolutions for feature extractions can be used but at the cost of fine detail loss. On the other hand, the point-based approach extracts features directly from the raw, unstructured 3D data point cloud using specifically designed neural networks, being able to preserve fine details but being prone to sparse data and computational complexity when it comes to large points clouds. The camera approach uses 2D RBG images from one or multiple cameras and processes them in order to induce a sense of depth in the detected objects.

2.4 Overview of Sensors: Radar, Camera, and LiDAR

In 3D Object Detection, the choice for the sensor plays an important role for the whole operation, determining the accuracy and reliability of the detection system. There are three different sensors that are going to be covered in this chapter, namely, LiDAR, radar and camera. All three have unique characteristics that help perform best under certain conditions, while still having drawbacks in some explicit cases [PAL22].

2.4.1 LiDAR

The LiDAR (Light Detection and Ranging) sensor works based on laser pulses emission and measures the time it takes for each pulse to return to the sensor after reaching an object. From this measurement, LiDAR sensors can generate highly accurate 3D point clouds that represent the surrounding area. Therefore, the main advantages of the LiDAR sensors are its high resolution, with great accuracy regarding distance and shape, the efficiency at medium ranges, the ability to provide 360° field of view of the surrounding area and also its effectiveness during low-light conditions or during nighttime. On the other hand, the disadvantages of using LiDAR are the degradation in performance in adverse weather conditions, quality of object classification, since it has a limited ability to detect colors or textures and lastly, performant LiDAR hardware is quite expensive compared to other sensor types. To exemplify the sensor positioning in the context of autonomous driving, a LiDAR sensor is usually put on top of the vehicle, in order to capture a complete field of view to be able to understand the surroundings [CAE20].

2.4.2 Radar

Radar stands for Radio Detection and Ranging. A radar sensor emits radio waves and detects the reflections that bounce back from objects, similarly to how a LiDAR sensor operates. However, unlike LiDAR, which uses laser pulses, radar uses electromagnetic waves at radio frequencies, typically in the millimeter-wave spectrum. After emitting the waves, the sensor calculates the time delay between the emission and the return of the signal and measures the distance to the detected object as following:

$$d = \frac{c \cdot t_d}{2}, \qquad \text{Eq. 2-13}$$

where c is the speed of light and t_d the time delay.

The radar technology brings important advantages, especially in the context of autonomous driving. Radar sensors excel in range capabilities, being able to detect objects at distances of up to several hundred meters. This long-range detection is further complemented by the sensor's robustness in different adverse weather conditions, such as fog, rain or snow, where the visibility for sensors such as LiDAR and camera can be severely impaired but also in low-light conditions, being extremely useful also for nocturne scenarios. Moreover, another feature is the Radar Cross Section (RCS), which is a measure to check how detectable an object is, by quantifying the amount of radar energy it scatters back to the receiver the moment it illuminates this object. Furthermore, another key benefit of the radar is its ability to directly measure the velocity of detected objects through the Doppler shift, offering valuable information about the speed and trajectory of nearby vehicles, pedestrians or other moving instances relative to the sensor. The velocity of a detected object can be calculated using the Doppler effect with the following formula

$$v = \frac{\Delta f \cdot \lambda}{2} \qquad \text{Eq. 2-14}$$

with Δf being the measured Doppler shift and λ the wavelength of the emitted radar signal.

However, radar sensor also have significant drawbacks that impact their full utility for 3D Object Detection problems. Its biggest limitation is the relative low spatial resolution, especially when comparing it to LiDAR. Due to the longer wavelength of radio waves, radars struggle to capture fine detail, making it challenging to distinguish between closely placed objects or accurately identify the shape and contours of detected objects. Therefore, such limited resolution can often lead to higher uncertainties in a model's predictions on the size and orientation of an object, which directly affect the performance of classification or regression tasks. Additionally, radar sensors are prone to interference, from both environmental factors and from other radar systems, especially in urban locations. This inference can thus degrade the quality of the radar signal, leading to potential misdetections or with a further increase in uncertainty.

Overall, in the context of autonomous driving, radars are highly robust and reliable sensors in any kind of conditions, but, compared to LiDAR, they are struggling to be as efficient for detection purposes due to the sparse resolution of their point clouds. Nevertheless, in the following chapter,

a more modern type of radar is to be introduced, together with an idea of increasing the spatial resolution of this sensor type.

2.4.3 Camera

The last covered type of sensors is the camera. Cameras function by capturing images using light that reflects off objects, providing detailed visual information about the surrounding environment. In order to perform 3D Object Detection using cameras, these sensors are usually used in conjunction with additional deep learning models to infer a sense of depth from the 2D image data. The advantages that cameras bring are the rich visual information with high resolution, that can ease the process of object classification and cost-effectiveness, since cameras are relatively inexpensive. When it comes to 3D scenarios, the downside of this sensor type is that it has a poor depth perception, which requires additional resources to be overcome. Furthermore, cameras are also prone to adverse weather like fog, rain or snow, as well to poor lightning conditions. To the positioning of cameras on a vehicle, these are usually mounted all around it, with a focus on the front side of the car [CAE20].

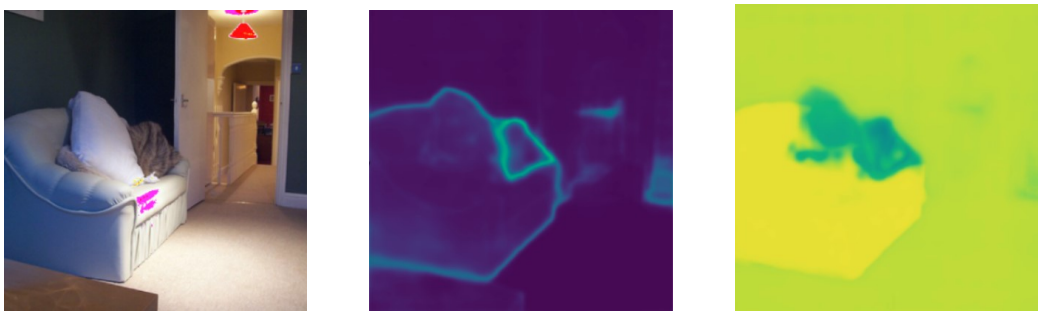
Table 2-7 shows a full comparison of the three sensors detailed above.

Criteria	**Radar**	**LiDAR**	**Camera**
Range	Long-range	Medium to long-range	Short to medium
Resolution	Low	High	High
Adverse Weather Performance	Excellent	Poor	Poor
Velocity	Yes	No	No
Cost	Moderate	High	Low
Data Output	1D or sparse 3D data	Dense 3D point clouds	2D images, with possible 3D inference
Color/Texture	No	No	Yes

Fig. 2-7: Comparison of Radar, LiDAR, and Camera sensors.

2.5 Uncertainty in Machine Learning

To define this term, uncertainty refers to the lack of confidence in the predictions of a model and quantifies how much a model knows about its predictions. Uncertainty is usually categorized into two main types that are **epistemic** and **aleatoric** uncertainty. The epistemic uncertainty, which is a model-related uncertainty, arises from the limitations of the model itself. Therefore, this type represents the uncertainty that comes from insufficient training data and can be reduced by improving the model or increasing the diversity of the dataset. To exemplify, in the case of 3D Object Detection, epistemic uncertainty would arise when the model is not sufficiently trained on an object type that it needs to classify. The second type is aleatoric uncertainty, which is data-related, and originates directly from the noise in the data itself. It can be a manifestation of the sensors during data acquisition



(i) Original image (ii) Aleatoric Uncertainty (iii) Epistemic Uncertainty

Fig. 2-8: Overview of the two types of uncertainty [MUK21].

or due to occlusion, when object are only partially visible and cannot be easily reduced, since this uncertainty is introduced during data gathering [MUN24] [GRO23]. Fig. 2-8 displays the differences between the aleatoric and epistemic uncertainties. In this image, the aleatoric uncertainty is seen close to the edges of objects, where there is higher ambiguity, while epistemic uncertainty arises on regions previously unseen by the model during training.

2.6 Baseline Methods for Uncertainty Estimation

In this chapter, three methods used for uncertainty quantification are to be presented. These quantification methods aim to align the predicted uncertainty of a model with its actual error rates, which is extremely useful in safety-critical fields like 3D Object Detection for autonomous driving, where reliable predictions are needed to improve decision-making. The following techniques will form the baseline for uncertainty quantification of the future experiments.

Deep Ensembles

Deep Ensembles (DE) [LAK17] is a popular method of improving the overall model robustness and uncertainty estimation. It relies on training multiple independent models and fusing their outputs or predictions into a single one, using a merging technique, as depicted in Fig. 2-9. It is important, that the models are from the same architecture and initialized and trained with different seeds in order to be independent from each other. The principle of DE can be formulated as following:

$$P(y \mid D) = \frac{1}{N} \sum_{n=1}^{N} P_{\phi_n}(y \mid D, \phi_n) \qquad \text{Eq. 2-15}$$

where N is the number of ensembles, ϕ_n refers to the model n set of weights and D is the datapoint used for inference.

Deep Sub Ensembles

Deep Sub Ensembles (DSE) [VAL19] is another baseline technique used in this thesis. It is a variation that bases on the concept of deep ensembles with a specific focus of reducing the extreme computational complexity needed while still maintaining their properties. Instead of training

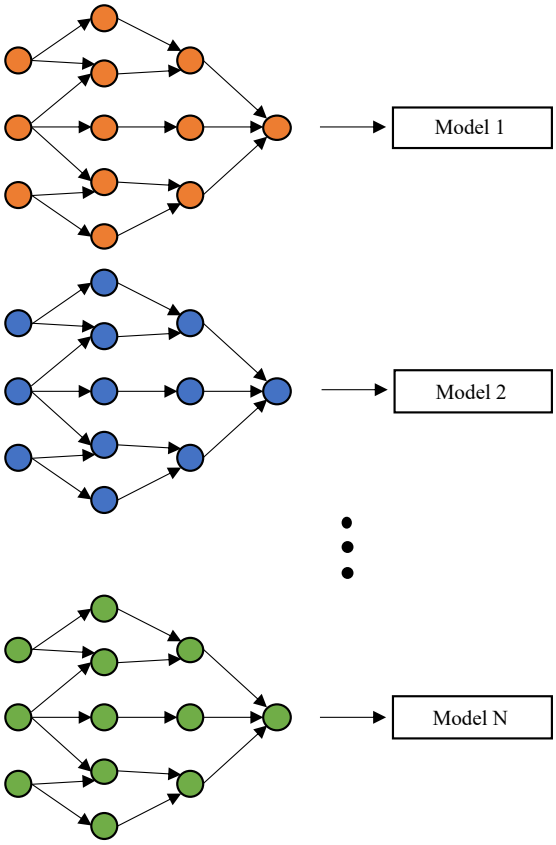


Fig. 2-9: The general idea of Deep Ensembles, where up to N models are independently trained so that their output is combined in order to lower the uncertainty of the final model.

full independent models with different initialization as deep ensembles do, DSE divide the model in sub-parts and only train one or some of these sub-parts of multiple models. This diversity in these sub ensembles leads to different predictions from the same input, thus, merging the outputs of the ensembles provides a more robust final prediction and an estimate for the uncertainty [PIT22].

Monte Carlo Dropout

Monte Carlo Dropout (MCD) [GAL16] is the third and last baseline technique of estimating uncertainty in deep learning models, which introduces randomness during inference. MCD leverages the already existing regularization technique of dropout, which is originally applied to prevent overfitting by randomly dropping out a portion of the nodes in the network during each forward pass, therefore resulting to a better generalization to unseen data, as shown in Fig. 2-10. However, dropout is only enabled during the training procedure and disabled during inference to ensure stable predictions. MCD, on the other hand, keeps dropout enabled also during inference and introduces multiple forward passes sequentially for the same batch. Therefore, this method transforms the deterministic model into a stochastic model. By running multiple forward passes, MCD provides a distribution of outputs, which can be used to estimate both the mean predictions and the associated uncertainties. In order to do so, MCD requires an additional merging step, that takes the multiple outputs and merges them into a final output of predictions [PIT22]. MCD can be formulated with the following

equation:

$$P(y \mid D) = \frac{1}{N} \sum_{i=1}^{N} P(y \mid D, \phi_{\text{dropout}}) \quad \text{Eq. 2-16}$$

where N is the number of forward passes, ϕ_{dropout} refers to the model's weights, and D is the inference datapoint.

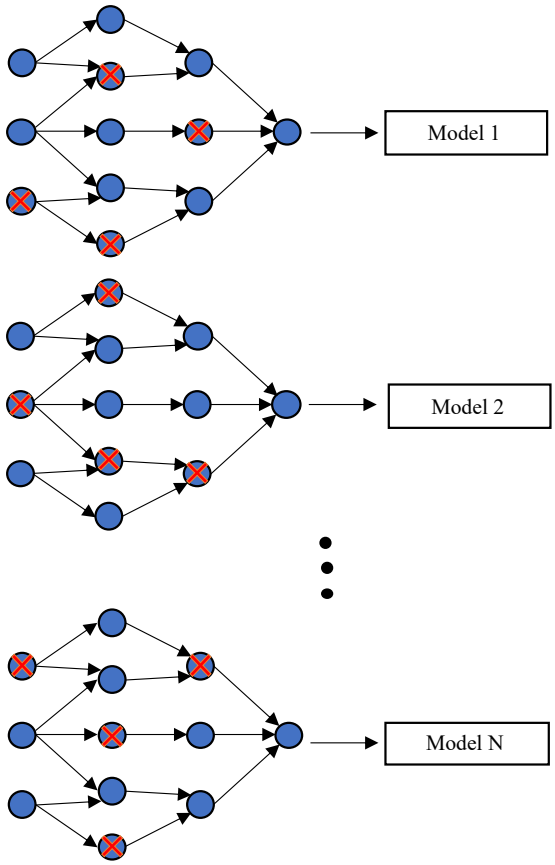


Fig. 2-10: Visualization of the the Monte-Carlo Dropout technique, where dropout is enabled during up to N forward passes, each time cancelling random nodes in the layers.

Merging techniques

In order to combine multiple predictions from MCD, DE or DSE, merging techniques are required to achieve a final merged output. Following existing literature on bounding box clustering for object detection, the most straight forward method of merging multiple outputs is to gather these over multiple forward passes or (sub-) ensembles and to perform an averaging operation on the score and the bounding box parameters for each input at a time. For the case of deciding the final predicted label, a majority voting system is implemented for making a decision, which can also be scaled by the corresponding scores. Another possible implementation is adding the IoU (Intersection over Union) values of the bounding boxes before averaging the predictions [REZ19]. The addition of IoU ensures that only bounding boxes that are predicted closely to each other are going to be averaged together.

Furthermore, another implementation is to include a pre-clustering in this process, based on the Euclidean Distance between center-points of bounding boxes. Then, the final merged bounding boxes are calculated by averaging the boxes in the obtained clusters [DEN20].

2.7 Evaluation Metrics and Plots for Uncertainty Estimates

This chapter delves into the evaluation metrics used to assess uncertainty estimates, which can quantify how well a model's uncertainty estimates align with the predicted performance.

2.7.1 Classification Uncertainty

Calibration metrics help quantify the alignment between the model's predicted probabilities and the true likelihood of the predictions being correct. As an example, when a model predicts an object with 80% confidence, then it should also be right 80% of the time. Following, several classification metrics for uncertainty quantification are to be introduced, such as Negative Log-Likelihood for classification, Brier Score, Expected Calibration Error and Detection Expected Calibration Error.

Classification Negative Log-Likelihood

The Negative Log-Likelihood (NLL) is a very popular metric that evaluates uncertainty estimates for probabilistic models [GUO17]. NLL measures how well a model's predicted probability distribution matches the actual outcomes by calculating the negation of the log-likelihood of true labels, given the predicted probabilities. To understand NLL, lower values reflect better model calibration and increased confidence, while higher values indicate that the model is being too overconfident on incorrect predictions or too uncertain about correct predictions. The formula of classification NLL is

$$\text{NLL} = -\sum_{i=1}^{N} \log(p(y_i|x_i)), \qquad \text{Eq. 2-17}$$

with N being the number of data samples, while $p(y_i|x_i)$ is the predicted probability of the true label y_i given input x_i.

Brier Score

Another metric that assesses the accuracy of probabilistic predictions is the Brier Score. This metric quantifies the mean squared difference between predicted probabilities and the true outcomes. Once again, low values of the Brier Score are desired, since these indicate good calibration.

$$\text{Brier Score} = \frac{1}{N}\sum_{i=1}^{N}(p_i - y_i)^2, \qquad \text{Eq. 2-18}$$

shows the formula of the Brier Score, with N being the number of predictions, p_i the i-th predicted probability and y_i the true outcome.

Expected Calibration Error

Expected Calibration Error (ECE) [MUN24] measures the gap between the predicted confidence and the actual accuracy. The main idea is to group the predictions into bins based on their predicted confidence levels and to compare the accuracy within each bin to the average confidence of the predictions from that same bin. A perfect calibrated model would have the same predicted confidence and accuracy in each bin, leading to an ECE of 0. The greater the value of the ECE formula is, the higher the miscalibration. It reads,

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} \left|\text{acc}(B_m) - \text{conf}(B_m)\right|, \qquad \text{Eq. 2-19}$$

where M represents the number of bins, B_m the set of samples in m-th bin and n the total number of samples. The accuracy states the proportion of correctly predicted samples in the current bin, while the confidence is the average predicted probability.

Detection Expected Calibration Error

D-ECE [MUN24] is varying from the ECE by including multi-class classification scenarios, when evaluating how well the model's predictions are calibrated for detecting a specific class of interest, stated as following:

$$\text{D-ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} \left|\text{precision}(B_m) - \text{conf}(B_m)\right| \qquad \text{Eq. 2-20}$$

This differences from the ECE by calculating the precision of the samples in the m-th bin, instead of their confidence. Precision gives the proportion of true positive predictions out of all detections made by the model. Once again, lower values of D-ECE translate to a better calibration, while higher values are signs of model miscalibration.

Reliability Diagram Plot

The reliability diagram is a very useful visual tool, complementary to the ECE, for visualizing the calibration of probabilistic predictions made by a model. This diagram shows the relationship between accuracy, usually depicted on the y-axis, and confidence on the x-axis [BRÖ07]. In well calibrated models, the points on the reliability plot will lie close to a 45 degree diagonal line, which would indicate that if a model predicts a detection with a certain probability, then the actual proportion of positive outcomes should have the same probability. Furthermore, if the points lie above the diagonal line, then the model behaves under-confidently, while conversely, if the points are under the diagonal, then it indicates overconfidence. Fig. 2-11 shows two opposing reliability diagrams. The image on top presents a poorly calibrated model that behaves overconfidently, while underneath it, there is the reliability plot of a well calibrated model, since the confidence of the predictions matches almost perfectly the accuracy.

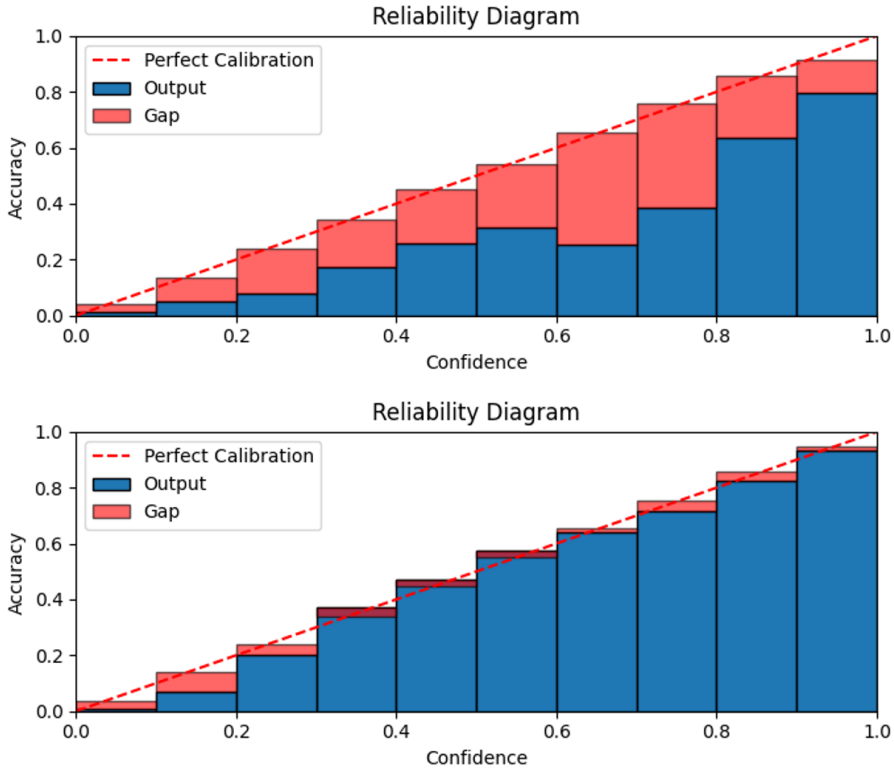


Fig. 2-11: Reliability diagrams showcasing two different uncertainty estimation scenarios. On top there is an example of an overconfident model, while beneath it, there is an example of a well calibrated model.

2.7.2 Regression Uncertainty

Mahalanobis Distance

Mahalanobis Distance (MD) is a measure of the distance between a point and a distribution, but unlike the Euclidian, the Mahalanobis distance scales the distance based on the variance of each dimension [MCL99]. The Mahalanobis distance is calculated as

$$D_M(\mathbf{y_{gt}}, \mathbf{x_{pred}}) = \sqrt{(\mathbf{y_{gt}} - \mathbf{x_{pred}})^T \mathbf{S}^{-1} (\mathbf{y_{gt}} - \mathbf{x_{pred}})}$$

with $\mathbf{y_{gt}}$ being the ground truth vector, $\mathbf{x_{pred}}$ the mean predicted vector and $\mathbf{S}$ the covariance matrix.

Regression Negative Log-Likelihood

Regression NLL metric estimates how well the model's predicted probability distribution matches with the observed data, while also quantifying uncertainty through the measured covariance matrix.

$$f(\mathbf{y_{gt,i}}) = \frac{1}{\sqrt{(2\pi)^D |\mathbf{S_i}^{-1}|}} \exp\left(-\frac{1}{2}(\mathbf{y_{gt,i}} - \mathbf{x_{pred,i}})^T \mathbf{S_i}^{-1} (\mathbf{y_{gt,i}} - \mathbf{x_{pred,i}})\right)$$

$$\text{Mean NLL} = \frac{1}{n}\sum_{i=1}^{n} -\log\left(f(\mathbf{y_{gt,i}})\right)$$

where $\mathbf{y_{gt,i}}$ is the ground truth vector for sample i, $\mathbf{x_{pred,i}}$ is the mean predicted vector of i, $\mathbf{S}$ the predicted covariance matrix and D the dimensionality.

Average Calibration Plot

A great visualization tool for assessing uncertainty is the average calibration plot [CHU21], which has a similar idea as the reliability diagram but works on the regression output. This plot describes how well the model's predicted intervals match the true outcomes. For any interval α, an $\alpha-$prediction interval should capture the true values $\alpha\%$ of the time. The x-axis represents the predicted proportion of data that is expected to be within the interval and the y-axis shows the actual observed proportion.

A perfectly calibrated model would produce a curve that perfectly aligns with the diagonal ($y = x$). Area between the diagonal and the curve expresses the level of miscalibration. If the curve is below the diagonal, the model shows overconfidence, while in the opposite case it would be underconfident. Fig. 2-12 shows a such example, where the model becomes progressively more overconfident with higher α.

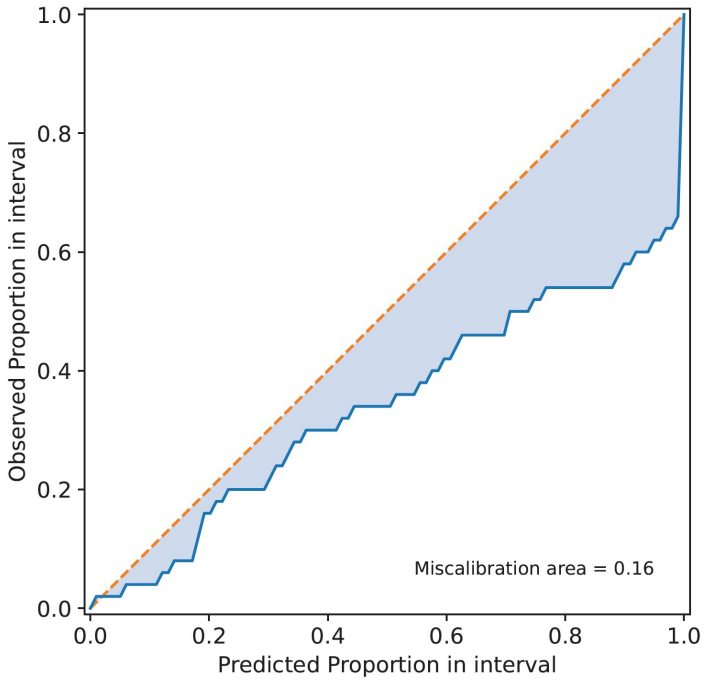


Fig. 2-12: Example of the average calibration plot that shows overconfidence with higher prediction intervals.

Residual Uncertainty Plot

The residual uncertainty plot [SIC21] is a scatter plot that visualizes the connection between residual errors ($y_i - \mu_i$) and their corresponding prediction uncertainty. For a multivariate case, the x-axis would correspond with the difference between the ground truth and the predicted bounding box,

while the y-axis would be defined by the square root of the trace of the covariance matrix. For a hypothetically perfectly calibrated model, the residual error should match the normal distribution, meaning that 68,3% of points should be inside the 1σ region (orange lines), while 99.7% should be within 3σ (blue lines). Fig. 2-13 shows a hypothetical perfect calibration example.

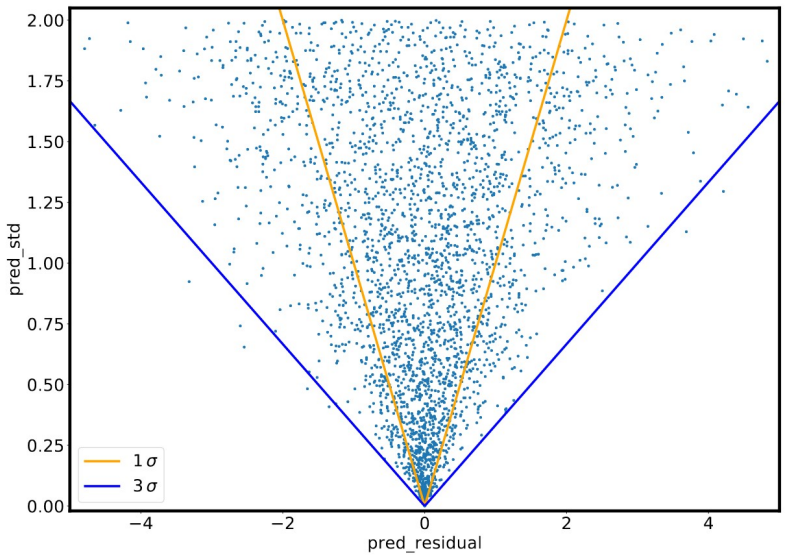


Fig. 2-13: A residual uncertainty plot of a perfectly calibrated model, where 68.3% of data falls within 1σ and 99.7% within 3σ.

3 Related Work

In this chapter, an overview of the existing research related to this thesis' topic is going to be provided, focusing on some key aspects of 3D Object Detection, in order to highlight the strengths and limitations of the current approaches and to identify gaps in the literature. The structure consists of a review of the relevant datasets for this work, followed by a description of two state of the art transformer-based architectures. Afterwards, the current state and the research gaps of the field of uncertainty estimation for object detection are assessed to put the work of this thesis into perspective.

3.1 Datasets

Large datasets are an essential element for training an AI model. In the context of 3D Object Detection, with a focus on autonomous driving, there are several state of the art datasets designed for training DL models. The work of thesis was conceived working on the NuScenes and View-of-Delft datasets.

3.1.1 NuScenes

NuScenes is a large multi-modal, publicly available dataset, primarily designed for the field of autonomous driving [CAE20]. It offers data acquired through LiDAR, Radar and Camera sensors, it was captured in Boston and Singapore and consists of 1000 driving scenes, 20 seconds long, captured at 2 HZ. The dataset has a 70-15-15 split, resulting in around 28000 training and 6000 testing and validation samples, covering 23 different classes and offers annotations for the training and validation set. The NuScenes dataset implements a new metric for evaluating the performance of models, called NuScenes Detection Score (NDS), which combines the mean average precision and mean true precision metrics. The layout of the sensors is as following: in the case of LiDAR, a single spinning 32-beam sensor is mounted on top, for the cameras, there are six HD, RGB pieces, positioned in such a way to provide a 360° field of view around the car. Lastly, there are five longer range radars used in total, one on the frontal bumper and the rest on every corner of the vehicle, capable of accurately estimating the velocity of nearby objects. Figure 3-1 shows the exact position and the coordinate system of each sensor.

3.1.2 View of Delft

View of Delft (VoD) is a newly available dataset, multi-modal and of medium size created for object detection in the autonomous driving field and recorded in Delft, Netherlands [PAL22]. It consists of data gathered through Camera, LiDAR and radar sensors. In the case of the radar data, the VoD dataset consists of three distinct radar sets, gathered accumulating 1, 3 and 5 successive scans. Compared to the NuScenes dataset, VoD has a 64-beam LiDAR sensor, which can produce denser point clouds. Moreover, VoD uses a ZF FRGen 21 "3+1D" radar instead of a "2+1D", as in the case of NuScenes, which is a high-performance, mid-range sensor that can additionally measure the elevation of the objects, besides the range, azimuth and velocity, providing overall three spatial and one velocity dimension. This new spatial dimensional feature allows to produce better radar data especially for tasks like 3D Object Detection, where density of point clouds is key. The radar data of

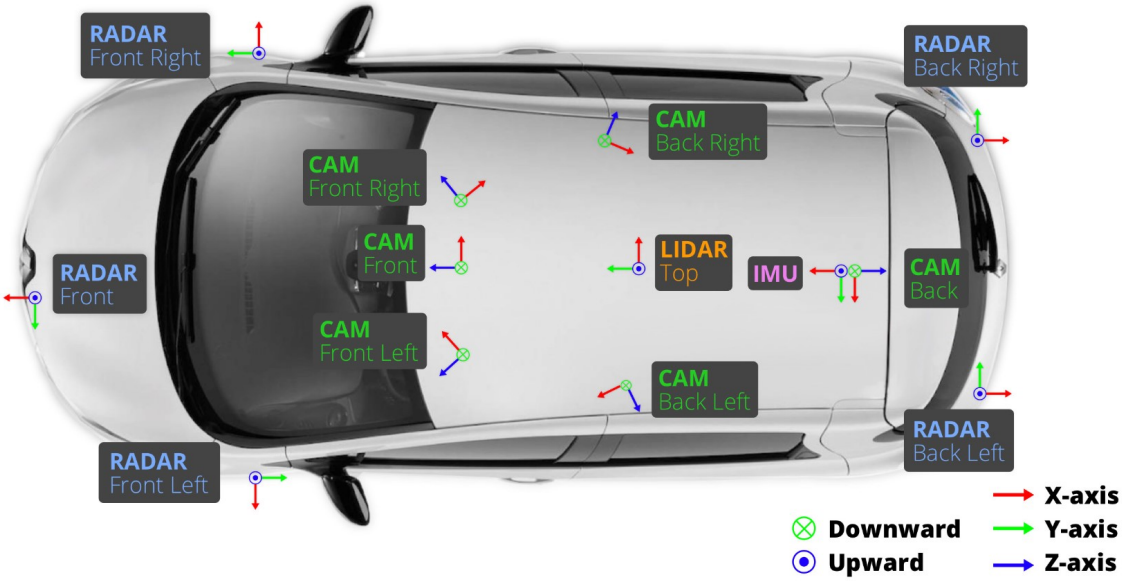


Fig. 3-1: NuScenes sensor layout of the five radars, six cameras and one LiDAR together with their corresponding coordinate system.

VoD can be visualized in Fig. 3-2 as colored dots, which shows that by including the third dimension, elevation, more descriptive information is being obtained from the objects ahead. Additionally, by accumulating multiple consecutive radar scans, the VoD authors manage to create much denser radar point clouds and provide additional temporal information that should help achieve better classification results. There is only one of such radars available on the car and is mounted behind the front bumper of the car. Moreover, VoD also has one stereo camera installed in the middle of the windshield, but the camera data from this dataset is not used in this thesis. The sensors' location, coordinate system and model are shown in Fig. 3-3.

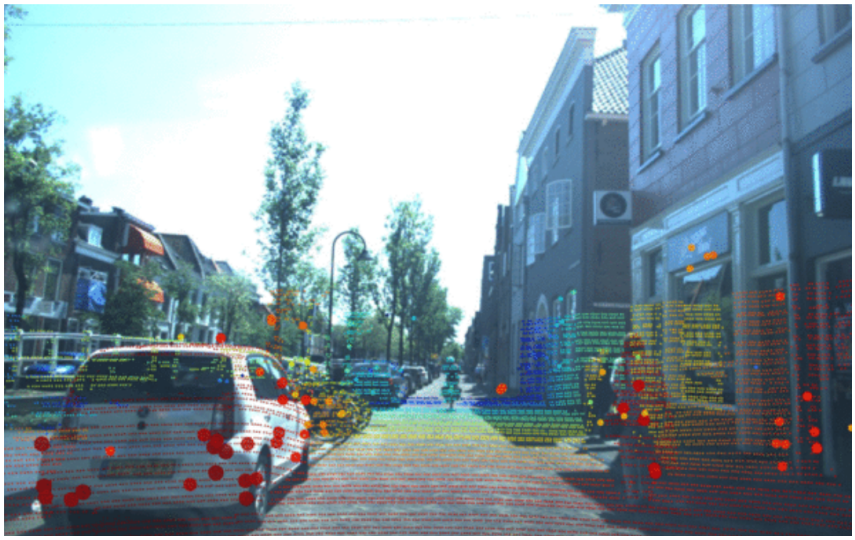


Fig. 3-2: Visualization of the Radar (colored dots) and LiDAR (colored sweeps) point clouds of the VoD dataset.

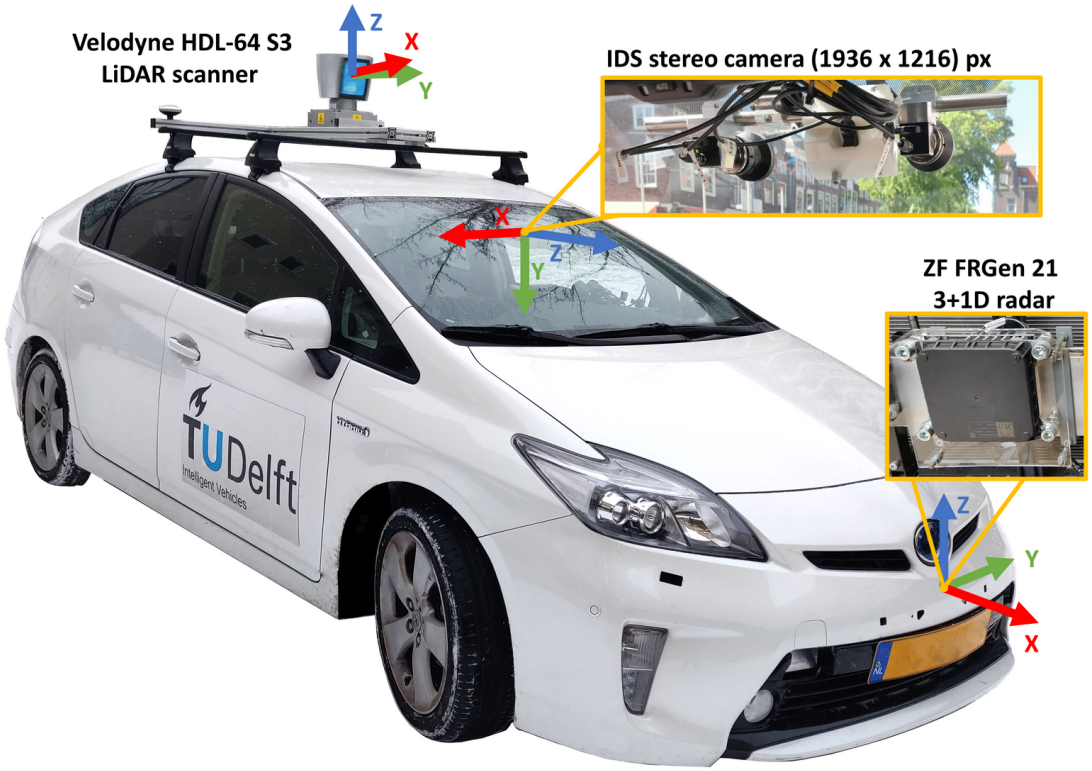


Fig. 3-3: VoD sensor layout, sensor models and their corresponding coordinate system.

3.2 Transformer-based Approaches in 3D Object Detection

Transformer-based models are modern deep learning approaches that make use of the transformer architecture. Recent literature on transformer-based 3D Object Detection shows significant advancements, especially in autonomous driving applications, due to their ability to effectively model long-range dependencies and integrate both local and global context. Some approaches, called hybrid architectures, integrate CNNs as backbones for feature extraction and then use the obtained features as input in the transformer architecture, to make use of the attention mechanism. Others designed end-to-end transformer models to simplify the process and to reduce the need for additional feature engineering. Such architectures can usually process directly raw input data without always needing an intermediate representations.

3.2.1 Position Embedding Transformation

Position Embedding Transformation (PETR) is a state of the art architecture derived from DETR and designed for multi-view 3D Object Detection [LIU22]. The innovation that PETR brings is a direct encoding of 3D position embeddings into the extracted 2D image features, thus allowing it to understand the position of objects, as well as the depth of a scene. This is pictured in Fig. 3-4, where the position embedded features serve as input for the encoder. The decoder side of the architecture is equivalent to the one used in DETR. Therefore, PETR does not necessarily need to rely on LiDAR or other depth sensor and can make direct use of camera data. The multi-view concept of PETR allows this architecture to work with views from multiple cameras, being able to cover a 360° view around.

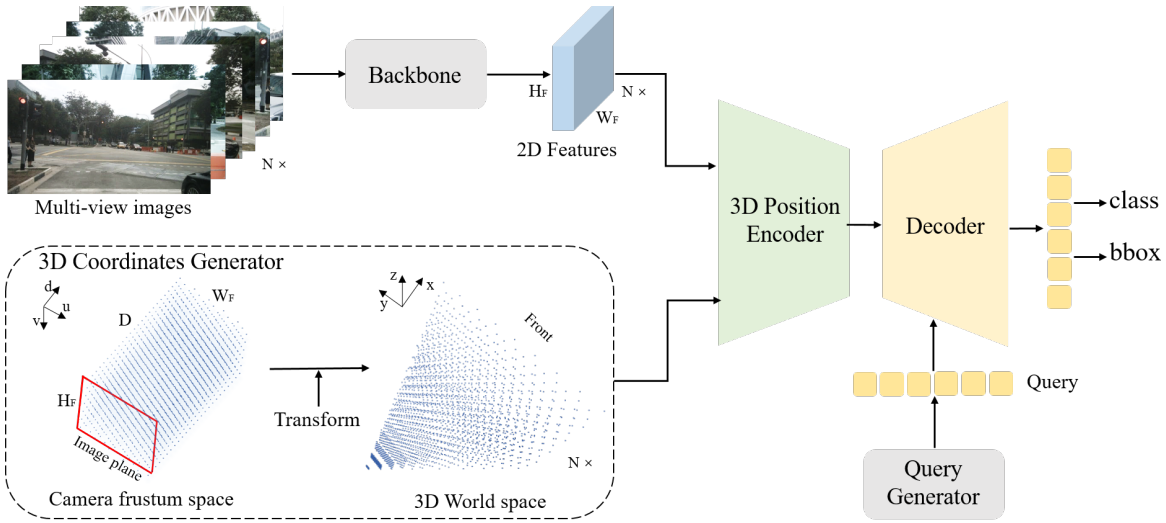


Fig. 3-4: Overview of the complete PETR architecture. Extracted 2D features with 3D coordinates are input into the encoder to obtain 3D positional-aware features, which are then used together with generated queries in the decoder for predicting the output [LIU22].

3.2.2 PillarFormer

The PillarFormer is a transformer decoder-only architecture, designed for handling LiDAR and radar data. First of all, the point clouds are preprocessed into voxels, then using a voxel encoder, features are extracted. The next step consists of using a middle, sparse encoder to further process the voxel features using sparse convolutions. This step assures with the reduction of computational overhead, since it only focuses on non-empty regions of the voxel grid. Furthermore, a SECOND (Sparsely Embedded Convolutional Detection) backbone takes the encoded voxel features from the previous step and processes them to extract high level feature maps. These extracted feature maps of the sparse encoder are then fed into a feature pyramid network (FPN) component, which creates a multi-level feature representation that helps to detect object of various sizes more effectively. Finally, on the decoder side, the architecture head is responsible with generating the final predictions, consisting of bounding boxes, class labels and class scores. The PillarFormer decoder head is inspired by the one used in the PETR architecture.

3.3 Uncertainty Estimation in Object Detection

Lately, there has been a significant amount of research towards the quantification and evaluation of uncertainty in object detection tasks. The authors in [GRO23] perform ample experiments on DETR, hypothesizing if deep sub-ensembles can help such architecture generalize better to unseen data and output more robust predictions. This research only focused on the 2D variant of object detection and showed that utilizing deep sub-ensembles of the DETR architecture does not consistently result in an increase in robustness.

Moreover, [MUN24] performs further analysis on uncertainty estimates, focusing on further extensions of the DETR architecture. Firstly, [MUN24] develops an uncertainty-guided logits modulation mechanism to adjust the class logits based on their uncertainty. Secondly, they introduce a logit mixing method that behaves as a regularizes and can be used independently and along with the

logits modulation to increase the calibration improvement. They assess the calibration values by calculating the ECE and D-ECE accordingly and compare these results to several baselines. Their results show a good calibration increase using their proposed approaches but only research the 2D realm of the transformer-based object detection.

Another work, [PIT22], developed an efficient way to estimate uncertainty for the case of LiDAR-based 3D Object Detection. They built an adaptation of the multi-input multi-output uncertainty estimation method [HAV20] for the 3D case using LiDAR data and showed promising performance compared to the ensemble and MC dropout methods, while requiring significant less computation resources. Although the authors looked into the 3D world of object detection, more performant transformer-based architectures can be used. Additionally, investigating the camera and radar data besides the LiDAR can also offer interesting result. One important aspect of the ensembles and MC dropout baselines is the chosen merging algorithm. The authors in [PIT22] used a mean merging method taken from [ROZ20], which compares and summarizes several merging methods, including max, mean, variance voting and weighted boxes fusion (WBF) [SOL21].

4 Approach and Implementation

There have been several recent papers regarding uncertainty estimation for object detection with a major focus on the 2D space. There is a notable lack of comprehensive studies analyzing the 3D version of uncertainty estimation specific for transformer-based object detection. Therefore, this thesis aims to investigate the uncertainty estimates of two state of the art 3D capable transformer-based architectures. First of all, the goal is to implement and train several baselines, Deep Ensembles. Deep Sub-Ensembles and Monte-Carlo Dropout, for comparison. Secondly, the proposals of [MUN24] are to be integrated into the analyzed architectures to check if, in the case of 3D Object Detection, the models can achieve better calibration. Moreover, two further post-hoc calibration techniques, Temperature Scaling and Depth Scaling, are to be implemented. Lastly, an in depth analysis of the results from the transformer-based 3D object detectors based on the UQ metrics presented previously is to be made.

In this chapter, the methodology and implementation details of uncertainty estimates for transformer-based 3D object detectors are presented. Firstly, the framework approach of this thesis is described. Secondly, the implementation of uncertainty calibration methods is presented. Moreover, an overview of the 3D clustering techniques designed for merging the results of the baseline models is shown. Finally, the chapter ends with the implementation flow of the experiments done in this thesis.

4.1 Calibration Mechanisms

In this thesis, several calibration methods for transformer-based 3D object detectors are evaluated. These include techniques for calibration of classification, such as Temperature Scaling, Logits Modulation, and Logits Mixing, and a method for regression output calibration, Depth Scaling. The goal is to reduce the model's uncertainty while maintaining lower computational costs compared to traditional baseline approaches. Additionally, these approaches can also be classified into loss-oriented techniques (Logits Mixing and Modulation) and post-hoc scaling techniques (Temperature and Depth Scaling). Fig. 4-1 illustrates the general approach of this thesis. The main idea is that, on the decoder side, both the classification and regression outputs are modelled using the above mentioned methods to improve the overall calibration of the model, so that, in the end, it can be compared to the designed baselines to see if the hypothesis of having better calibrated models with lower computational resources is true. Another important mention is that all these calibration mechanisms are independent from the input modality, meaning that they can be identically applied for camera, radar or LiDAR input data.

4.1.1 Logits Modulation

Logits modulation [MUN24] is an uncertainty-guided modulation mechanism of the calculated logits that works by modulating the uncertainty in the class scores on the decoder side of the transformer.

First of all, let $O_D \in \mathbb{R}^{L \times B \times Q \times C}$ be the output of all decoder layers, where L is the number of decoder layers, B is the batch size, Q is the query size and C is the number of classes. In or-

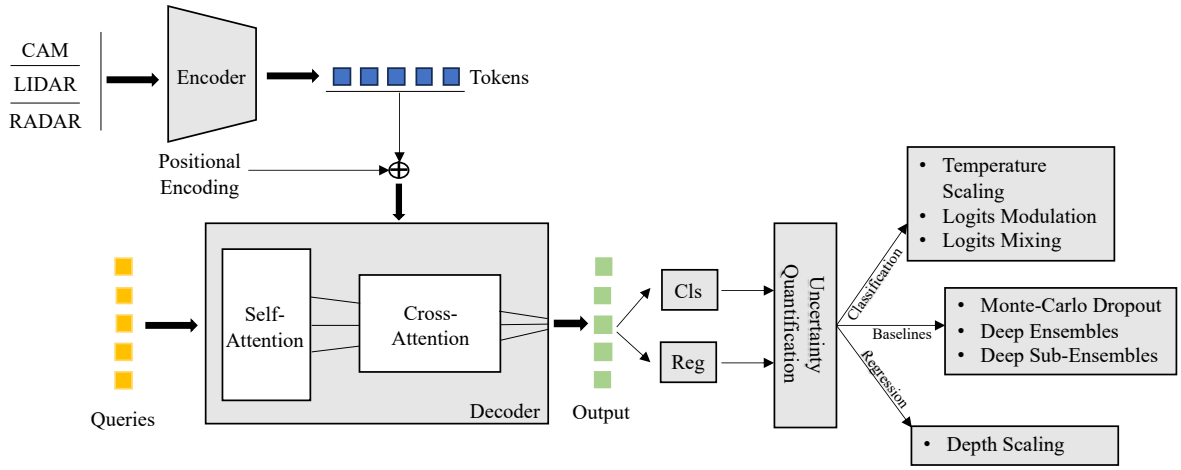


Fig. 4-1: The approach setup used in this thesis. The main focus lies on the decoder of the architectures, which uses generated queries as well as extracted features with positional information for its attention mechanisms in order to output classification and regression predictions. In the case of the loss oriented techniques, the predictions are already being modeled during training, while the post-hoc techniques are applied after the result has been computed. Afterwards, the processed classification and regression outputs are being evaluated and compared to the baselines.

der to quantify the uncertainty to modulate the logits, one needs to calculate the variance along the $L-th$ dimension of all the decoder layer outputs. Computing this variance shall results in a tensor $u \in \mathbb{R}^{B\times Q\times C}$, which gives the uncertainty of all class logits across the batch and query dimensions.

Secondly, the uncertainty vector u needs to scaled down from the $[0, \inf]$ range to $[0, 1]$ using the $\tanh$ operation and transformed from uncertainty into certainty by taking $1 - \tanh(u)$, which is then used to modulate the logits of the final decoder layer using the formula:

$$\tilde{O}_D^L = O_D^L \otimes (1 - \tanh(\mathbf{u})), \qquad \text{Eq. 4-1}$$

where $O_D^L \in \mathbb{R}^{B\times Q\times C}$ is the final decoder layer and $\tilde{O}_D^L$ is the new logits modulated output of the final decoder layer. By multiplying with $1-\tanh(u)$, this approach scales down class scores that have a higher uncertainty, which should then improve the calibration of the model.

4.1.2 Logits Mixing

The second mechanism, logits mixing [MUN24], behaves as a regularizer with its own detection-specific loss function and can be used in combination with the logits modulation approach, to further improve the confidence calibration of the models.

The first step of logits mixing is identifying the positive queries from the logit space, which is of dimensions $B \times Q \times C$ with B being the number of batches, Q the number of queries and C the number of classes. Positive queries are queries that are correctly assigned to a class dur-

ing the matching process and these are assumingly directly responsible for detecting objects in a frame.

The second step consists of calculating the mean over all these positive queries, which is then used to compute the logits mixing for each individual positive query. Let $\tilde{Q}$ be the mean of all positive queries from all B batches and Q_i be a positive query, then the formula for computing the logits mixing can be expressed as:

$$\acute{Q}_i = \alpha Q_i + (1 - \alpha)\tilde{Q}, \qquad \text{Eq. 4-2}$$

where $\acute{Q}_i$ represents the newly modified positive query that came from Q_i and α is the logits mixing weight. Finding a proper α value is important, since at 1, the mixed query is equal to the old positive query, meaning that there is will be no change in calibration, while an α of 0 means that the importance of Q_i is discarded and $\acute{Q}_i$ is equal to the mean of all queries.

Additionally to mixing the confidence scores, this mechanism also modifies the predicted labels by smoothing them using the same parameter α. Instead of having plain one-hot encoders as labels, α determines now the weightage of the label for a given positive query, while all the other positive queries that contributed to the mean calculation share the $1 - \alpha$ value to smooth the corresponding labels uniformly. The $1 - \alpha$ remainder needs to be further scaled down according to the number of positive occurrences, which can be done in two ways. The first option, "$orig$", is to directly divide $1-\alpha$ to the number of positive queries, which would result in a smoothed one-hot encoder that does not necessarily sum up to one, while the second possibility "new" is to divide $1 - \alpha$ to the number of unique classes from all the positive occurrences.

As mentioned above, logits mixing employs a regularization loss using the new mixed class scores and the smoothed one-hot encoding labels. This loss is computed together with the usual classification and regression loss (bounding box loss) as shown here:

$$\mathcal{L} = \mathcal{L}_{cls} + \mathcal{L}_{bbox} + \lambda \mathcal{L}_{regularizer}, \qquad \text{Eq. 4-3}$$

where λ can be used to scale the contribution of the regularization loss. The intuition behind this regularizer is to capture details from around the object instances in the logit space and to compute a non-zero entropy label distribution, which should penalize overconfident decisions and improve model calibration.

4.1.3 Temperature Scaling

Temperature scaling is a post-hoc technique used directly on an already trained model to improve the calibration of its predictions and adapting its confidence scores to better match the expected accuracy [GUO17]. It does not affect the actual predictions of the model but it adjusts the predicted probabilities to better match the true likelihoods in order to provide a more reliable estimate of uncertainty. The working principle of this methods consists of introducing a single parameter,

temperature T, which is used to adjust the Softmax output of the model, as shown in the following formula:

$$\text{softmax}_i(z;T) = \frac{\exp\left(\frac{z_i}{T}\right)}{\sum_{j=1}^{C}\exp\left(\frac{z_j}{T}\right)} \quad \text{Eq. 4-4}$$

where:

- z: The logits of the model
- T: The temperature parameter used for scaling the logits.
- C: The number of classes in the classification task.

This parameter allows the smoothing (for T > 1) or sharpening (for T < 1) of the Softmax distribution, thus modifying the predicted probabilities. The approach of determining the temperature is done by finding the most optimal value T that minimizes the classification NLL on a the training dataset.

4.1.4 Depth Scaling

Since most calibration methods focus only on calibrating the classification results, there are few to none mechanism for regression uncertainty calibration. Therefore, this work proposes Depth Scaling, a simple but effective post-hoc calibration method specifically designed for calibrating the regression output of the architecture's decoder.

Depth Scaling extends the idea of Temperature Scaling by scaling the main diagonal (variance) of the covariance matrix. The covariance matrix of a bounding box can be calculated in multiple ways, for example, from the output of all decoder layers or from the created clusters during DBScan. The variance is scaled using the distance between the ego vehicle and the detected object multiplied by an optimization factor, as shown in the following equation:

$$\mathbf{\Sigma}_{scaled} = \mathbf{\Sigma} + k \cdot d \cdot \mathbf{I}, \quad \text{Eq. 4-5}$$

where $\mathbf{\Sigma}_{scaled}$ is the newly computed covariance matrix, d the distance to the detected object, k the optimized parameter and I the identity matrix. In contrast to the Temperature Scaling, where a single constant is learnt for the whole model, Depth Scaling finds for each class a specific k, making it a per-class scaling approach.

Since k is an optimized scaler, it needs to be computed on the training dataset. In this thesis, the optimal value of k is found by utilizing a gradient free technique, that uses Brent's algorithm [GEG92] for finding a local minimum by minimizing the regression NLL. This optimization problem is described as following:

$$\min_k \frac{1}{n}\sum_{i=1}^{n} -\log\left(f(\mathbf{y_{gt,i}}, \mathbf{\Sigma}_{scaled,i}(k,d))\right), \quad \text{Eq. 4-6}$$

with:

$$f(\mathbf{y_{gt,i}}, \mathbf{\Sigma}_{scaled,i}) = \frac{1}{\sqrt{(2\pi)^D |\mathbf{\Sigma}_{scaled,i}{}^{-1}|}} \exp\left(-\frac{1}{2}(\mathbf{y_{gt,i}} - \mathbf{x_{pred,i}})^T \mathbf{\Sigma}_{scaled,i}^{-1}(\mathbf{y_{gt,i}} - \mathbf{x_{pred,i}})\right), \quad \text{Eq. 4-7}$$

where $\mathbf{\Sigma}_{scaled,i}$ is the scaled covariance matrix, $\mathbf{x_{pred,i}}$ is the predicted bounding box, $\mathbf{y_{gt,i}}$ represents the ground truth bounding box, D is the dimension and n the number of samples.

4.2 Cluster Merging

An important aspect for creating the necessary baselines, Monte-Carlo Dropout, Deep Ensembles and Deep Sub-Ensembles, is implementing the process used for merging the predicted clusters. Specifically when it comes to 3D Object Detection, this process can be quite complex due the seven degrees of freedom that come with a 3D oriented bounding box.

In this work, multiple merging approaches were designed. The first technique, described in Algorithm 1, is more straight forward and bases on few simple steps. At first, one needs to iterate through all the clusters, and at each one, compute the 3D IoU over all the bounding boxes of the current cluster. The bounding boxes with an overlap higher than a specified threshold are given a positive index, the ones that do not overlap are ignored. The bounding box and score values of the positive ones are then averaged, while the label is picked based on a voting system inside the cluster. Label with most votes from the positive indices is assigned as the new label.

Algorithm 1 IoU and Average Merging Algorithm

```
acquire clusters of bounding boxes, scores and labels.
for cluster in clusters do
    calculate overlap between bounding boxes in cluster.
    get positive indices of overlapping bounding boxes.
    average bounding boxes in cluster at positive indices.
    average scores in cluster at positive indices.
    select the label with the most votes from positive indices in cluster.
    store the new bounding box, score, and label.
end for
return new bounding boxes, scores, and labels.
```

The steps of the second merging algorithm are shown in Algorithm 2. This concept builds upon the previous, by introducing a class-wise DBScan clustering method. In this merging technique, one iterates through every class, computes a 3D IoU overlap table over all class occurrences to serve it as input for the DBScan clustering. Then, the same averaging and voting process happens as before, followed by returning the newly calculated bounding box, score and label of the DBScan cluster for the given class. It needs to be mentioned, that both a general and a class-wise DBScan clustering method were tried, but the last one showed better performance and was therefore kept for further experiments.

Algorithm 2 DBScan Merging Algorithm

```
acquire clusters of bounding boxes, scores and labels.
for class in classes do
    calculate overlap table between all bounding boxes in class.
    perform DBScan clustering with class overlap table
    for DBScan cluster in DBScan clusters do
        average bounding boxes in DBScan cluster.
        average scores in DBScan cluster.
        select the label with the most votes in DBScan cluster.
        store the new bounding box, score, and label.
    end for
end for
return new bounding boxes, scores, and labels.
```

4.3 Implementation Flow of Uncertainty Estimates

4.3.1 Logits Mixing and Modulation

The above introduced calibration methods are experimented on both the PETR and PillarFormer architectures. Fig. 4-2 shows the flowchart used with both these approaches. It consists of either taking a pretrained backbone model, in the case of PETR, or training the model from scratch, as in the case of PillarFormer and then doing an a complete training on it. This process results into a so called "vanilla" model, which is a model of an architecture that has been fully trained until it reached the expected performance for the architecture. Afterwards, using this vanilla model, experiments with logits modulation, logits mixing and also with both schemes are executed by training individual models with each setup. In the case of logits mixing, an ablation study is performed by varying the α and λ parameters to search for best results. The flowchart ends with an in-depth evaluation that includes performance as well as uncertainty quantification metrics.

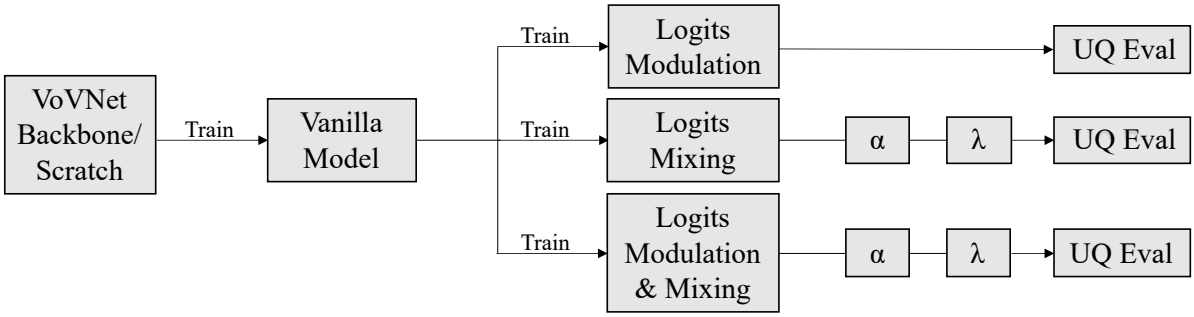


Fig. 4-2: Flow of PETR and PillarFormer experiments. A vanilla model of PETR and PillarFormer is trained with three different main setup, Logits Modulation, Logits Mixing and Logits Modulation and Mixing, followed by an uncertainty quantification evaluation.

4.3.2 Deep Ensembles

The implementation of the DE baseline is shown in Fig. 4-3. Since this baseline requires multiple models, up to N vanilla models are trained with a different random seed. During this process, these N models produce N distinct outputs, and using one of the two merging methods

discussed above, a final output is computed and evaluated based on performance and UQ metrics.

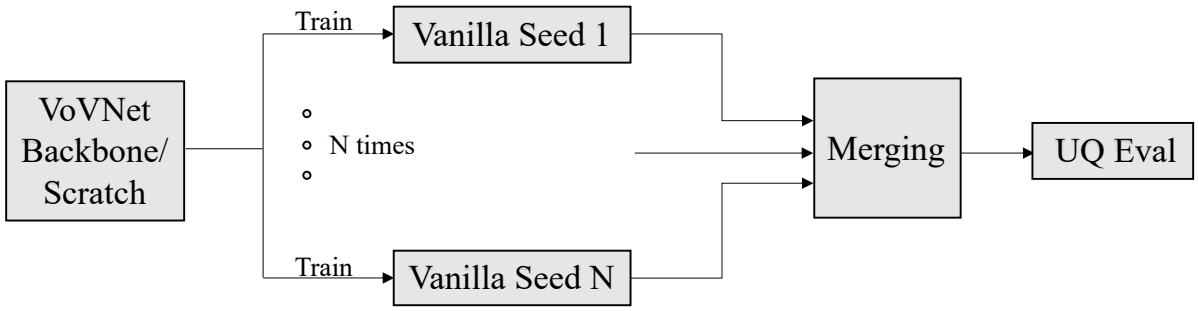


Fig. 4-3: The implementation flow of Deep Ensembles. For the PETR architecture, from the backbone, up to N models are trained, independent from one another by varying the random seed. The output of every independent vanilla model is accumulated and merged together to be then evaluated. In the case of PillarFormer, N models are trained from scratch with distinct randomness.

4.3.3 Deep Sub-Ensembles

Starting from a pre-trained vanilla model as foundation, up to N new different heads are to be trained. For each head, a regression and a classification loss are applied. DSE produces up to N head outputs, which require a merging operation before the evaluation to obtain the final output. Fig. 4-4 shows this procedure.

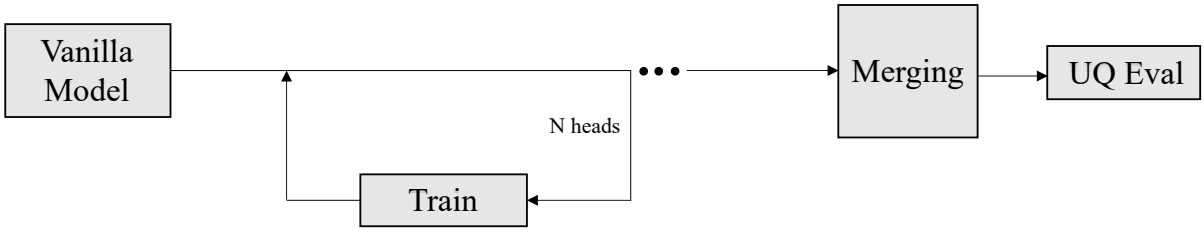


Fig. 4-4: Experimental approach for implementing Deep Sub-Ensembles. During the training process, multiple heads are trained sequentially, initializing a loss for each. During interference, the predictions of all heads are accumulated for merging and evaluation.

4.3.4 Monte-Carlo Dropout

The Monte-Carlo Dropout (MCD) method does not need any additional training of a vanilla model as it can be performed directly during inference, as shown in Fig. 4-5. In the case of both architectures, it takes a pre-trained vanilla model and computes a forward pass up to N times, each forward pass having dropout enabled. Since random nodes are being dropped in each forward pass, MCD produces N different outputs, which need to be merged into one. A visualization of the modules where MCD is applied is shown in Fig. 4-6. The red crosses symbolize dropping neurons in the feed-forward networks inside of the architecture.

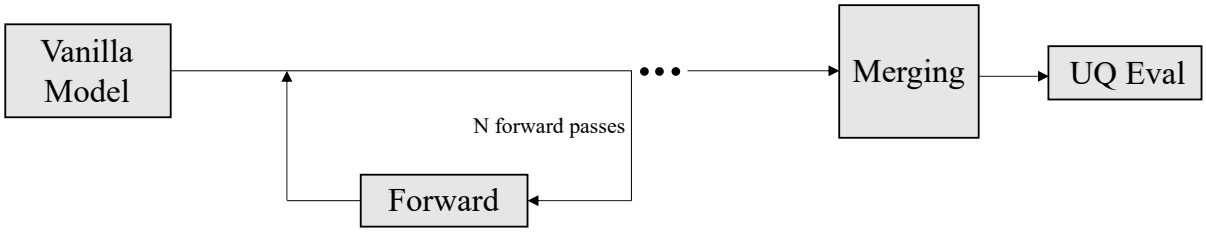


Fig. 4-5: Experimental approach for implementing Monte-Carlo Dropout. Multiple sequential forward passes are done, each with random nodes dropped. Each forward pass results into a new prediction, accumulating and merging all results then in a final output for evaluation.

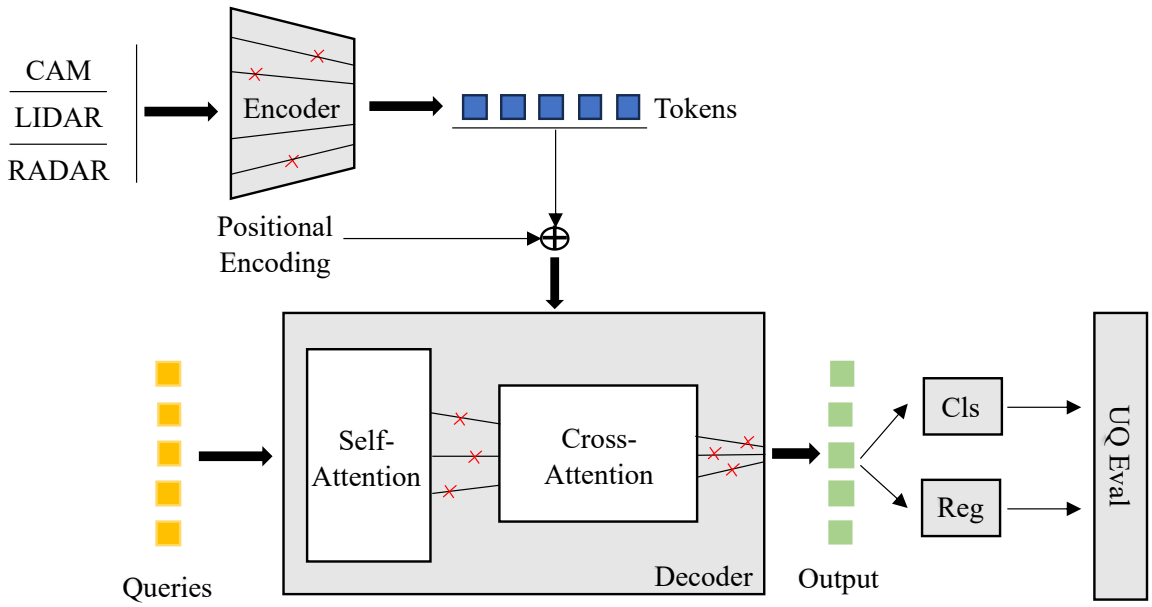


Fig. 4-6: During MCD, random nodes from the multi-layer perceptrons from both the encoder and decoder are dropped, this process being indicated with the red crosses.

5 Results

This chapter will provide an in-depth overview of the camera, radar and LiDAR experiments done in this thesis using the introduced baseline methods and the proposed calibration techniques, together with a showcase of the gathered results and a discussion based on them. Furthermore, two main research topics are tackled, regarding the presented results:

1. Which type of sensor, camera, radar and LiDAR, provides the best results, based on criteria such as performance and model calibration?

2. What is most optimal calibration technique, given the required resources to implement it, between the loss oriented technique and the post hoc scaling methods?

All experiments for camera-based, radar-based with one and five scans and lidar-based data follow the same experimental structure. This consists of results from the vanilla model, an uncertainty quantification baseline made up of Monte-Carlo Dropout with $N = [2, 4, 6]$ forward passes, and of Deep Ensembles with $N = [2, 4]$ random seeded models. Additionally, due to time and resource constrains, the Deep Sub-Ensembles method is used only for the camera-based experiments with the PETR architecture and consists of $N = 2$ heads. All baseline tests are using the same merging algorithm, the class-wise DBScan clustering, since it outperformed in detection accuracy the proposed Algorithm 1. Moreover, it should be mentioned that also a general (not class-wise) DBScan clustering method was tried, but was still underperforming compared to the class-wise option.

For the analysis on uncertainty calibration, all results include the Temperature Scaling method used on the vanilla model, followed by a plain Logits Modulation and then a plain Logits Mixing experiment with $\alpha = 0.9$ and original remainder to assess the calibration performance of each technique with a basic setup. Afterwards, six configurations of Logits Mixing combined with Logits Modulations are tested, three of them having the original one hot remainder proposed by [MUN24], and three with the variation proposed in this thesis of one hot remainders normalized to 1. The original one-hot remainder is indicated in the following sections by $orig$, while the other variant by new. Both these sets of three are being tested with $\alpha = [0.9, 0.8, 0.5]$. In addition, all Logits Mixing trials were done using the same $\lambda = 1$ regularization scaler, since it consistently performed best. The second last experiment combines Temperature Scaling with the best ECE performing calibration model, to evaluate if the classification calibration can be further increased while combining all implemented mechanisms. Lastly, the Depth Scaling experiment is performed on the previous one to calibrate the regression output. Further in this chapter, the term "Depth Scaling" will be used to describe the overall best calibrated model, which also includes the most optimal configuration for reducing the classification uncertainty.

The uncertainty quantification evaluation metrics for the camera-, radar- and LiDAR- driven experiments are the regression Negative Log-Likelihood (NLL) and Mahalanobis Distance (MD) for the regression task, and the classification Negative Log-Likelihood and Expected Calibration Error (ECE) for the classification outcome. To analyze the overall detection performance and accuracy, for the

camera-based experiments on NuScenes, both mean average precision (mAP) and NuScenes Detection Score (NDS) are used, while for the point cloud-based tests, only the mAP is calculated. Additionally, the computational time required for the forward pass is also shown.

5.1 Detection Performance

Fig. 5-1 shows the achieved performance of the PillarFormer architecture on the one scan radar, five scans radar and LiDAR datasets using a vanilla model. The results are divided into class-wise AP for the three classes and the mAP. The LiDAR is performing best, followed by the multi-scan radar and the one scan. Additionally, the five scans radar experiment has an additional temporal feature, due to accumulating consecutive sweeps.

Method	**Features**	**Car AP** ↑	**Pedestrian AP** ↑	**Cyclist AP** ↑	**mAP** ↑
PillarFormer (1 scan)	x, y, z, RCS, v_r	26.5	20.5	54.3	33.8
PillarFormer (5 scans)	x, y, z, RCS, v_r, t	32.9	28.2	57.2	39.4
PillarFormer (LiDAR)	x, y, z, intensity	62.8	60.5	76.1	66.4

Fig. 5-1: Performance comparison between the radar one scan, radar five scans and the LiDAR datasets of VoD.

5.2 Camera-driven Results

Fig. 5-2 shows the results of the PETR experiments on the camera dataset of NuScenes. Starting with the UQ baseline, firstly, it can be noticed that the detection and accuracy performance is still somewhat below the vanilla model. Since these baselines all require bounding box merging, doing it especially in the 3D space proves to be much more difficult than 2D merging, due to the addition of bounding box depth and its orientation. In the case of MCD and DE, it can be seen that increasing the N returns better performance and hypothetically, with a big enough N, it should approach the performance of the vanilla, but obviously, with much greater computational requirements. Nevertheless, the MCD and DSE techniques improve on the classification uncertainty over the vanilla model, while in the case of DE it worsens. Looking at the regression results of the baseline, in the case of the MD metric, it can be seen that for the DE and MCD experiments with $N = 4$, the values are slightly better than the base model. The trend indicates that with growing N, the uncertainty of the bounding boxes should technically decrease even further. The regression NLL values also decrease significantly with larger N, but are still severely larger than vanilla, due to the fact that this metric takes also accuracy into consideration, besides uncertainty.

Looking at the calibration techniques of Fig. 5-2, it can be immediately seen that the forward pass time for all tests remains the same with the vanilla model and the detection performance and accuracy scores are showing similar values. Regarding classification calibration, all implemented techniques are showing improvements in terms of ECE and NLL compared to the base model. Generally, combining both the modulation and mixing of logits seems to be producing the best results, better than TS or than applying them individually, especially in the case of ECE. Moreover, assigning an α of 0.8 appears to be more rewarding, as well as using the other variant (new) of one-hot encoding,

	Regression		Classification				
	NLL ↓	MD ↓	NLL ↓	ECE ↓	mAP ↑	NDS ↑	Time ↓
Vanilla	414.18	10.652	0.1488	0.0383	0.3829	0.4304	0.21s
MCD$_{N=2}$	176936	15.088	0.1988	0.0332	0.3066	0.3976	0.27s
MCD$_{N=4}$	47701	10.191	0.1645	0.0355	0.3219	0.4026	0.56s
MCD$_{N=6}$	7947.2	7.6981	0.1734	0.0367	0.3138	0.3968	0.78s
DSE$_{N=2}$	461293	19.711	0.2166	0.0281	0.2680	0.3671	0.21s
DE$_{N=2}$	181487	14.911	0.2906	0.0397	0.2832	0.3933	0.27s
DE$_{N=4}$	86742	10.477	0.2580	0.0478	0.3233	0.4103	0.56s
TS	414.18	10.652	0.1416	0.0343	0.3804	0.4331	0.21s
Mod	500.69	10.399	0.1385	0.0323	**0.3830**	0.4379	0.21s
Mix$_{\alpha=0.9,orig}$	341.72	9.3231	0.1391	0.0326	0.3810	0.4343	0.21s
Mod+Mix$_{\alpha=0.9,orig}$	428.17	9.8108	0.1407	0.0317	0.3803	0.4352	0.21s
Mod+Mix$_{\alpha=0.8,orig}$	407.94	9.6712	0.1410	0.0292	0.3795	0.4349	0.21s
Mod+Mix$_{\alpha=0.5,orig}$	384.58	9.8014	0.1438	0.0291	0.3771	0.4335	0.21s
Mod+Mix$_{\alpha=0.9,new}$	429.29	10.064	0.1385	0.0294	0.3770	0.4336	0.21s
Mod+Mix$_{\alpha=0.8,new}$	389.31	9.6587	0.1410	0.0289	0.3804	0.4395	0.21s
Mod+Mix$_{\alpha=0.5,new}$	363.44	9.6651	0.1485	0.0300	0.3773	0.4341	0.21s
TS + Mod+Mix$_{\alpha=0.8,new}$	363.44	9.6651	0.1362	0.0264	0.3804	0.4395	0.21s
Depth Scaling	**2.5829**	**1.4955**	**0.1362**	**0.0264**	0.3804	**0.4395**	**0.21s**

Fig. 5-2: NuScenes experiment on the camera dataset using the PETR architecture. It presents the evaluation of the implemented baselines and several different configurations of the introduced calibration techniques.

since it produces slightly better results. Furthermore, combining TS with the best performing Mix and Mod configuration improves even more the classification calibration. Depth Scaling is the only technique that influences the regression uncertainty directly, showing much bigger improvement than all other experiments. Additionally, since Depth Scaling is independent from the other techniques, it can be used on top of all other calibration mechanisms to produce the most calibrated model across all the performed experiments.

Fig. 5-3 and Fig. 5-4 are showing the per-class head to head comparison between the vanilla model and the best calibrated experiment configuration, previously called Depth Scaling. In the case of the regression uncertainty, one can easily observe that all classes are having major improvements over the base model, for both NLL and MD. Switching to the classification metrics, the same pattern of improvements occurs for all the 10 classes. Both the classification NLL and the ECE are showing better results in the table of the calibrated model compared to the vanilla, besides the $constructionvehicle$ class, which seems to have a slightly worse NLL score after calibration. This is mostly due to the fact that the NLL metric is a measure for both accuracy and uncertainty, while ECE is a pure uncertainty metric.

	Regression		Classification	
Class	NLL ↓	MD ↓	NLL ↓	ECE ↓
car	328.18	9.4129	0.1555	0.0536
truck	407.81	11.213	0.1317	0.0448
bus	375.81	10.700	0.1207	0.0277
trailer	443.34	13.327	0.1728	0.0483
construction vehicle	792.7	14.9017	0.1137	0.042
pedestrian	362.85	10.148	0.1341	0.0426
motorcycle	454.33	9.741	0.0716	0.0311
bicycle	314.19	8.767	0.074	0.0335
traffic cone	339.05	8.790	0.1595	0.0434
barrier	323.51	9.515	0.3547	0.0168

Fig. 5-3: Class-wise regression and classification results on NuScenes of the vanilla PETR model.

	Regression		Classification	
Class	NLL ↓	MD ↓	NLL ↓	ECE ↓
car	1.9373	1.4256	0.1502	0.0431
truck	2.6373	1.5017	0.1174	0.0307
bus	2.7124	1.4933	0.1199	0.0256
trailer	3.3059	1.6245	0.1395	0.0301
construction vehicle	3.6789	1.6199	0.1224	0.0286
pedestrian	2.4967	1.4643	0.1200	0.0268
motorcycle	2.4912	1.4844	0.0666	0.0200
bicycle	2.2400	1.4704	0.0648	0.0216
traffic cone	2.0457	1.4284	0.1335	0.0258
barrier	2.2833	1.4426	0.3276	0.0116

Fig. 5-4: Class-wise regression and classification results on NuScenes with best performing PETR setup consisting of Depth Scaling, Temperature Scaling and Logits Modulation and Mixing with $\alpha = 0.8$ and the new mixing remainder approach.

5.3 Radar-driven Results

Fig. 5-5 presents the results of the radar experiments on PillarFormer done using the one scan radar dataset of VoD. First of all, due to the relative limited size of the VoD dataset, combined with the sparsity of the radar point clouds, all radar trainings and evaluations have shown small but noticeable fluctuations in the results, best seen along the "mAP" column.

Starting with the UQ baseline, the regression results are once again showing improvements in both NLL and MD with increasing N, at the cost of computational resources as seen under the "Time" column. In the case of classification, for the DE technique, a trend for lowering the uncertainty can be observed, when using larger N, while in the case of MCD, this trend is less clear, since both the NLL and ECE are going down then back up. Once again, this might be due to the variability of the data. Performance-wise, for the MCD technique the mAP is once again lower than vanilla, since it uses the same 3D merging technique, while in the case of DE, mAP seems to be growing closer to the performance of the base model.

Analyzing the implemented calibration methods, it can once again be observed that the time needed for a forward pass remains equal at 0.07 for all configurations, these techniques requiring almost no additional overhead compared to the baseline. Firstly, analyzing the first three experiments, where each technique (TS, Mod and Mix) is individually tested, it can be seen that for this single radar scan, the TS method is showing better classification calibration for both the NLL and ECE metrics, also highlighting improvements over the vanilla model. Covering the Mod and Mix configurations, both the NLL and ECE results are indicating small fluctuations, nevertheless, these are still returning lower ECE values compared to the base test. Out of these six Mod and Mix configurations, the one using $\alpha = 0.5$ and the new one-hot remainder approach achieved the lower ECE score. Applying then TS on the best performing model achieves even slightly better classification calibration. Furthermore, applying the Depth Scaling approach with the configuration that achieved the best ECE value displays a significant improvement for both regression metrics.

Overall, the final experiment of Fig. 5-5 manages to obtain the lowest uncertainty values in three out of the four analyzed metrics.

The second radar experiment has been done using the VoD dataset that consists of five successive radar scans and the gathered results are showcased in Fig. 5-6. Although there is a third radar dataset made of three consecutive radar scans, it has not been utilized for these tests, since its result would theoretically be in-between the two other presented radar tables.

First of all, accumulating multiple successive scans together should yield to a better detection performance and accuracy, because the point clouds become denser due to addition of extra radar measurements. Moreover, since the scans are consecutive, temporal information is also added, which can contribute further to improving the performance. These statements can be verified in Fig. 5-6 under the "mAP" column as it presents generally higher values compared to the single radar scan results of Fig. 5-5. Moreover, the five radar scans experiment also shows less variability or fluctuations in the results compared to the one scan radar outcome. The cost of accumulating multiple scans together is having a slightly bigger computational overhead for the forward passes, because having more points into the point cloud induces a slower voxelization.

	Regression		Classification			
	NLL ↓	MD ↓	NLL ↓	ECE ↓	mAP ↑	Time ↓
Vanilla	4455.2	170.01	0.1264	0.0256	33.8	0.07
MCD$_{N=2}$	39755	190.07	0.1477	0.0494	32.3	0.14s
MCD$_{N=4}$	38487	161.91	**0.1087**	0.0453	30.7	0.28s
MCD$_{N=6}$	5357.5	156.14	0.1222	0.0537	31.0	0.40s
DE$_{N=2}$	394591	179.52	0.2648	0.0813	34.0	0.15s
DE$_{N=4}$	36575	136.04	0.1796	0.0809	**36.7**	0.30s
TS	4455.2	170.01	0.1154	0.0161	35.7	0.07s
Mod	4447.7	184.56	0.1391	0.0177	35.9	0.07s
Mix$_{\alpha=0.9,orig}$	4266.0	175.46	0.1568	0.0263	35.0	0.07s
Mod+Mix$_{\alpha=0.9,orig}$	4583.6	181.99	0.1489	0.0220	34.7	0.07s
Mod+Mix$_{\alpha=0.8,orig}$	3917.6	178.59	0.1369	0.0162	35.2	0.07s
Mod+Mix$_{\alpha=0.5,orig}$	5211.2	175.31	0.1627	0.0257	34.4	0.07s
Mod+Mix$_{\alpha=0.9,new}$	4710.8	181.54	0.1448	0.0194	33.8	0.07s
Mod+Mix$_{\alpha=0.8,new}$	4630.1	177.16	0.1495	0.0205	35.6	0.07s
Mod+Mix$_{\alpha=0.5,new}$	4332.0	180.98	0.1318	0.0153	33.7	0.07s
TS + Mod+Mix$_{\alpha=0.5,new}$	4332.0	180.98	0.1302	0.0152	33.7	0.07s
Depth Scaling	**10.370**	**1.078**	0.1302	**0.0152**	33.7	**0.07s**

Fig. 5-5: View-of-Delft experiment using the single radar scan dataset with the PillarFormer architecture for creating an uncertainty quantification baseline and implementing the proposed calibration techniques.

Furthermore, looking at the UQ baseline for Fig. 5-6, a similar behavior to the single scan radar is displayed. The higher the N, the more are the regression and classification metrics improving, especially for the MD, where starting from $N = 4$ it already shows lower regression uncertainty compared to the vanilla evaluation. Once again, the MCD technique struggles to keep up the same mAP performance as the base model, while the DE method manages to outscore it.

Going over to the calibration techniques, between the three individual tests (TS, Mod and Mix), it is the Mod experiment that achieves the lowest ECE, while the Mix has the overall lowest classification uncertainty between them. The best Mod and Mix configuration proves to be with $\alpha = 0.5$ with the new remainder, since it highlights the smallest ECE score and the second lowest NLL value. Adding TS on top of it improves the ECE marginally, while also slightly worsening the NLL score. When applying Depth Scaling, both the regression NLL and MD are improving considerably.

	Regression		Classification			
	NLL ↓	MD ↓	NLL ↓	ECE ↓	mAP ↑	Time ↓
Vanilla	5738	186.89	0.1380	0.0275	39.4	0.08s
MCD$_{N=2}$	410678	200.47	0.1477	0.0536	32.2	0.15s
MCD$_{N=4}$	38483	161.91	**0.1087**	0.0453	30.7	0.29s
MCD$_{N=6}$	5586.8	156.14	0.1222	0.0537	31.3	0.41s
DE$_{N=2}$	41805	191.64	0.2839	0.0959	38.8	0.15s
DE$_{N=4}$	25310	136.04	0.1997	0.0813	**40.5**	0.31s
TS	5728.1	186.64	0.1271	0.0199	38.6	0.08s
Mod	5365.9	179.28	0.1312	0.0148	36.4	0.08s
Mix$_{\alpha=0.9,orig}$	4532.2	166.04	0.1277	0.0175	35.6	0.08s
Mod+Mix$_{\alpha=0.9,orig}$	5090.4	173.05	0.1216	0.0147	36.3	0.08s
Mod+Mix$_{\alpha=0.8,orig}$	4839.5	169.71	0.1170	0.0154	36.4	0.08s
Mod+Mix$_{\alpha=0.5,orig}$	5412.0	183.02	0.1229	0.0138	38.3	0.08s
Mod+Mix$_{\alpha=0.9,new}$	4923.4	171.30	0.1202	0.0132	37.1	0.08s
Mod+Mix$_{\alpha=0.8,new}$	4086.8	170.68	0.1238	0.0148	36.9	0.08s
Mod+Mix$_{\alpha=0.5,new}$	5716.6	178.28	0.1211	0.0108	36.1	0.08s
TS + Mod+Mix$_{\alpha=0.5,new}$	5716.6	178.28	0.1223	0.0101	36.0	0.08s
Depth Scaling	**10.303**	**1.0668**	0.1223	**0.0101**	36.0	**0.08s**

Fig. 5-6: View-of-Delft experiment using the radar dataset with five successive scans with the PillarFormer architecture for creating an uncertainty quantification baseline and implementing the proposed calibration techniques.

5.4 LiDAR-driven Results

Fig. 5-7 shows the final set of experiments on the LiDAR dataset of VoD using the PillarFormer architecture. Starting with the UQ baseline, the MCD technique achieves better classification calibration than the base model for both NLL and ECE metrics, while still being somewhat behind in the performance and accuracy of detections. Moreover, it can be observed that once again, for the regression metrics, increasing N yields better values for both NLL and MD. Equivalently, for DE, a higher N demonstrates overall improvement up until the ECE which gets slightly larger.

Over to the classification calibration techniques, when used individually, TS appears to display better classification calibration than Mod or Mix. Furthermore, applying TS on the best ECE performing configuration of Mod and Mix, which is with $\alpha = 0.8$ and new remainder, the model attains the lowest ECE and second lowest classification NLL values among the calibration methods used. Using Depth Scaling on the best performing ECE configuration yields once again substantial improvements for reducing the regression uncertainty.

Additionally, it should be mentioned that in the case of the LiDAR dataset of VoD, there are no more significant fluctuations between experiments, seen under the "mAP" column, as both the vanilla

	Regression		Classification			
	NLL ↓	MD ↓	NLL ↓	ECE ↓	mAP ↑	Time ↓
Vanilla	27248	261.60	0.2261	0.1652	66.4	0.05s
MCD$_{N=2}$	454692	219.18	0.1808	0.0881	65.4	0.11s
MCD$_{N=4}$	36148	191.74	**0.1649**	0.0900	64.9	0.19s
MCD$_{N=6}$	5736.4	186.82	0.1720	0.0940	64.7	0.26s
DE$_{N=2}$	568089	223.45	0.4389	0.1923	58.8	0.12s
DE$_{N=4}$	57339	179.48	0.3946	0.2161	62.3	0.21s
TS	27247	261.62	0.2360	0.0619	66.4	0.05s
Mod	36610	269.24	0.2376	0.0711	67.5	0.05s
Mix$_{\alpha=0.9,orig}$	40049	276.18	0.2425	0.0675	66.9	0.05s
Mod+Mix$_{\alpha=0.9,orig}$	38145	275.11	0.2327	0.0737	**67.7**	0.05s
Mod+Mix$_{\alpha=0.8,orig}$	40714	290.31	0.2334	0.0659	67.0	0.05s
Mod+Mix$_{\alpha=0.5,orig}$	42146	286.11	0.2356	0.0721	66.5	0.05s
Mod+Mix$_{\alpha=0.9,new}$	37755	276.11	0.2201	0.0763	67.6	0.05s
Mod+Mix$_{\alpha=0.8,new}$	43849	288.86	0.2288	0.0645	67.5	0.05s
Mod+Mix$_{\alpha=0.5,new}$	43329	286.86	0.2353	0.0732	67.4	0.05s
TS + Mod+Mix$_{\alpha=0.8,new}$	43329	286.86	0.2252	0.0609	67.6	0.05s
Depth Scaling	**10.826**	**1.158**	0.2252	**0.0609**	67.6	**0.05s**

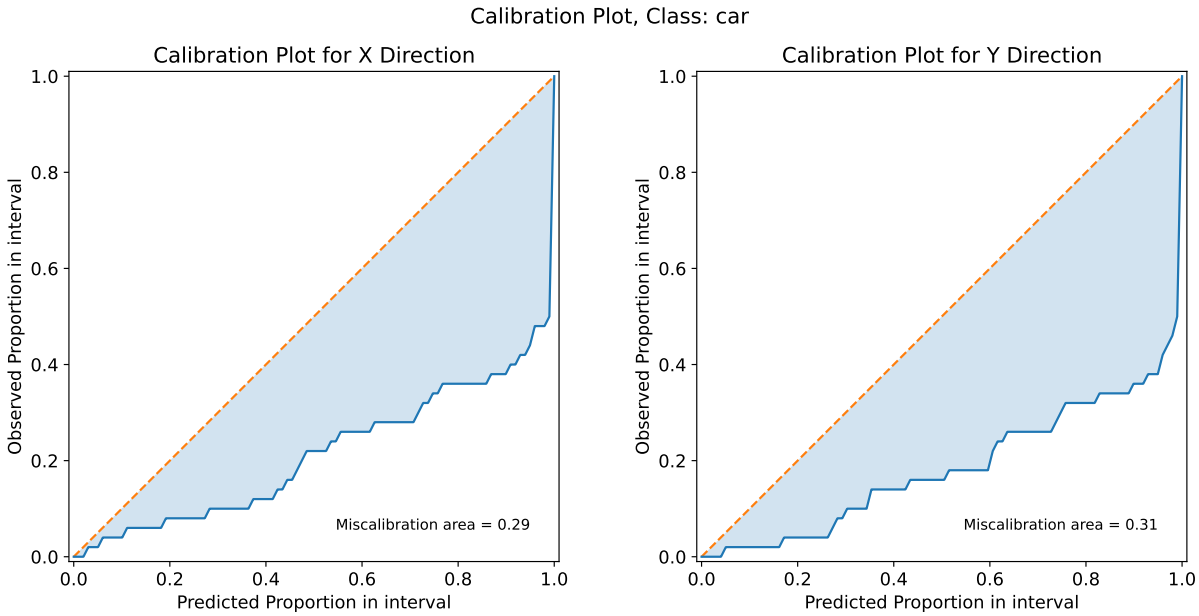
Fig. 5-7: View-of-Delft experiment using the LiDAR dataset with the PillarFormer architecture for creating an uncertainty quantification baseline and implementing the proposed calibration techniques.

model and the calibration configuration show similar values. The reason is due to the point clouds obtained through the LiDAR sensor, which are much denser compared to the ones from the one radar scan and five radar scans datasets.

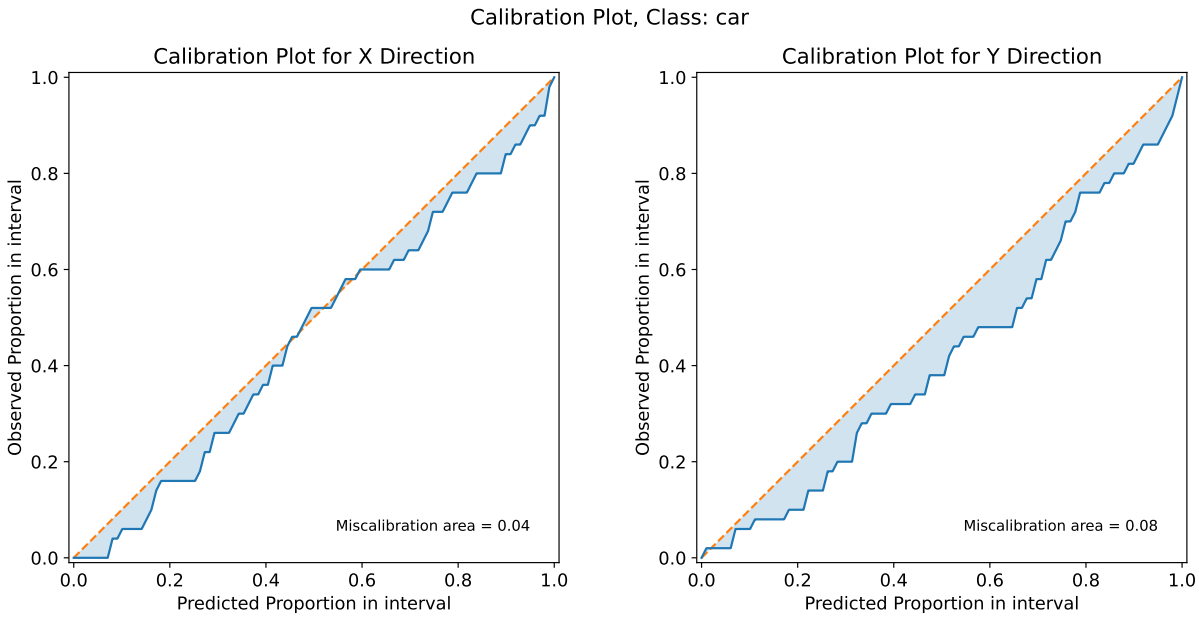
5.5 Results Visualization

Fig. 5-8 displays a comparison between the vanilla model and the Depth Scaling experiment. Each plot consists of an average calibration plot of the x-direction and for the y-direction, for the car class. It can be observed, that in the case of the base model, the miscalibration area is significant, the model showing signs of high overconfidence, given that the curve is under the $y = x$ diagonal. In the case of Depth Scaling configuration, which calibrates the regression uncertainty, it can be seen that the miscalibration area is a lot smaller. Especially in the x-direction case, the plotted curve follows very closely the ideal $y = x$ diagonal, while also the y-dimension show much better calibration than the vanilla model.

The second visualization plot is the residual uncertainty, shown in Fig. 5-9, which is composed of a representation for the x-dimension, a representation of the y-dimension and their combined effect. For a proper model calibration, the residual error should follow the normal distribution. One can see



(i) Vanilla

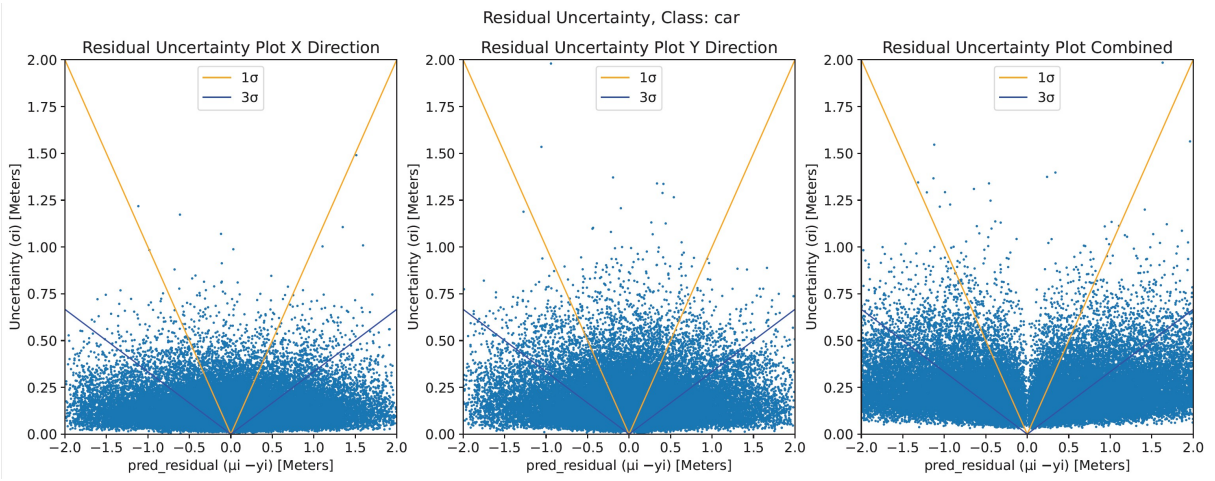


(ii) Depth Scaling

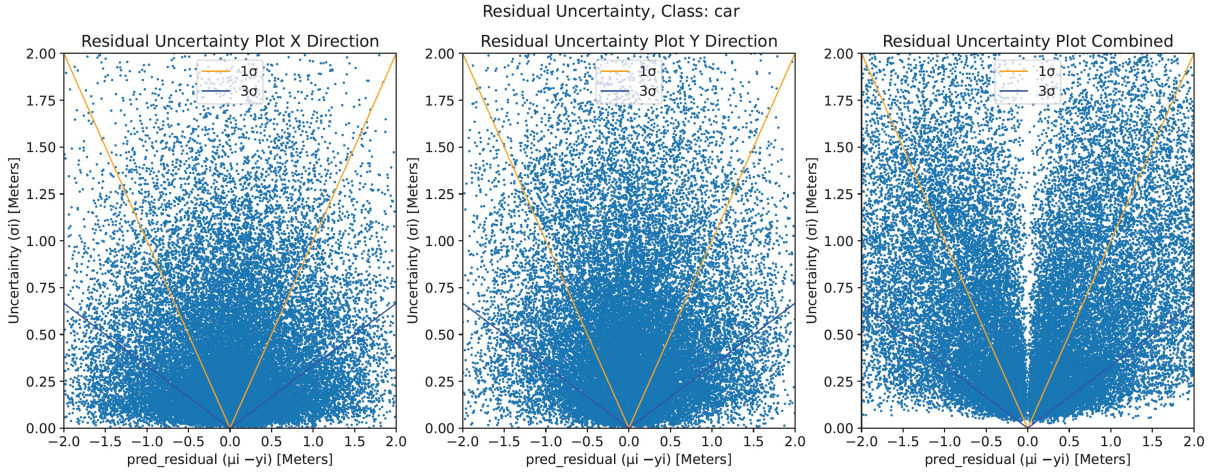
Fig. 5-8: Regression calibration plot of the vanilla configuration (i) and the Depth Scaling experiment (ii), which shows considerable improvement regarding regression uncertainty.

in Fig. 5-9 (i) that the vanilla model underestimates the uncertainties, since a high percentage of points lies outside the 3σ region. An improvement can be seen in Fig. 5-9 (ii), where the $\text{DE}_{N=4}$ has significantly more point inside the 1σ and the 3σ regions. The best calibrated model is achieved by the Depth Scaling configuration in Fig. 5-9 (iii), which tends to fulfil the imposed conditions the best, since the majority of points is above the orange line, while almost all of the points lie above the blue line.

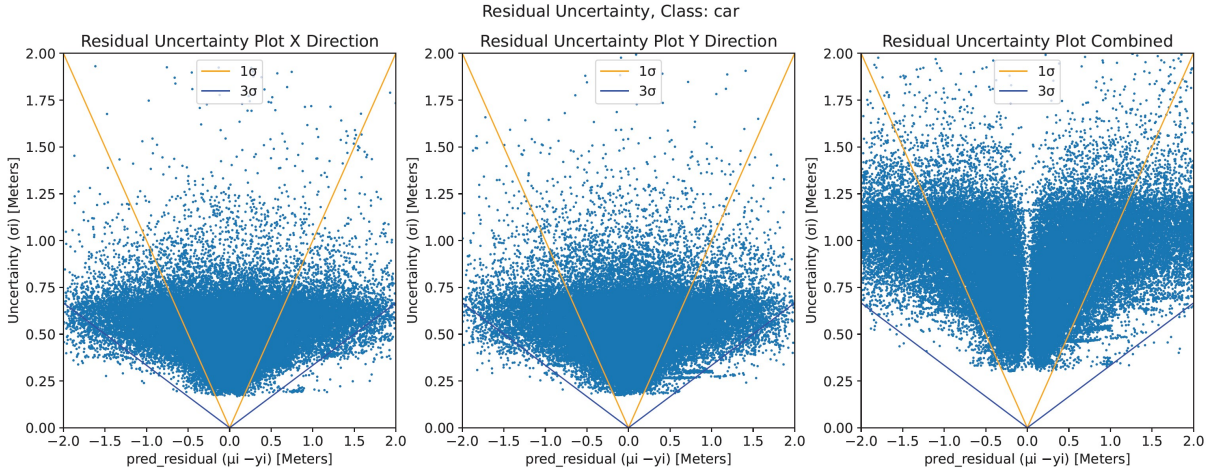
Lastly, the third plot, seen in Fig. 5-10, visualizes the differences between the vanilla model and the overall best calibrated model, the Depth Scaling experiment, using the reliability diagram on three NuScenes classes, car, bus and $pedestrian$. All these diagrams were created using 10 confidence interval and the respective ECE score is displayed inside of each plot. Fig. 5-10 (i) and (ii) show the comparison for the car class. On the left, (i), the vanilla model is being overconfident for predictions with less than 0.4 confidence, while above this value it acts underconfidently. On the right, (ii), the Depth Scaling plot improves on the results of the base model at pretty much every confidence interval. This improvement can be best seen looking at the [0.4, 0.5] and [0.6, 0.7] intervals, where the Depth Scaling experiment reaches almost perfect calibration. Looking at the bus example, experiment (iv) shows again better calibration than the vanilla model in (iii), making the model act overall less overconfident, with the highest improvement being at the [0.6, 0.7] confidence interval. Fig. 5-10 (v) and (vi) visualize the evolution of the $pedestrian$ class, where the Depth Scaling improves once again over the vanilla model, best seen at lower confidence values but also between [0.4, 0.7]. Therefore, all these reliability diagrams are visually confirming the improvements of the ECE values, showed class-wise inside of the plots but also in Fig. 5-3 and Fig. 5-4.



(i) Vanilla

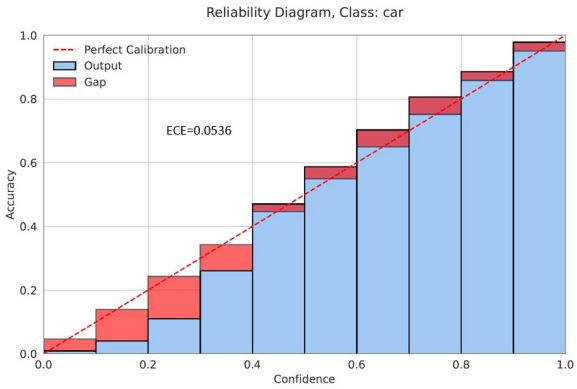


(ii) $\text{DE}_{N=4}$

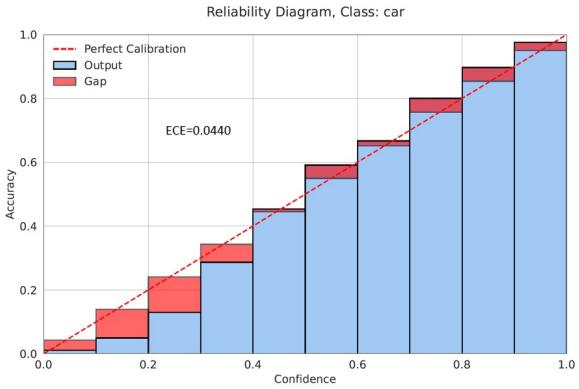


(iii) Depth Scaling

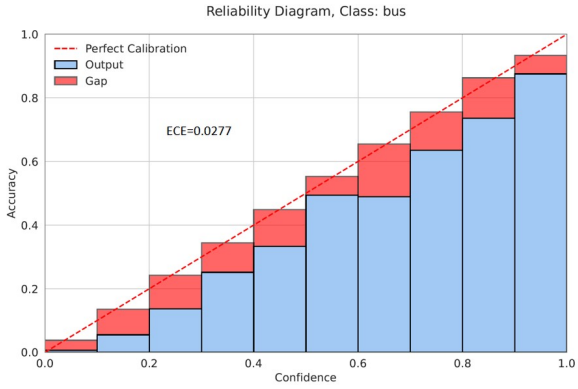
Fig. 5-9: Residual plot comparison between (i) Vanilla, (ii) $\text{DE}_{N=4}$ and (iii) Depth Scaling, that presents separately the x and y dimension and also their combined effect.



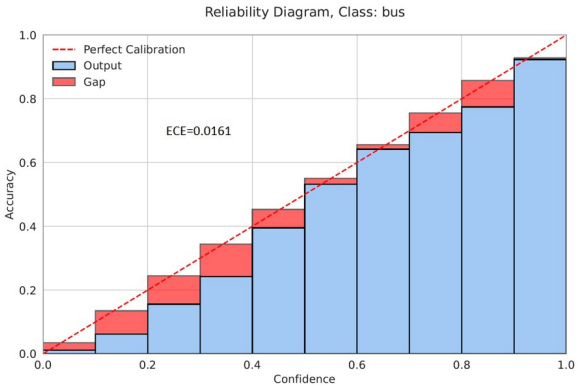
(i) Vanilla



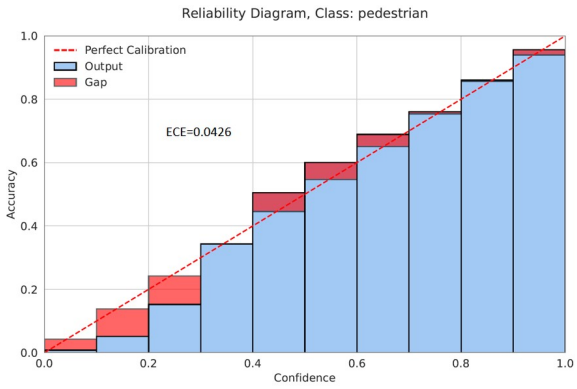
(ii) Depth Scaling



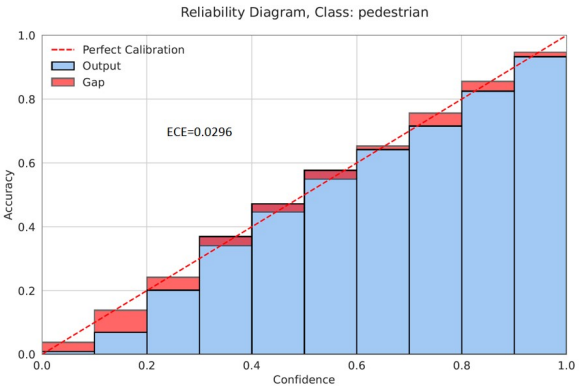
(iii) Vanilla



(iv) Depth Scaling



(v) Vanilla



(vi) Depth Scaling

Fig. 5-10: Reliability diagram comparison between the car, bus and $pedestrian$ classes of the vanilla model and the overall best calibrated configuration, Depth Scaling.

5.6 Uncertainty Visualization

To better understand visually the effects of the calibration techniques presented previously, two Bird's Eye View scenes from NuScenes were created. Each scene scene is presented twice, once with the output of the vanilla model and once with the output of the Depth Scaling configuration. Fig. 5-11 and Fig. 5-12 show the visualization of the vanilla and Depth Scaling experiments for the first presented frame. The predicted bounding boxes are depicted in blue, while the ground truths in green. Additionally, each detection has its predicted classification score displayed in black and its regression uncertainty, which is derived from the covariance matrix, is illustrated with a red ellipse. For this first example, one can observe that, in the case of the vanilla model in Fig. 5-11, the ellipses are very small, seen almost as dots. This denotes high certainty, because the model believes that all of its predictions are accurate, but this is not entirely true, because many predictions are clearly incorrect, and thus, their regression uncertainty should be higher. Best example is the predicted bounding box at the top of Fig. 5-11 that has a score of 0.48. Its covariance matrix has low values meaning that the model thinks it is a certain prediction, but one can clearly observe, that the ground truth box is few meters behind it. Looking at the same bounding box in Fig. 5-12, the red ellipse appears much bigger, which aligns with the reality, that the model should be less confident in its prediction, given that it might be wrong. Similar behavior happens also with the many other predicted bounding boxes in the bottom side of Fig. 5-11 and Fig. 5-12. It should be mentioned that these plots are only displaying predictions with a confidence score of above 0.25, to make the visualization clearer. Regarding classification calibration, one can see that Fig. 5-12 has less predictions displayed than the vanilla model in Fig. 5-11, because the calibration techniques helped scale down the overconfidence of erroneous or less accurate predictions. Conversely, the scores show improvement for the case of underconfident detections.

The second example frame is illustrated in Fig. 5-13, which depicts the vanilla model and in Fig. 5-14, which shows the Depth Scaling configuration, that is calibrated for both the regression and classification uncertainty. First of all, there are a lot less objects in this scene, but once again, one can observe that in Fig. 5-14, the Depth Scaling configuration has bigger covariance ellipses to lower the overconfidence in the regression prediction that the vanilla model shows in Fig. 5-13. Regarding classification uncertainty, the confidence scores are also adapted to better match their true outcome. It is best exemplified looking at the farthest object in this scene, in Fig. 5-13 having two predictions but only one in Fig. 5-14, since the second prediction was overconfident and was scaled down under the score threshold of 0.25.

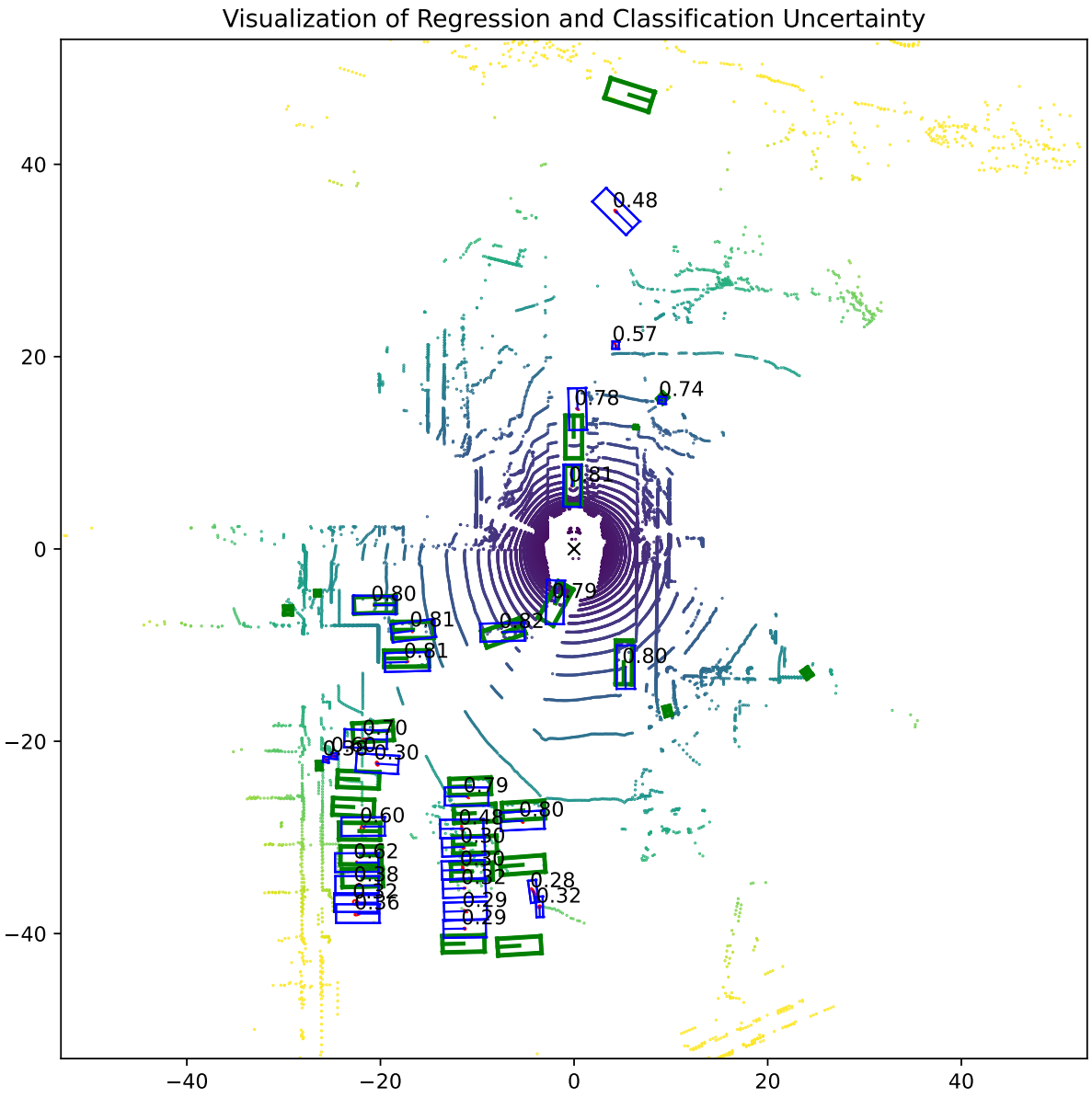


Fig. 5-11: Bird's Eye View visualization of a NuScenes frame using the **vanilla** model. The regression predictions are shown in blue boxes together with their classification scores and the covariance matrix of each bounding box is drawn with a red ellipse. The ground truths are plotted in green.

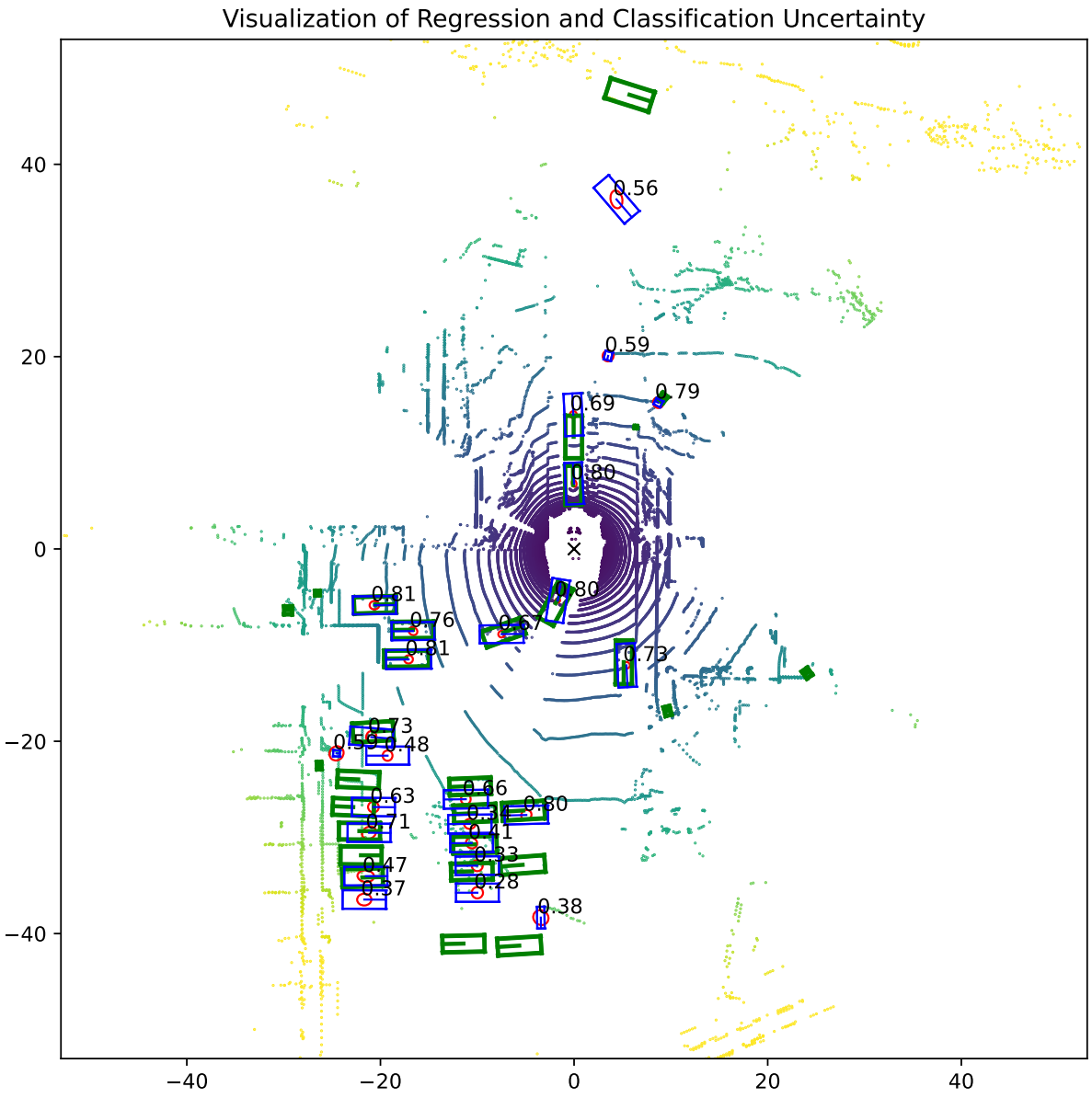


Fig. 5-12: Bird's Eye View visualization of a NuScenes frame using the **Depth Scaling** configuration. The regression predictions are shown in blue boxes together with their classification scores and the covariance matrix of each bounding box is drawn with a red ellipse. The ground truths are plotted in green.

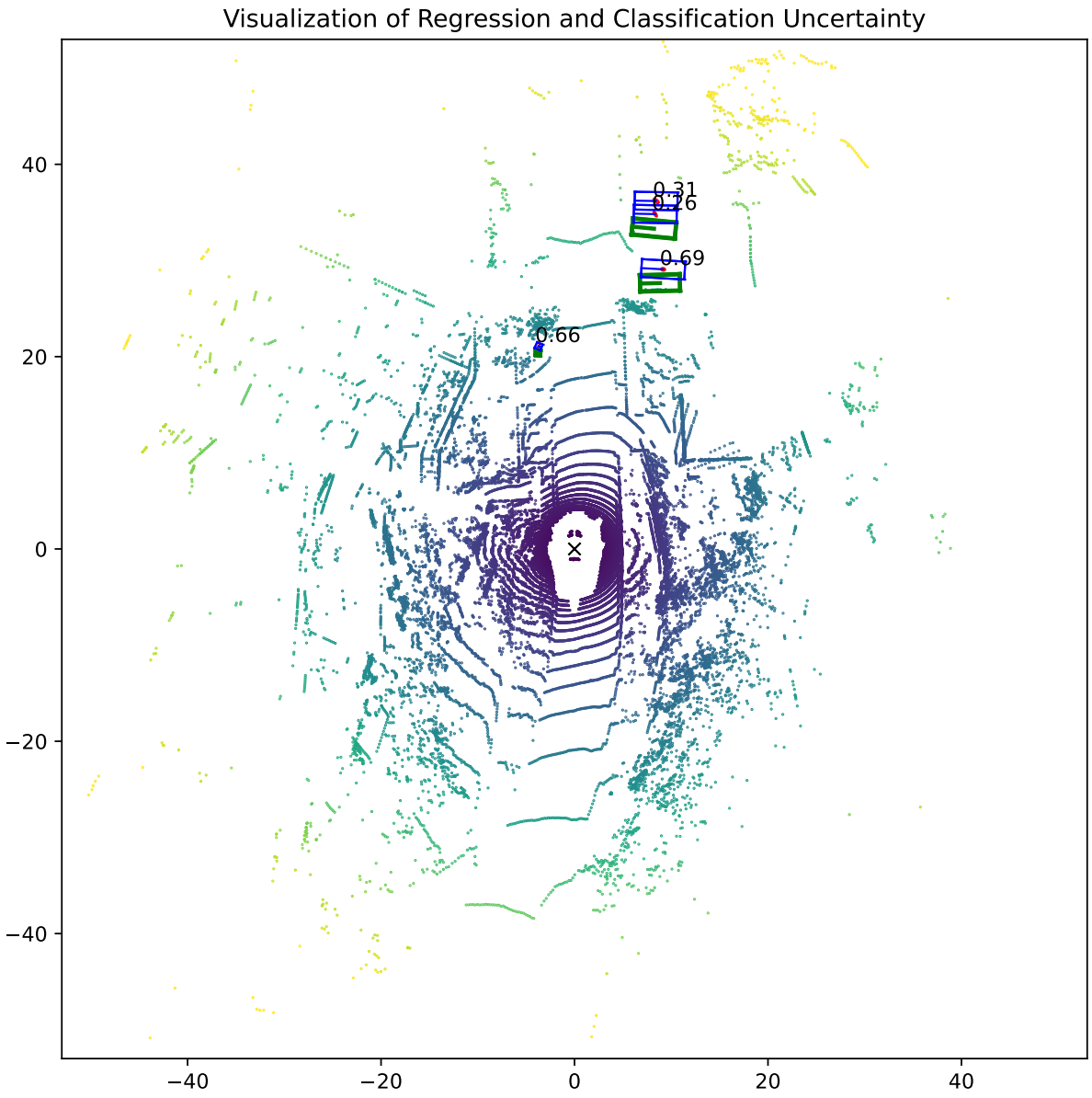


Fig. 5-13: Bird's Eye View visualization of a NuScenes frame using the **vanilla** model. The regression predictions are shown in blue boxes together with their classification scores and the covariance matrix of each bounding box is drawn with a red ellipse. The ground truths are plotted in green.

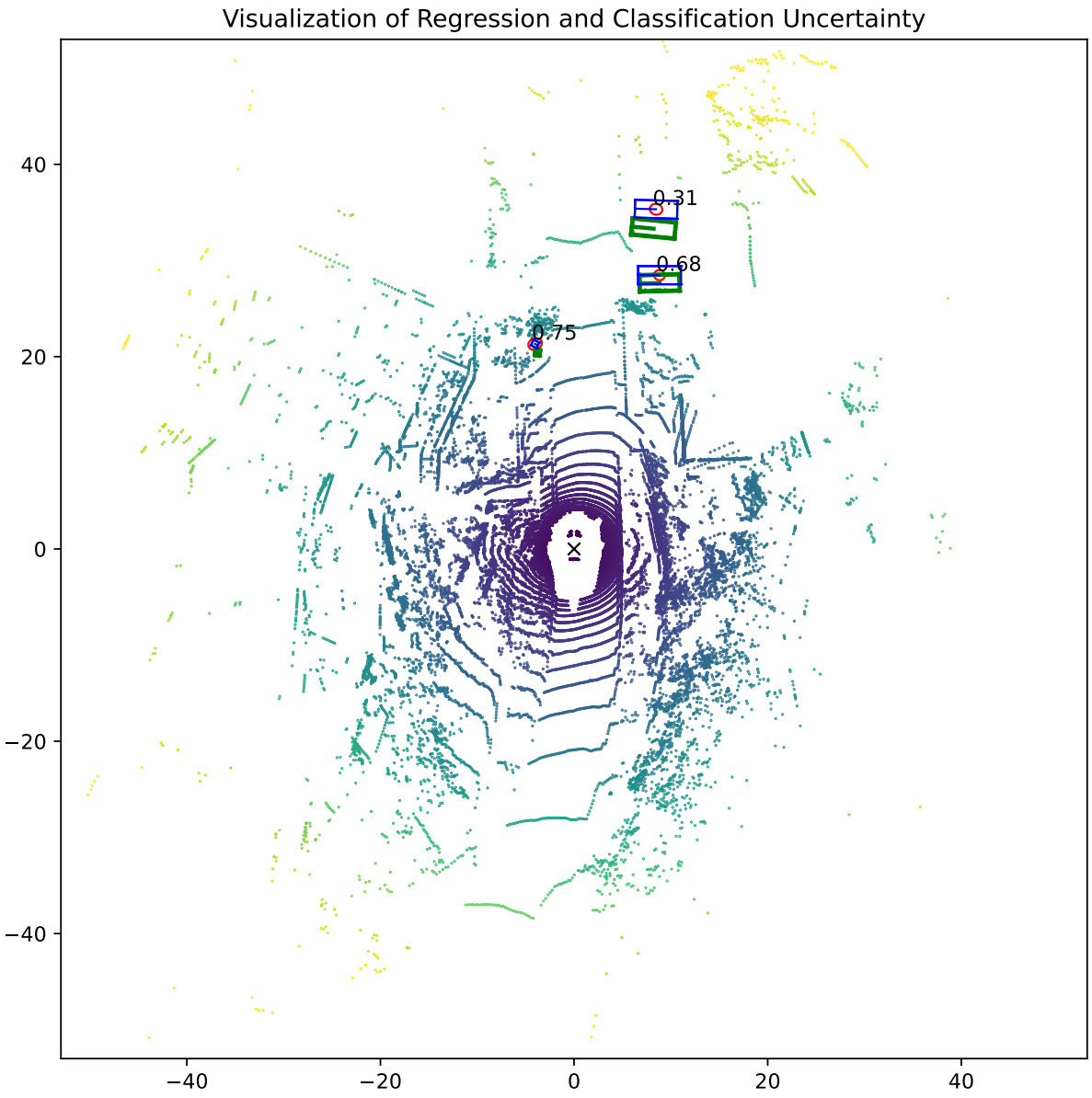


Fig. 5-14: Bird's Eye View visualization of a NuScenes frame using the **Depth Scaling** configuration. The regression predictions are shown in blue boxes together with their classification scores and the covariance matrix of each bounding box is drawn with a red ellipse. The ground truths are plotted in green.

5.7 Discussion

This section will discuss the previously presented results based on the two questions with which this chapter was opened, which are: what type of sensor has the best overall performance and what calibration technique produces better model calibration for the least requirements.

Addressing the first question, it should be mentioned that the comparison between the sensor cannot be very straightforward since the camera and the LiDAR and the radar experiments come from two separate datasets that were realized in different circumstances.

Regarding the View-of-Delft dataset, Fig. 5-1 shows the head-to-head comparison between the one radar scan, five radar scans and LiDAR experiments, broken down into class-wise performance and mAP. This table proves the hypothesis that accumulating consecutive radar scans to form richer point clouds and to gather an additional temporal feature results into an increase in performance and accuracy for the 3D detections. This improvement can be seen for each class in particular between the five scans and the one scan experiments. Moreover, [PAL22] showed in their results that the addition of the third spatial dimension, the elevation z, achieved much better mAP than when using only two spatial dimensions, therefore the new generation of "3+1D" radar showcasing significant growth and potential compared to what has been used in the past. Nevertheless, the LiDAR sensor still outperforms any presented version of the radar, but the gap between these two has for sure been reduced. Regarding the performances of camera sensor, it cannot be directly compared to the other two sensors because of different datasets. However, generally speaking, the camera sensor should situate itself, performance-wise, in-between the radar and the LiDAR sensors. The most important advantage of the camera sensor is the ability to classify objects more accurately, while the other two radars are being better at measuring distances and estimating bounding boxes.

Discussing the presented calibration techniques, these can be divided into two categories, loss oriented and post-hoc calibration techniques. First of all, all four methods have shown to be reducing the classification (Temperature Scaling, Logits Modulation, Logits Mixing) or regression (Depth Scaling) uncertainty. Looking at the loss oriented techniques, Logits Modulation and Mixing, these require a base model to be trained having one of these methods implemented, and afterwards, its results of the inference can be evaluated. The full-on training requires significant computational resources, especially when done on large datasets. Conversely, the post-hoc techniques, Temperature Scaling and Depth Scaling, do not need any additional model training, if the scaling factors are already known, since these can used directly for the inference of a model. But if the temperature or depth parameters still need to be computed, then additional overhead is required for finding these, however significantly less than for the loss oriented mechanisms. For the classification calibration, the Logits Mixing and Modulation appear to produce the biggest effect in reducing the uncertainty of models, however, for the LiDAR experiments, the Temperature Scaling was outperforming the other two methods. Nevertheless, finding the best Logits Mixing configuration and using all presented classification calibration methods together yielded the least uncertain model for all three analyzed sensors. For the calibration of regression uncertainty, Depth Scaling can only be compared to the UQ baselines. Using much bigger $N's$ than the ones tested previously would theoretically close the gap to the results obtained by Depth Scaling, but this would come with great computational costs, while Depth Scaling adds almost no real inference overhead.

6 Conclusion & Outlook

In this thesis, uncertainty estimates for transformer-based camera-, radar- and LiDAR- 3D object detectors have been implemented and investigated with the scope of improving model calibration for safety critical environments. This work tried to address the lack of academic research for the 3D realm of uncertainty quantification for the case of autonomous driving. Its goals were to analyze state of the art 3D capable architectures, such as PETR and PillarFormer, design an UQ baseline formed from several techniques and implement and test various calibration methods, both for classification and regression tasks, to assess how the uncertainty of decision making can be quantified and calibrated.

First of all, a brief introduction into the field of AI, ML and DL was conducted to build up a concrete foundation for the upcoming concepts. Next, the 3D Object Detection fundamentals were discussed to understand what 3D bounding boxes are and how these differ from the well-studied 2D case. Furthermore, an overview on the functionality, strengths and weaknesses of the main three types of sensors used in safety-critical environments was presented, followed by an introduction of the two types of uncertainty, aleatoric and epistemic, in the field of AI. Here, popular baseline techniques for quantification of uncertainty were introduced, like the Deep Ensembles or the Monte-Carlo Dropout approach. The final theoretical step stone was to present relevant metrics, such as the Expected Calibration Error and classification NLL, to estimate the classification uncertainty of a model, and for the regression task, metrics such as Mahalanobis Distance or regression NLL. To help put the computed metrics into perspective, visualization tools like the reliability diagram, the average calibration plot or the residual uncertainty plot were also covered. Additionally, the details of the two datasets, NuScenes and View-of-Delft, were presented, followed by the two 3D transformer-based architectures, PETR and PillarFormer.

Second of all, four calibration techniques, Temperature Scaling, Logits Modulation and Logits Mixing for classification calibration and Depth Scaling for regression calibration, were implemented and a solid UQ baseline was designed for comparison. The goal was to assess how the classification and regression calibration methods compare and improve to the vanilla model and the designed baseline. This thesis divided the experiments into three categories: camera-, radar- and LiDAR-driven results. Each experiment consisted of the vanilla result, the baseline composed of MCD and DE with multiple $N's$ and several different calibration configuration, for finding and assessing the best performing setup. For the radar experiments, the results were gathered from two datasets, consisting of one radar scan and five successive radar scans. The accumulation of multiple radar scans together with the utilization of a 3D radar sensor proved to be effective, increasing considerably the detection performance and accuracy. Nevertheless, the LiDAR still outperformed the accuracy of the radar. Since the camera data was taken from the NuScenes dataset, its performance cannot be directly compared to the other two sensors, but theoretically, it should lie in-between these two. After analyzing all the experiments, this thesis showed that all implemented calibration techniques, Temperature Scaling, Logits Modulation, Logits Mixing and Depth Scaling, created better calibrated models, aiding these with making less uncertain predictions to further improve the reliability and robustness of 3D object detectors. These four calibration techniques were divided into two further categories, loss oriented, which are the Logits Mixing and Modulation approaches, that require additional training,

and post-hoc methods, Temperature and Depth Scaling, which are simple yet effective methods, that can be directly applied on an already trained model.

For future work, in the context of uncertainty quantification for 3D Object Detection, it would be highly interesting to implement a sensor fusion approach, especially between the radar and LiDAR features, but also in a combination with camera. Such an approach should hypothetically combine the strengths of all these sensor types together, making the predictions of a model even more reliable and robust. Additionally, since there is almost no research on regression calibration, 2D or 3D, this work could be extended by implementing a loss oriented regression calibration technique, which could be used together with Depth Scaling to lower the regression uncertainty even more.

In conclusion, this thesis has contributed to the field of autonomous driving with a comprehensive analysis of uncertainty estimates for 3D Object Detection by creating an uncertainty quantification baseline and implementing various calibration techniques to further improve the reliability of the decision making process and to increase the trust in such safety-critical applications.

7 List of Symbols

B_m	set of samples in m-th bin
M	number of bins
N	number of ensembles or forward passes
O_D^L	final decoder layer
Q	query
Δf	Doppler shift
Σ	covariance matrix
σ	standard deviation
b	bias
c	speed of light
d	distance
t_d	time delay
u	uncertainty
v	velocity
w	weight
y_i	true label
x_{pred}	predicted bounding box
y_{gt}	ground truth bounding box

8 List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Average Precision
CNN	Convolutional Neural Network
DE	Deep Ensembles
DECE	Detection Expected Calibration Error
DETR	Detection Transformer
DL	Deep Learning
DS	Depth Scaling
DSE	Deep Sub-Ensembles
ECE	Expected Calibration Error
FPN	Feature Pyramid Network
IoU	Intersection over Union
LiDAR	Light Detection and Ranging
mAP	Mean Average Precision
MCD	Monte-Carlo Dropout
MD	Mahalanobis Distance
ML	Machine Learning
MSE	Mean Squared Error
NDS	NuScenes Detection Score
NLL	Negative Log-Likelihood
NLP	Natural Language Processing
NN	Neural Network
PETR	Positional Embedding Transformation
Radar	Radio Detection and Ranging
RCS	Radar Cross Section
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SECOND	Sparsely Embedded Convolutional Detection

TS	Temperature Scaling
UQ	Uncertainty Quantification
ViT	Vision Transformer
VOD	View-of-Delft
WBF	Weighted Boxes Fusion

9 Bibliography

[BEN17] BENGIO, Y., GOODFELLOW, I., COURVILLE, A.
Deep learning
Vol. 1, MIT press Cambridge, MA, USA, 2017

[BRÖ07] BRÖCKER, J., SMITH, L. A.
Increasing the reliability of reliability diagrams
Weather and forecasting 22.3 (2007), pp. 651–661

[CAE20] CAESAR, H., BANKITI, V., LANG, A. H., VORA, S., LIONG, V. E., XU, Q., KRISHNAN, A., PAN, Y., BALDAN, G., BEIJBOM, O.
nuscenes: A multimodal dataset for autonomous driving
Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2020, pp. 11621–11631

[CHO21] CHOLLET, F.
Deep learning with Python
Simon and Schuster, 2021

[CHU21] CHUNG, Y., CHAR, I., GUO, H., SCHNEIDER, J., NEISWANGER, W.
Uncertainty toolbox: an open-source library for assessing, visualizing, and improving uncertainty quantification
arXiv preprint arXiv:2109.10254 (2021)

[DEN20] DENG, D.
DBSCAN clustering algorithm based on density
2020 7th international forum on electrical engineering and automation (IFEEA), IEEE, 2020, pp. 949–953

[DOS20] DOSOVITSKIY, A.
An image is worth 16x16 words: Transformers for image recognition at scale
arXiv preprint arXiv:2010.11929 (2020)

[DU22] DU, X., WANG, Z., CAI, M., LI, Y.
Vos: Learning what you don't know by virtual outlier synthesis
arXiv preprint arXiv:2202.01197 (2022)

[GAL16] GAL, Y., GHAHRAMANI, Z.
Dropout as a bayesian approximation: Representing model uncertainty in deep learning
international conference on machine learning, PMLR, 2016, pp. 1050–1059

[GEG92] GEGENFURTNER, K. R.
PRAXIS: Brent's algorithm for function minimization
Behavior Research Methods, Instruments, & Computers 24 (1992), pp. 560–564

[GRO23] GROEFSEMA, S.
Uncertainty Quantification in DETR for Pedestrian Detection
PhD thesis, 2023

[GRO13] GROSSBERG, S.
Recurrent neural networks
Scholarpedia 8.2 (2013), p. 1888

[GU18] GU, J., WANG, Z., KUEN, J., MA, L., SHAHROUDY, A., SHUAI, B., LIU, T., WANG, X., WANG, G., CAI, J., et al.
Recent advances in convolutional neural networks
Pattern recognition 77 (2018), pp. 354–377

[GUO17] GUO, C., PLEISS, G., SUN, Y., WEINBERGER, K. Q.
On calibration of modern neural networks
International conference on machine learning, PMLR, 2017, pp. 1321–1330

[HAV20] HAVASI, M., JENATTON, R., FORT, S., LIU, J. Z., SNOEK, J., LAKSHMINARAYANAN, B., DAI, A. M., TRAN, D.
Training independent subnetworks for robust prediction
arXiv preprint arXiv:2010.06610 (2020)

[HE19] HE, Y., ZHU, C., WANG, J., SAVVIDES, M., ZHANG, X.
Bounding box regression with uncertainty for accurate object detection
Proceedings of the ieee/cvf conference on computer vision and pattern recognition, 2019, pp. 2888–2897

[KIN14] KINGMA, D. P.
Adam: A method for stochastic optimization
arXiv preprint arXiv:1412.6980 (2014)

[KRE11] KRENKER, A., BEŠTER, J., KOS, A.
Introduction to the artificial neural networks
Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech (2011), pp. 1–18

[LAK17] LAKSHMINARAYANAN, B., PRITZEL, A., BLUNDELL, C.
Simple and scalable predictive uncertainty estimation using deep ensembles
Advances in neural information processing systems 30 (2017)

[LI22] LI, Y., MAO, H., GIRSHICK, R., HE, K.
Exploring plain vision transformer backbones for object detection
European conference on computer vision, Springer, 2022, pp. 280–296

[LIU22] LIU, Y., WANG, T., ZHANG, X., SUN, J.
Petr: Position embedding transformation for multi-view 3d object detection
European Conference on Computer Vision, Springer, 2022, pp. 531–548

[MCL99] MCLACHLAN, G. J.
Mahalanobis distance
Resonance 4.6 (1999), pp. 20–26

[MED01] MEDSKER, L. R., JAIN, L., et al.
Recurrent neural networks
Design and Applications 5.64-67 (2001), p. 2

[MEI22] MEINHARDT, T., KIRILLOV, A., LEAL-TAIXE, L., FEICHTENHOFER, C.
Trackformer: Multi-object tracking with transformers
Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2022, pp. 8844–8854

[MIL07] MILLS-TETTEY, G. A., STENTZ, A., DIAS, M. B.
The dynamic hungarian algorithm for the assignment problem with changing costs
Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-07-27 (2007)

[MIS21] MISRA, I., GIRDHAR, R., JOULIN, A.
An end-to-end transformer model for 3d object detection
Proceedings of the IEEE/CVF international conference on computer vision, 2021, pp. 2906–2917

[MUK21] MUKHOTI, J., AMERSFOORT, J. van, TORR, P. H., GAL, Y.
Deep deterministic uncertainty for semantic segmentation
arXiv preprint arXiv:2111.00079 (2021)

[MUN24] MUNIR, M. A., KHAN, S. H., KHAN, M. H., ALI, M., SHAHBAZ KHAN, F.
Cal-DETR: calibrated detection transformer
Advances in neural information processing systems 36 (2024)

[OSH15] O'SHEA, K.
An introduction to convolutional neural networks
arXiv preprint arXiv:1511.08458 (2015)

[PAL22] PALFFY, A., POOL, E., BARATAM, S., KOOIJ, J. F., GAVRILA, D. M.
Multi-class road user detection with 3+ 1D radar in the View-of-Delft dataset
IEEE Robotics and Automation Letters 7.2 (2022), pp. 4961–4968

[PIT22] PITROPOV, M., HUANG, C., ABDELZAD, V., CZARNECKI, K., WASLANDER, S.
LiDAR-MIMO: efficient uncertainty estimation for LiDAR-based 3D object detection
2022 IEEE Intelligent Vehicles Symposium (IV), IEEE, 2022, pp. 813–820

[REZ19] REZATOFIGHI, H., TSOI, N., GWAK, J., SADEGHIAN, A., REID, I., SAVARESE, S.
Generalized intersection over union: A metric and a loss for bounding box regression
Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, 2019, pp. 658–666

[ROZ20] ROZA, F. S., HENNE, M., ROSCHER, K., GÜNNEMANN, S.
Assessing Box Merging Strategies and Uncertainty Estimation Methods in Multimodel Object Detection
Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16, Springer, 2020, pp. 3–10

[RUS16] RUSSELL, S. J., NORVIG, P.
Artificial intelligence: a modern approach
Pearson, 2016

[SIC21] SICKING, J., AKILA, M., PINTZ, M., WIRTZ, T., FISCHER, A., WROBEL, S.
A novel regression loss for non-parametric uncertainty optimization
arXiv preprint arXiv:2101.02726 (2021)

[SIM14] SIMONYAN, K., ZISSERMAN, A.
Very deep convolutional networks for large-scale image recognition
arXiv preprint arXiv:1409.1556 (2014)

[SOL21] SOLOVYEV, R., WANG, W., GABRUSEVA, T.
Weighted boxes fusion: Ensembling boxes from different object detection models
Image and Vision Computing 107 (2021), p. 104117

[STR21] STRUDEL, R., GARCIA, R., LAPTEV, I., SCHMID, C.
Segmenter: Transformer for semantic segmentation
Proceedings of the IEEE/CVF international conference on computer vision, 2021, pp. 7262–7272

[VAL19] VALDENEGRO-TORO, M.
Deep sub-ensembles for fast uncertainty estimation in image classification
arXiv preprint arXiv:1910.08168 (2019)

[VAS17] VASWANI, A.
Attention is all you need
Advances in Neural Information Processing Systems (2017)

[YAN20] YANG, L., SHAMI, A.
On hyperparameter optimization of machine learning algorithms: Theory and practice
Neurocomputing 415 (2020), pp. 295–316

[ZHU20] ZHU, X., SU, W., LU, L., LI, B., WANG, X., DAI, J.
Deformable detr: Deformable transformers for end-to-end object detection
arXiv preprint arXiv:2010.04159 (2020)

10 Appendix

10.1 Framework

MMDetection3D

The work of this thesis has been done using the MMDetection3D toolbox, which is built on top of MMDetection framework and it is specifically designed for 3D detection tasks. The core features of MMDetection3D are multi-modal sensor support, a model zoo, with lots of popular architectures and pre-trained models, extensibility, which allows users to extend the framework with other architectures, datasets or metrics and finally, modular design, which enables users to easily swap and configure different architecture components through config files.

For the purpose of this thesis, the NuScenes dataset and the PETR architecture were included by default in this toolbox, while the View-of-Delft dataset and the PointFormer architecture had been manually integrated.

The workflow of this toolbox consists of multiple stages. Data preparation is the first step, which can include dataset specific preprocessing and loading of ground truth annotations. Secondly, the components of the model are built, which can include encoders, decoders, backbones, necks, heads, losses and so on. These can be then further customized by a multitude of parameters. As next, the training or testing is performed on the chosen dataset and with the customized modules. Finally, MMDetection3D has an in-built metric system, tailored for each dataset, that can evaluate the performance trained models on different metrics, including ones for uncertainty estimation.

Configuration Files

The core of working with MMDetection3D lies on creating, editing and using its config files. These configuration script are responsible for setting the entire environment up, starting from the actual model, to data-loader, evaluator, training, testing and optimization configs. Listing 10.1 shows a portion of a model setup from a PETR config file. In this example, the whole model is represented by a dictionary. The type keyword significates the module or submodule that MMDetection3D shall use to build that model. This config starts by specifying the main architecture, PETR, and afterwards, initializing further modules, as in the case of the data preprocessor, backbone and neck or submodules, as in the case of the head. This example of head module is of class PETRHead, it initializes the whole decoder side of the Vision Transformer as well as the bounding box coder, the positional encoding used and the losses used to train it. Lastly, each model needs to have a train configuration for the hyperparameters of its component modules.

```
1     model = dict(
2         type='PETR',
3         data_preprocessor=dict(
4             type='Det3DDataPreprocessor',),
5         use_grid_mask=True,
6         img_backbone=dict(
7             type='VoVNetCP',),
8         img_neck=dict(
9             type='CPFPN', in_channels=[768, 1024], out_channels=256, num_outs=2)
    ,
10        pts_bbox_head=dict(
11            type='PETRHead',
```

```
            transformer=dict(
                type='PETRTransformer',
                decoder=dict(
                    type='PETRTransformerDecoder',
                    return_intermediate=True,
                    num_layers=6,
                    transformerlayers=dict(
                        type='PETRTransformerDecoderLayer',
                        attn_cfgs=[
                            dict(
                                type='MultiheadAttention',
                                embed_dims=256,
                                num_heads=8,
                                attn_drop=0.1,
                                dropout_layer=dict(type='Dropout', drop_prob
    =0.1)),
                            dict(
                                type='PETRMultiheadAttention',
                                embed_dims=256,
                                num_heads=8,
                                attn_drop=0.1,
                                dropout_layer=dict(type='Dropout', drop_prob
    =0.1)),
                        ],
                        feedforward_channels=2048,
                        ffn_dropout=0.1,
                        operation_order=('self_attn', 'norm', 'cross_attn', '
    norm',
                                         'ffn', 'norm')),
                )),
            bbox_coder=dict(
                    type='NMSFreeCoder',
                    post_center_range=[-61.2, -61.2, -10.0, 61.2, 61.2, 10.0],
                    pc_range=point_cloud_range,
                    max_num=300,
                    voxel_size=voxel_size,
                    num_classes=10),
            positional_encoding=dict(
                type='SinePositionalEncoding3D', num_feats=128, normalize=True),
            loss_cls=dict(
                type='mmdet.FocalLoss',
                use_sigmoid=True,
                gamma=2.0,
                alpha=0.25,
                loss_weight=2.0),
            loss_bbox=dict(type='mmdet.L1Loss', loss_weight=0.25),
            loss_iou=dict(type='mmdet.GIoULoss', loss_weight=0.0)),
        train_cfg=dict(
            pts=dict(
                grid_size=[512, 512, 1],
                voxel_size=voxel_size,
                point_cloud_range=point_cloud_range,
                out_size_factor=4,
                assigner=dict(
                    type='HungarianAssigner3D',
                    cls_cost=dict(type='FocalLossCost', weight=2.0),
                    reg_cost=dict(type='BBox3DL1Cost', weight=0.25),
                    iou_cost=dict(
                        type='IoUCost', weight=0.0),
                    pc_range=point_cloud_range))))
```

Listing 10.1: Snippet of a config file for the PETR architecture