TUM School of Computation, Information and Technology
Technical University of Munich

Master’s Thesis in Informatics

Deep Multi-Modal Cooperative 3D Object Detection of Traffic Participants Using Onboard and Roadside Sensors

Tiefe multimodale kooperative 3D-Objekterkennung von Verkehrsteilnehmern mit Hilfe von Fahrzeug- und Infrastruktursensorik

Supervisor	Prof. Dr.-Ing. habil. Alois C. Knoll
Advisor	Walter Zimmer, M.Sc.
Author	Gerhard Arya Wardana
Date	October 15, 2023 in Munich

Disclaimer

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

Munich, October 15, 2023

(Gerhard Arya Wardana)

Abstract

It is necessary to ensure that the deployed autonomous driving systems are as robust, reliable, and safe as possible. Most research today is focused on onboard vehicle autonomous driving. This approach inherently is weak against occlusion. Cooperative driving, combining vehicle and infrastructure sensors, alleviates this weakness. Camera-LiDAR fusion is used to cover the weaknesses of each sensor modality. However, this increases the amount of data that needs to be processed. As such, it is necessary to find a fusion method that provides the best trade-off between accuracy and speed. Motivated by this, this work proposes a deep, cooperative, camera-LiDAR fusion method in a unified bird's eye view perspective for 3D object detection. A new TUMTraf Cooperative Dataset was also created. This proposed deep cooperative multi-modal method provides the best performance over any other combination that doesn't use all cameras and LiDARs on both the vehicle and infrastructure for the TUMTraf Cooperative Dataset. The proposed method is also proven to be better in infrastructure-only, camera-LiDAR fusion mode than the InfraDet3D method for the TUMTraf Intersection Dataset. Finally, this work provides recommendations to improve the proposed method further in the future.

Zusammenfassung

Es muss sichergestellt werden, dass die eingesetzten autonomen Fahrsysteme so robust, zuverlässig und sicher wie möglich sind. Die meisten Forschungsarbeiten konzentrieren sich heute auf das autonome Fahren im Fahrzeug. Dieser Ansatz ist von Natur aus schwach gegenüber Verdeckungen. Kooperatives Fahren, bei dem Fahrzeug- und Infrastruktursensoren kombiniert werden, mildert diese Schwäche. Die Kamera-LiDAR-Fusion wird eingesetzt, um die Schwächen der einzelnen Sensormodalitäten auszugleichen. Dies erhöht jedoch die zu verarbeitende Datenmenge. Daher ist es notwendig, eine Fusionsmethode zu finden, die den besten Kompromiss zwischen Genauigkeit und Geschwindigkeit bietet. Aus diesem Grund wird in dieser Arbeit eine tiefe, kooperative Kamera-LiDAR-Fusionsmethode in einer einheitlichen Vogelperspektive zur 3D-Objekterkennung vorgeschlagen. Außerdem wurde ein neuer TUMTraf Cooperative Dataset erstellt. Die vorgeschlagene kooperative multimodale Methode bietet die beste Leistung im Vergleich zu allen anderen Kombinationen, die nicht alle Kameras und LiDARs sowohl am Fahrzeug als auch an der Infrastruktur verwenden, für den TUMTraf Cooperative Dataset. Die vorgeschlagene Methode erweist sich auch im reinen Infrastruktur-, Kamera-LiDAR-Fusionsmodus als besser als die InfraDet3D-Methode für den TUMTraf-Kreuzungsdatensatz. Schließlich gibt diese Arbeit Empfehlungen zur weiteren Verbesserung der vorgeschlagenen Methode in der Zukunft.

Contents

Chapter 1

Introduction

1.1 Motivation

The automotive industry is currently undergoing a boom. Statista [Sta22] forecasts that the number of autonomous vehicles with at least Level 2 capabilities will only increase with an increasing speed over time in the near future as shown in Figure 1.1. Research by S&P Global [Glo18] and McKinsey & Company [Com23] also suggest the same trend.

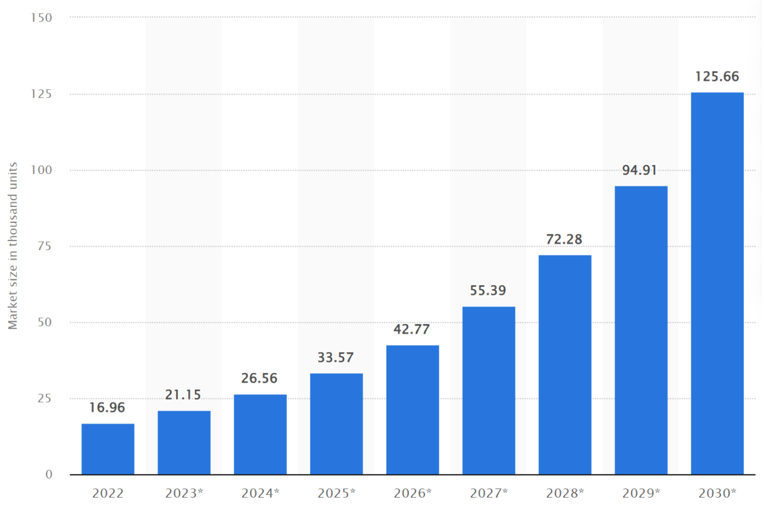


Figure 1.1: Statista estimate of the number of autonomous vehicles in 2022 with a forecast until 2030 [Sta22]. This figure shows a trend that autonomous vehicle adoption will increase at an increasing speed in the near future.

As a result, the field of autonomous driving has become a popular subject of research worldwide. Based on information aggregated using https://app.dimensions.ai/, the number of papers containing the phrase "autonomous driving" in their title or abstract was 638 in 2015. This number increased every year since, reaching 4475 in 2020, 5707 in 2021, and 6841 in 2022. While this popularity and rapid progress is generally a good thing, it must be ensured that any system deployed has to be as robust, reliable, and safe as possible. This is important if the industry wants to win and maintain public trust, which in turn is important for the future prospects of this field.

1.1.1 Cooperative Driving

Most of the research conducted today is focused on onboard autonomous driving, utilizing only the vehicle and the sensors plus the computation unit it carries. As a result, the performance, robustness, and safety of such systems are steadily improving. Unfortunately, such systems are inherently limited by the very characteristics of on-board systems. Some of these are occlusion, limited hardware power, and the lack of consensus with other traffic participants.

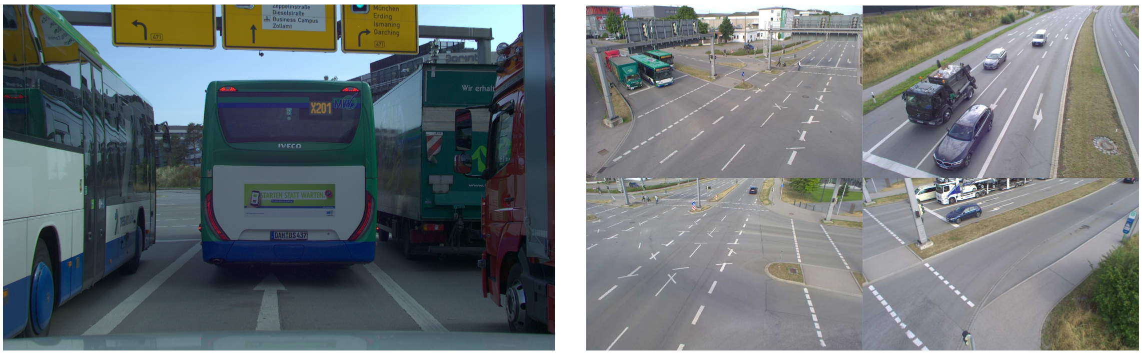


Figure 1.2: Occlusion problem from vehicle and infrastructure POV images. While the vehicle could not see anything beyond the three large vehicles in front of it, infrastructure based sensors could clearly see everything.

Due to its position in the environment, onboard sensors will often have their view blocked by objects or other traffic participants. As a result, the vehicles are practically blind to possible dangers behind these objects. Hardware limitation comes from the fact that a vehicle only has so much space and power that it can supply to the various sensors and computation units it might carry on board. Lack of consensus is caused by the fact that such on-board systems usually do not communicate with other traffic participants and have to react to whatever these other participants decide to do to ensure safety.

Cooperative driving, that is autonomous driving using both vehicle and infrastructure-based sensors, solves all three issues. Due to the fact that infrastructure sensors are usually mounted at a higher position than vehicle sensors, it is able to provide a better overview of the current space the vehicle is in. By combining them, it is possible to obtain the most complete representation possible, meaning now the vehicle will no longer be blind even if an object is occluding its view. This should improve reliability and safety.

Infrastructure sensors are also able to utilize more powerful computation units that demand more space and power than what a vehicle could provide. This should help the system to be able to run more reliably in real-time (minimum of 10 Hz) by tasking the vehicle with the lighter task of extracting features from its sensor data, before sending them to the infrastructure and having it combine the various features into a complete, larger feature, which can then be used for detection.

All vehicles that are connected to the network will also have the same understanding of the environment they are in thus decreasing the likelihood that each will have a misunderstanding because they can't see certain participants or objects in the environment that others can see. Additionally, future cooperative driving systems could also be able to assign each participant a path for the goal that they each want to achieve, effectively conducting the traffic in a certain area and synchronizing all participants. All of this should contribute to improving the safety of autonomous vehicles.

1.1.2 Multi-Modal Fusion

Based on existing research efforts [Bai+22a] [Li+22] [Liu+23b], it is also clear that multimodal systems are more robust, reliable, and offer better performance potential than systems that utilize only one modality. Each sensor type has its own characteristics, strengths, and

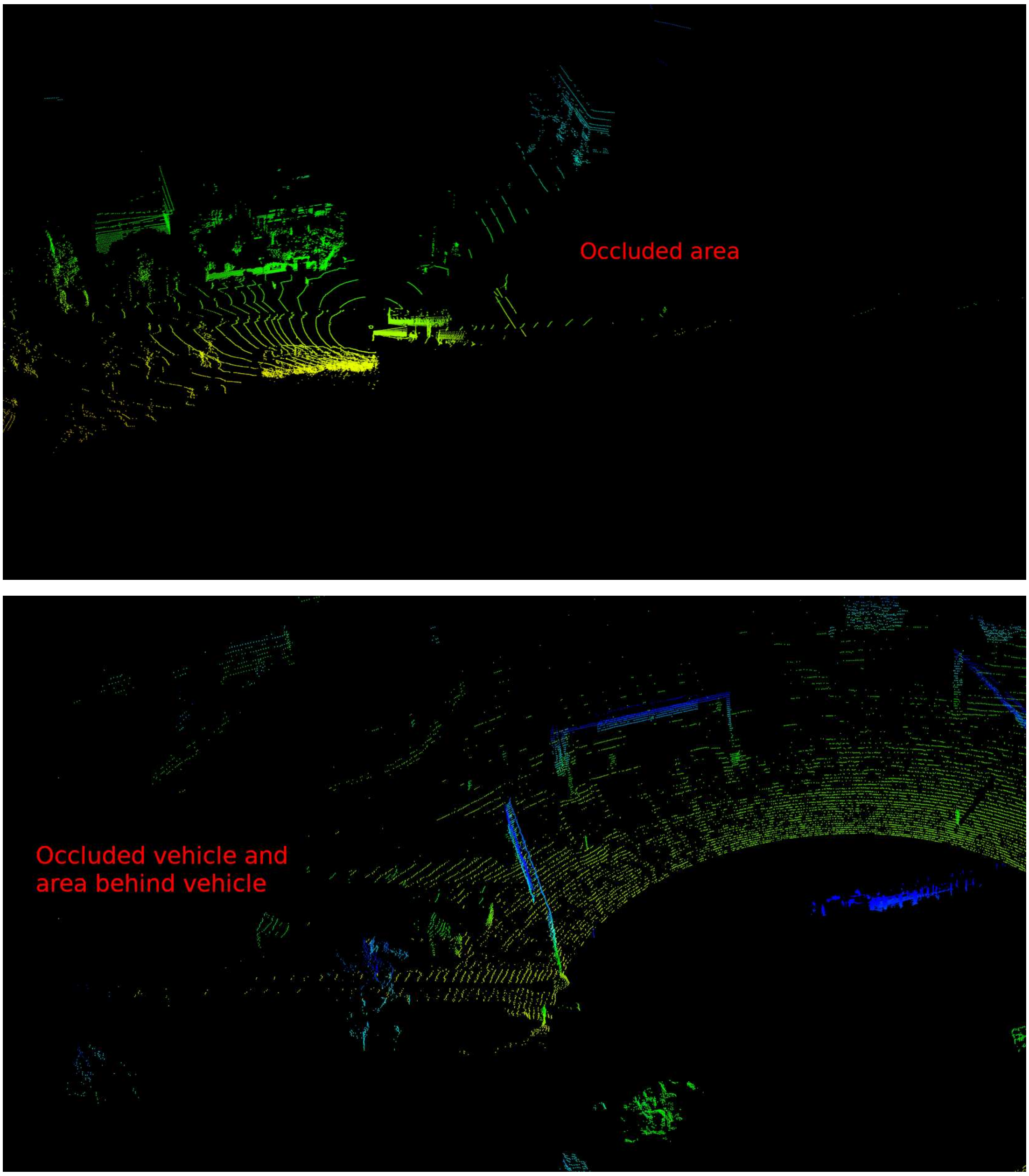


Figure 1.3: Occlusion problem from vehicle and infrastructure POV LiDAR point clouds. The vehicle point cloud contains a large area with no points beyond the three large vehicles, while the infrastructure point cloud contains no points in the area behind the three large vehicles, which includes the ego-vehicle.

weaknesses.

Fusing both modalities allows the sensors to cover for each other's weaknesses and adds redundancy, creating a more robust and reliable view of the environment. On top of that, since the infrastructure sensors are more likely to include LiDARs, even vehicles that don't have LiDARs will now be able to take advantage of LiDAR data and its characteristics to help cover for the weaknesses of its cameras. Plus, having more modalities results in better redundancy, allowing the system to still perform reasonably well, even if one modality is down or affected by environmental factors.

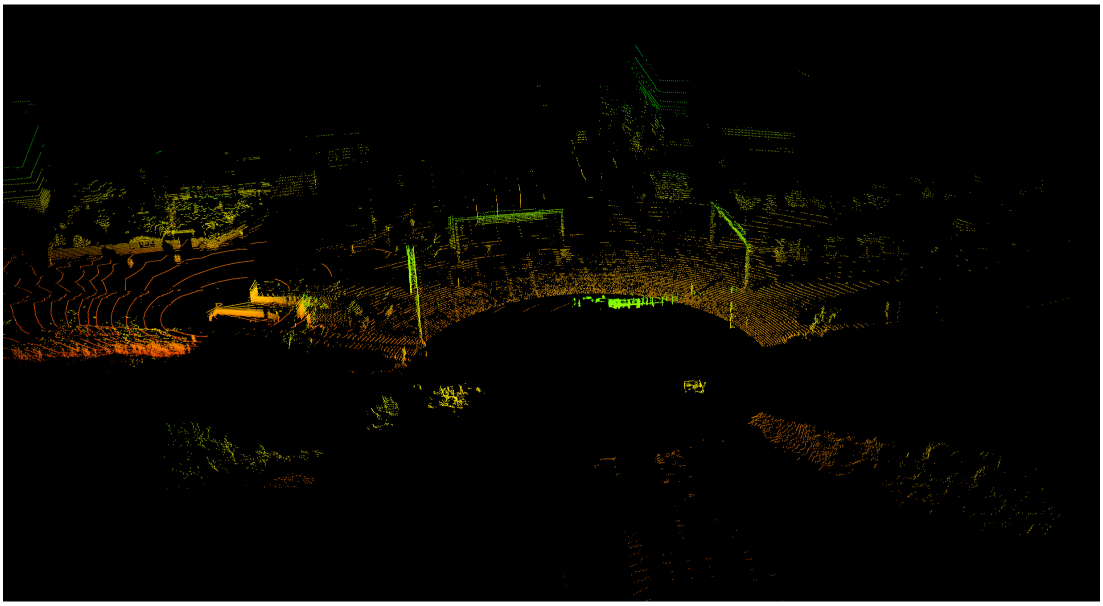


Figure 1.4: Fused Vehicle and Infrastructure LiDAR Point Cloud. By fusing both point clouds, a more complete view of the scene is achieved. Each individual point cloud provides information that the other lacked.

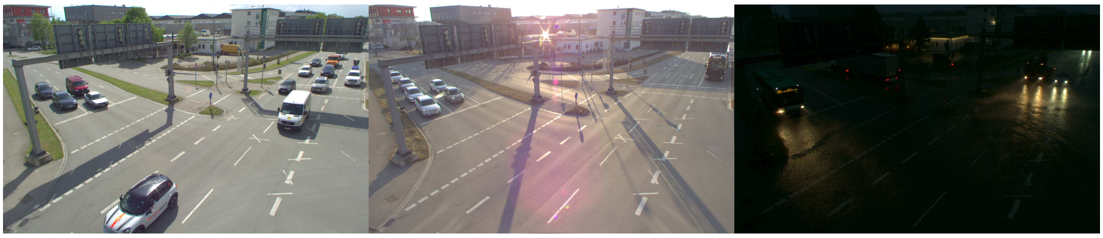


Figure 1.5: Left to right: image with good lighting, slightly too bright with lens flare, and too dark and with heavy rain.

1.1.3 Level of Fusion

For both types of fusion mentioned above, there are multiple levels of fusion that can be chosen for this work. These are early, deep, and late fusion. The method chosen in this work is deep fusion. The reason for this decision is the trade-off between the amount of data that needs to be processed compared to the amount of information lost by the time fusion happens, which influences performance.

This sweet spot offered by deep fusion is important due to the inherent bottleneck caused by bandwidth and latency limitations of V2X communication available today, which is required for cooperative driving. The system needs to run in real-time (at least 10 Hz) while meeting the goal of achieving high performance, robustness, and reliability. Because of this, deep fusion can be considered to be the best way forward for this particular problem.

This belief is also supported by previous survey papers written by Bai et. al. [Bai+22c] [Bai+22d] and Sun et. al. [Sun+22], with Bai et. al. specifically recommending deep fusion as the path forward for future research efforts.

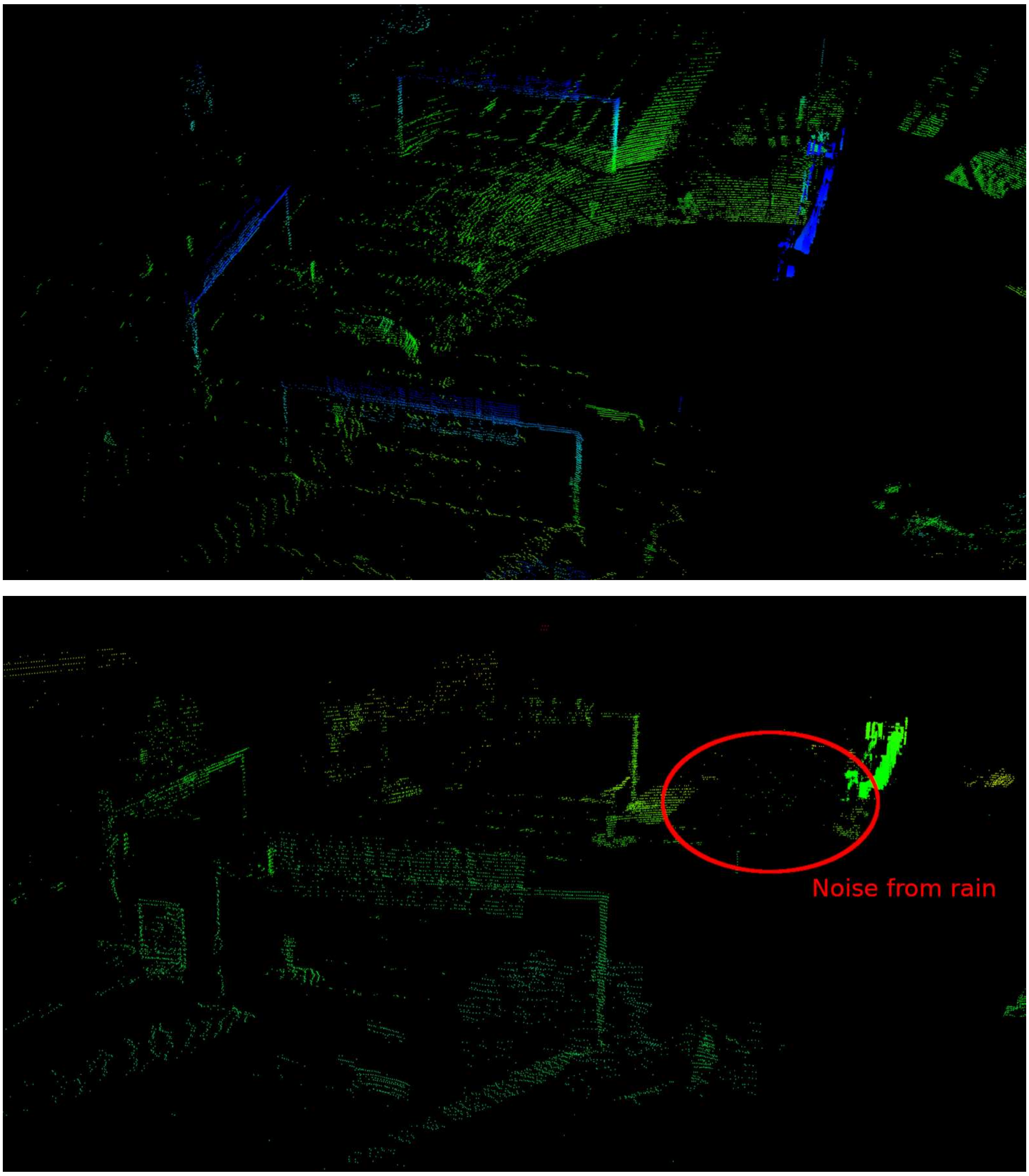


Figure 1.6: Point cloud from dry and rainy weather. Not only is the point cloud from rainy weather noisy, but it also contains ∼60% fewer points than the point cloud from dry weather.

1.2 Concepts and Terminology

In cooperative driving, there are certain concepts and terminologies that might not be commonly used elsewhere. This subsection aims to provide a definition and short explanation for each of these concepts and terminologies.

In this work, vehicle node(s) refers to any and all vehicles participating in cooperative driving. These vehicles carry their own set of sensors (camera and LiDAR) and their own onboard computation unit. Infrastructure nodes refer to sensors (camera and LiDAR) mounted on road infrastructure, such as traffic lights, road signage, road lamps, etc. These are then connected to a Road Side Unit (RSU) where the computation unit for this node is housed.

V2X communication means vehicle-to-everything communication. In this work's experiments, this is represented by V2I (vehicle-to-infrastructure) communication, where participant vehicle node(s) and the infrastructure nodes communicate wirelessly, either using commercial LTE networks, direct C-V2X (Cellular V2X) communication, or direct communication using WiFi.

1.3 Contribution

The contributions of this work are two-fold. First, a deep multi-modal cooperative 3D object detection method that utilizes data from a camera and LiDAR from a vehicle and a set of sensors mounted on a road infrastructure. This method utilizes an existing state-of-the-art method of deep camera-LiDAR fusion in a unified BEV perspective and expands it to allow it to work in cooperative mode, with different camera and LiDAR backbones, while running in real-time and using low enough VRAM to allow it to be trained and used for inference on mainstream GPUs. The second contribution is the creation of a new cooperative camera and LiDAR dataset named the TUMTraf Cooperative Dataset. This dataset was originally created for this work but will eventually become the next public release of the TUMTraf dataset family.

1.4 Proposed Method

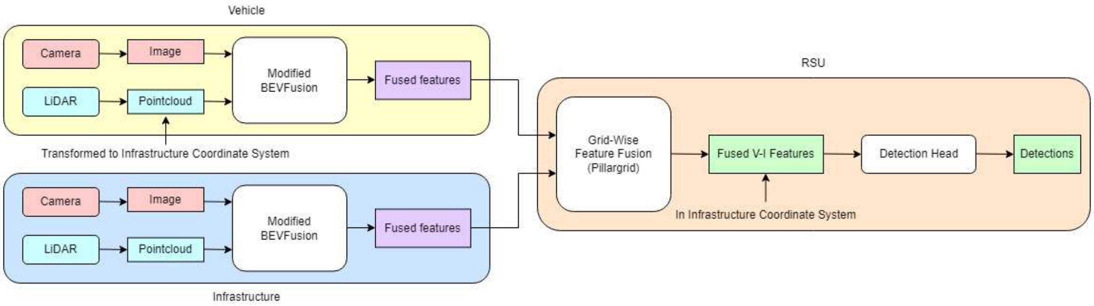


Figure 1.7: A general outline of our proposed method. The vehicle and infrastructure each run an instance of modified "headless" BEVFusion [Liu+23b] to fuse camera-LiDAR features. These are then fused infrastructure-side using a grid-wise feature fusion inspired by PillarGrid [Bai+22b] to create the final fused vehicle-infrastructure feature. A more detailed architecture of the proposed method is also provided later in fig. 4.5.

Based on the reasoning provided in the Motivation section, the general idea of the method proposed in this work is as follows: First, features are extracted from the camera and LiDAR data of the vehicle and infrastructure nodes separately. The LiDAR features of the vehicle node are extracted from a point cloud that is already transformed into the coordinate system of the infrastructure node. The camera features, LiDAR features, and other necessary information are then transmitted from the vehicle node to the infrastructure node.

On the infrastructure node, two fusion steps are performed. First, camera and LiDAR features from both the vehicle and infrastructure node are fused separately. This generates several fused camera-LiDAR features representing the perspective of each node. The second step is fusing these separate fused camera-LiDAR features into one global feature. This global feature is then used to perform 3D object detection. The results are then transmitted back to the vehicle node.

1.5 Goals

The cooperative configurations must be better than vehicle-only and infrastructure-only configurations to justify the extra amount of work a cooperative model needs to handle compared to the single-node model. This needs to be especially true in cases with heavy occlusion.

The camera-LiDAR fusion configurations of the proposed method should also perform better than the camera-only and LiDAR-only configurations on the data we use. Our metric for this is BEV and 3D mean average precision (mAP). The reason for this goal is to prove that using both modalities provides a meaningful improvement over using cameras or LiDARs only.

The proposed method must also perform better than the state-of-the-art infrastructure-only, camera-LiDAR fusion model running on the research group's toolchain: InfraDet3D [Zim+23a]. Doing this will prove that deep fusion is indeed a better approach compared to late fusion.

As mentioned earlier, the system also needs to run in real-time. This means that it runs at a minimum rate of 10 Hz. This number is also supported by the works of [Zha+18] and [Liu+22]. Both mention that cooperative driving systems are generally required to run at 10 Hz or more.

The final goal is to prove that the model can be trained on a high-end mainstream GPU and not only deep learning specialist GPUs. It needs to also be able to run on mainstream GPUs during inference. To show this, we will use VRAM usage during training and inference/testing as the metric.

1.6 Outline

The remaining chapters are structured as follows: Chapter 2 explains other works related to this work in more detail. Chapter3 explains in detail the dataset used in this work. Chapter 4 covers the implementation details of the proposed method. Chapter 5 covers the details of the experiments performed to collect data for quantitatively and qualitatively measuring the capabilities of the method proposed and lists the data collected. Chapter 6 analyzes this data and provides some interpretations. Chapter 7 provides recommendations for future works. Finally, Chapter 8 covers the conclusion reached by this work.

Chapter 2

Related Work

This chapter provides an overview of all works that are related to this work. The state-of-the-art methods that serve as both inspiration and baseline target for the proposed method of this work, will be presented in the next chapters.

To do this a taxonomy of methods that are related or adjacent to the method proposed in this work is created. Since the method is related to camera-LiDAR fusion and vehicle-infrastructure fusion, this is what this taxonomy of methods will be based on. The first group of methods only handles a single node (vehicle or infrastructure). This will be divided into camera-only, LiDAR-only, and camera-LiDAR fusion methods. Then the methods that handle both vehicle and infrastructure (cooperative) methods will be covered. They are then also divided in a similar manner to the single-node methods.

2.1 Image Object Detection

Machine learning-based image object detection was researched much earlier than its point cloud equivalent. Viola-Jones created a method for face detection in 2001 [VJ01] and Dalal et.al. created Histograms of Oriented Gradients (HOG) [DT05] for human detection in 2005. In the taxonomy, only methods that are either popular, relevant to the work, or state-of-the-art will be covered. They are going to be divided into CNN-based methods and transformer-based methods. Then, the backbones used by these methods to extract features from the images will also be covered.

2.1.1 Methods

One of the most popular methods of image-based object detection is Convolutional Neural Network (CNN). The model that popularized deep CNN was created in 2012 [KSH17] and it won the ImageNet Large Scale Visual Recognition Challenge of the year. This was the first time that a CNN was able to beat the performance of traditional methods.

Other popular CNN methods include the R-CNN family: R-CNN [Gir+14], Fast R-CNN [Gir15], and Faster R-CNN [Ren+15]. The general idea of R-CNN is to reduce the number of regions using the Selective Search algorithm to reduce it down to 2000 region proposals. These are then warped into rectangles and CNNs are applied to these regions to extract the features. The extracted features are then fed into an SVM to determine if an object exists inside this region followed by bounding box registration. However, this method is still heavy since it still needs to check 2000 regions, meaning it is also slow. On top of this, Selective Search is not a learning algorithm, which sometimes results in bad region proposals.

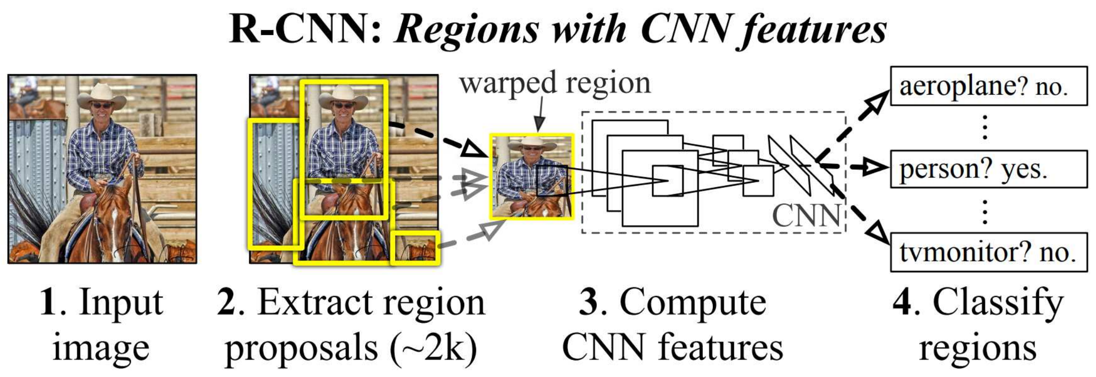


Figure 2.1: R-CNN Architecture from [Gir+14].

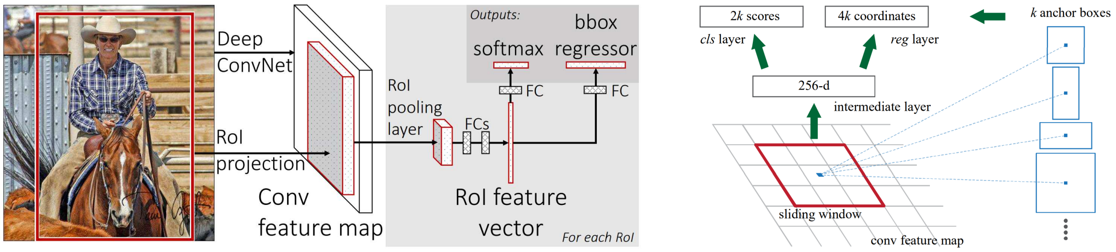


Figure 2.2: Fast R-CNN Architecture from [Gir15] and Region Proposal Network of Faster R-CNN [Ren+15].

Fast R-CNN [Gir15] was created in 2015 in order to fix this issue. Fast R-CNN applies the CNN only once instead of 2000 times. The CNN is applied to the whole image, creating a feature map. Region proposals are generated from this feature map, warped into rectangles, reshaped into a fixed shape by ROI pooling, and then used for detection. Faster R-CNN [Ren+15] was a follow-up in the same year that improved upon Fast R-CNN. It replaced the Selective Search algorithm with a Region Proposal Network (RPN). With this change, the whole network is trainable end-to-end, improving its performance. At the same time, RPN is also much faster than the Selective Search algorithm, making Faster R-CNN the first near-real-time deep learning-based object detector.

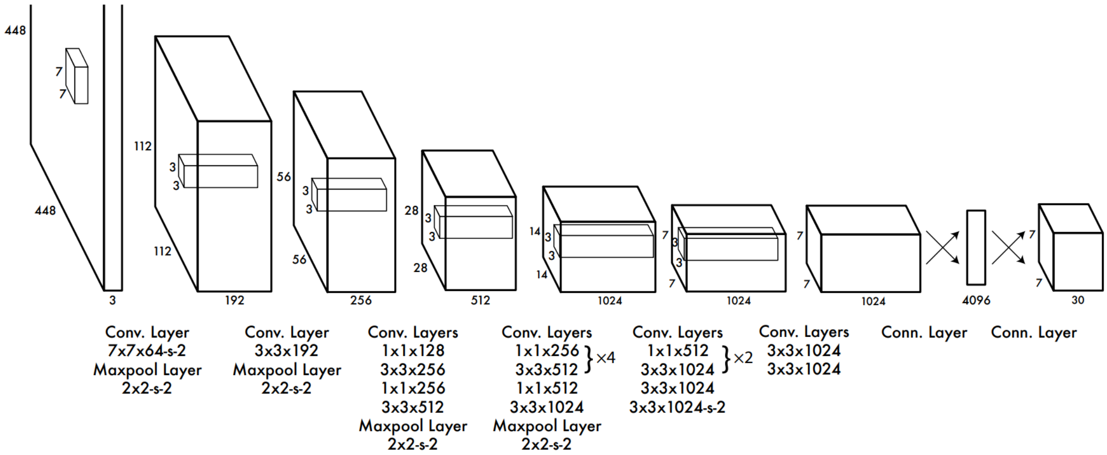
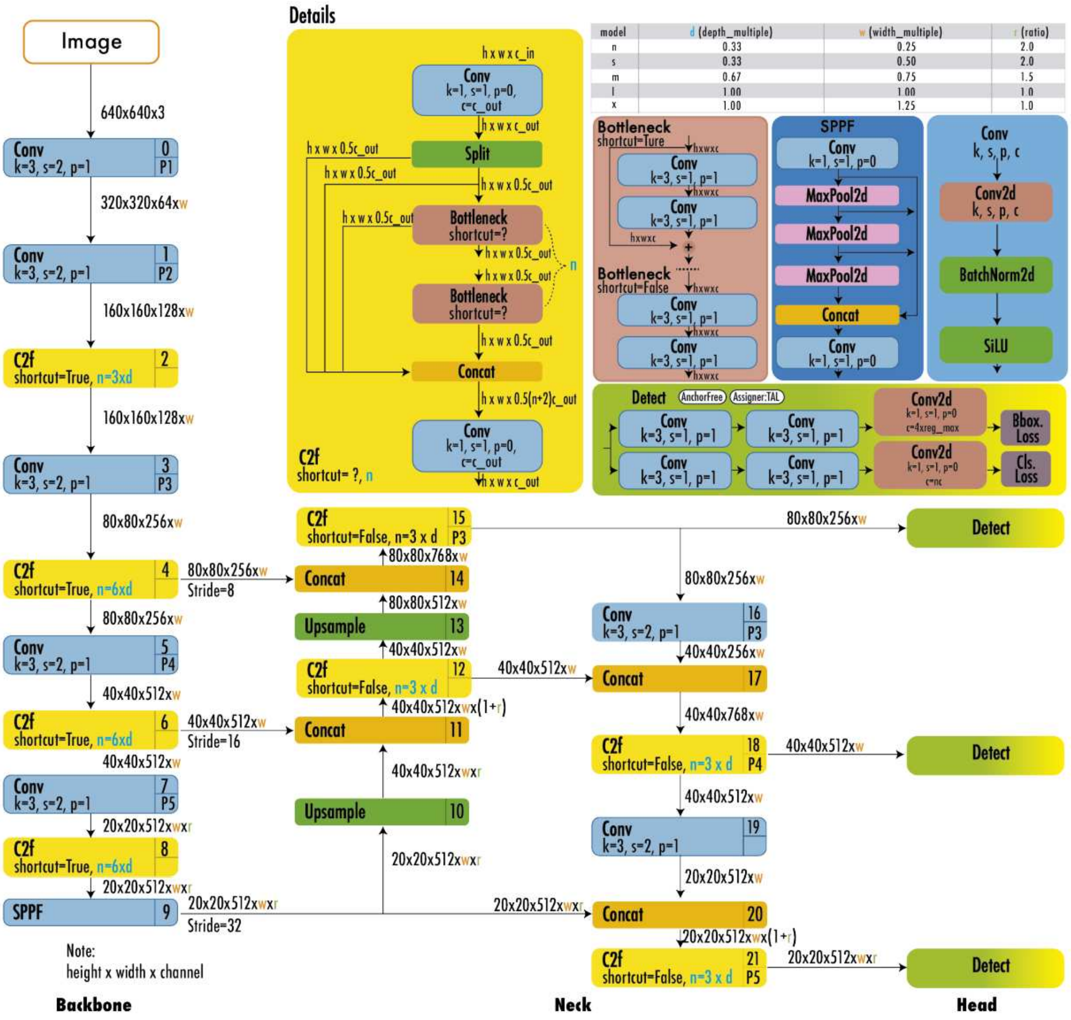


Figure 2.3: YOLOv1 Architecture from [Red+16].

Another popular group of image object detectors is single-stage detectors. The first and one of the most popular detectors of this group is the YOLO (You Only Look Once) family.

YOLO [Red+16] uses only one CNN to predict the class and regress bounding boxes in one pass, without needing region proposals. To do this, YOLO first divides the image into a $S \times S$ grid. In each cell, YOLO applies N bounding boxes. For each box, YOLO computes the class probability and bounding box offset values. Boxes that are above a certain threshold are used to find the object. Since all of this happens in a single pass, YOLO is considerably faster than two-stage detectors. Running at 45 FPS, even the first iteration of YOLO (YOLOv1) is well within the threshold of running in real-time.



model	d (depth_multiple)	w (width_multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

Figure 2.4: YOLOv8 Architecture from [TC23].

Over the years, multiple iterations of YOLO have been released. The latest iteration of YOLO, called YOLOv8 [Ult23], was released on January 10th, 2023 by Ultralytics, the same company that released YOLOv5. At the time of writing, there is no official paper for YOLOv8, yet. There are 5 variants of YOLOv8: nano, small, medium, large, and extra large. Based on a survey paper by Terven, et.al. [TC23] and the code at Github [Ult23] One big change in YOLOv8 is changing the CSPLayer into a C2f module (cross-stage partial bottleneck with two convolutions). YOLOv8 doesn't use anchors and it has separate heads for objectness (does a box contain an object or not), classification, and regression tasks. A sigmoid activation function is used for objectness score, while classification uses softmax. YOLOv8 uses CIoU [Zhe+20] and DFL [Li+20] for bounding box loss and binary cross-entropy for classification loss. Based on the GitHub page, when evaluated on the COCO val2017 dataset, YOLOv8 achieved up to 53.9 mAP at 283,28 FPS using TensorRT on a NVIDIA A100 (YOLOv8x).

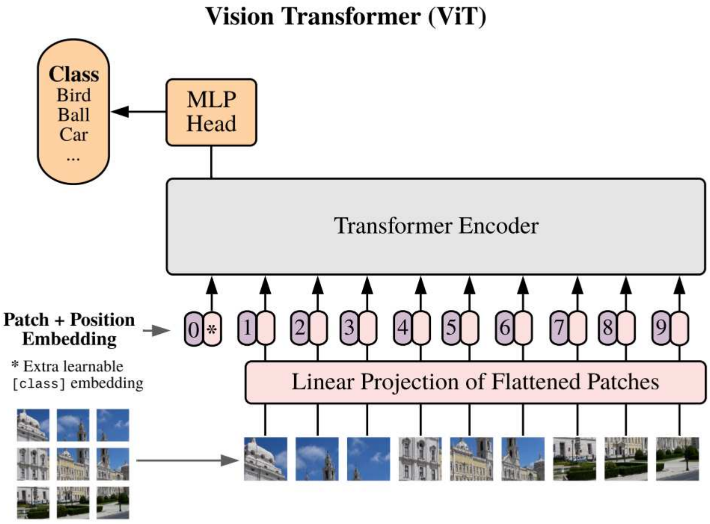
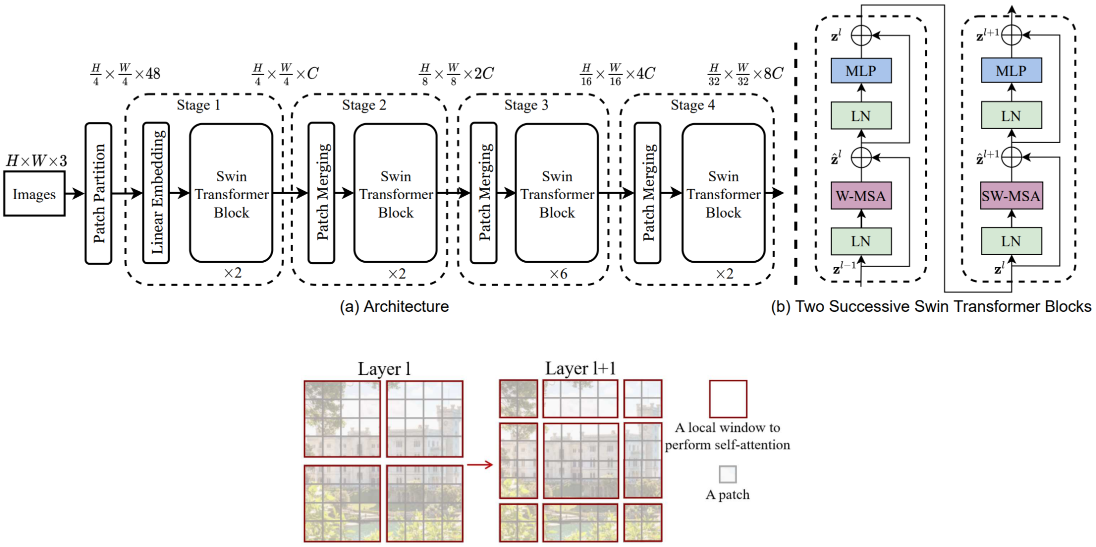


Figure 2.5: ViT Architecture from [Dos+20].

The last group of popular image detectors covered in this work are transformers-based detectors. Some well-known examples of this family are ViT and SWin. Vision Transformer (ViT) [Dos+20] and its successors are based on transformer networks commonly used in Natural Language Processing (NLP) tasks. ViT works by splitting the input image into 16 × 16 patches before flattening and then encoding them using a linear layer. This linear layer applies the same weights to every patch. Position embeddings are then added to them. The final embeddings are then used as tokens in a manner similar to how embedded words are used as tokens in NLP transformers. For classification, a learnable token is prepended to the sequence of tokens. The architecture of the transformer encoder itself is practically the same as the original NLP transformer architecture [Vas+17]. As its classification head, ViT uses a simple MLP. A key characteristic of ViT compared to CNN-based methods is the trade-off between dataset size and performance. Generally speaking, on smaller datasets CNN-based methods beat ViT, while on larger datasets ViT beats CNN-based methods.



(a) Architecture

(b) Two Successive Swin Transformer Blocks

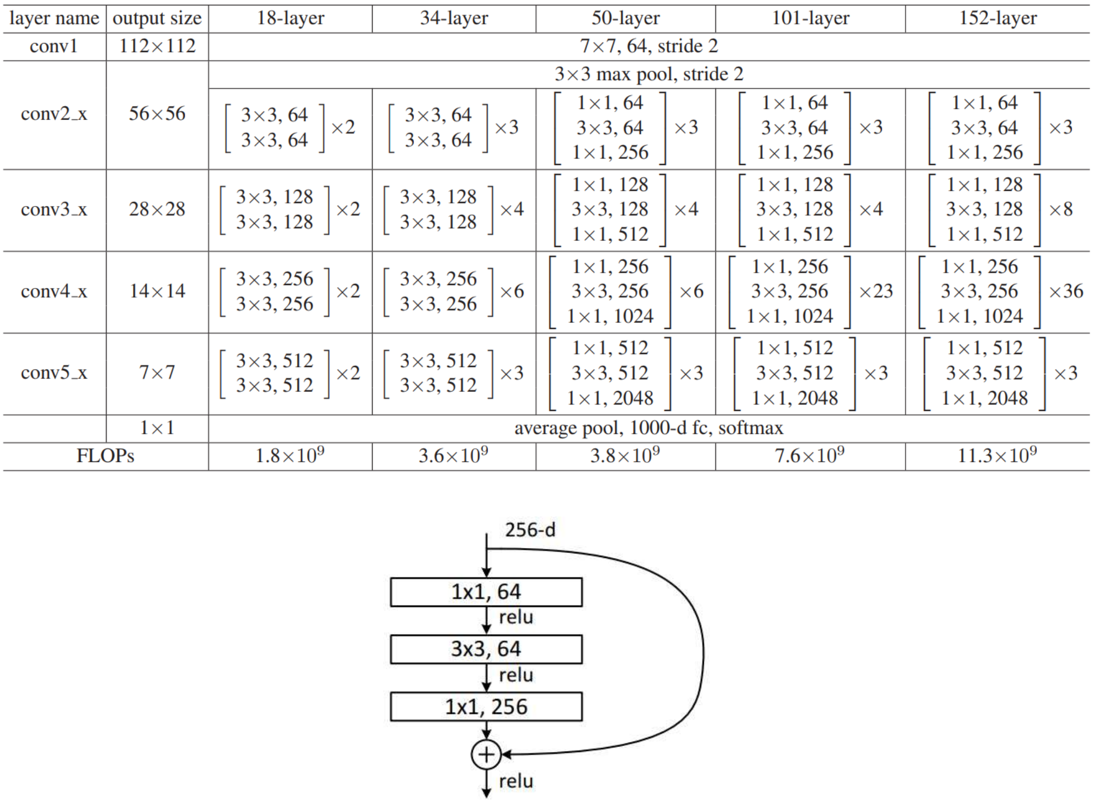
Figure 2.6: SwinT Architecture and visualization of *shifting windows* from [Liu+21].

SWin [Liu+21] works using sliding windows. On top of the patches, SWin adds non-overlapping windows on top. Each window contains multiple patches. The attention is applied to all patches within a window, but not outside it. But, this causes it to lack cross-window connections, limiting its power. To introduce these connections, they shift the windows between consecutive SWin blocks. This creates hierarchical feature maps, creating a global representation of the image. To create this global feature, they used adaptive average pooling and normalization. Unlike ViT, it doesn't use positional embedding. Instead, learned positional bias is introduced in the attention. The resulting global feature is used for object detection.

2.1.2 Image Backbones

All of the deep learning-based methods mentioned above utilize a backbone network to extract features from the input image. These feature extractors are the most relevant part of these methods to this work's use case, since they are what is needed to extract the features from input images, that will then be fused with LiDAR features.

R-CNN, Fast R-CNN, and Faster R-CNN use CNNs as their backbone. Often, ResNet [He+16] and its variations are used. Residual Network (ResNet) was first introduced in 2016. In a case where there are two consecutive convolution layers, the core idea of ResNet is introducing a skip connection to allow the input of the first layer to be added to the output of the second layer. This helps solve the vanishing gradient issue that standard deep networks generally have.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64 3×3, 64] ×2	[3×3, 64 3×3, 64] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[3×3, 128 3×3, 128] ×2	[3×3, 128 3×3, 128] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×8
conv4_x	14×14	[3×3, 256 3×3, 256] ×2	[3×3, 256 3×3, 256] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×23	[1×1, 256 3×3, 256 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512 3×3, 512] ×2	[3×3, 512 3×3, 512] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.7: ResNet Configurations and a Residual Block of ResNet50 [He+16].

One of the most popular variants of ResNet is ResNet50. Introduced in the same paper as Resnet, instead of skipping two layers, the building blocks of ResNet50 contain three layers and the skip connection bypasses all three layers. The three layers are a 1 × 1 layer, a 3 × 3 layer, and finally another 1 × 1 layer. Each layer has different depths depending on its particular configuration and uses ReLU as its activation function. ResNet50 has 16 of these 3 layer blocks with different configurations, a 7 × 7 convolution layer as pre-processing, and a fully-connected layer as a classifier at the end. In total, it has 50 layers, hence the name. The feature that a ResNet50 feature extractor generates will be a 7 × 7 × 2048 feature.

The YOLO variants use a variety of different backbone networks. The original YOLOv1 uses VGG-16 as its backbone. However, since the backbone network is just used to extract features and is interchangeable with other backbone networks, some implementations of YOLOv1 exist with ResNet50 as the backbone. Starting from YOLOv2 up to YOLOv5, DarkNet and its variants were used. YOLOv6 uses a network called EfficientRep as its backbone, with RepBlocks as its main component. RepBlocks are made of a stack of RepVGG blocks during training and RepConv blocks during inference, with ReLU activation layers. For medium and large networks, these blocks are revised into CSPStackRep blocks. YOLOv8 uses CSPDarknet53 with C2f modules (cross-stage partial bottleneck with two convolutions) replacing the CSPLayers. Only VGG-16 and CSPDarknet53 will be covered since they are used in the original YOLOv1 and the YOLOv8, which is the latest iteration of YOLO.

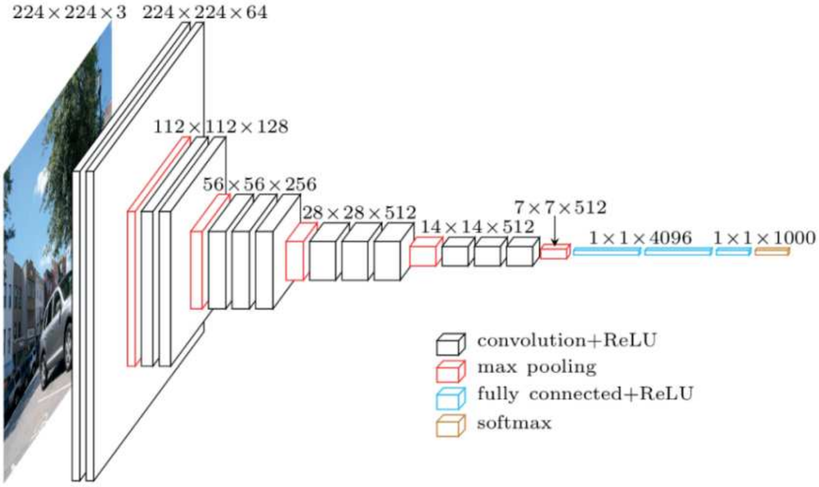


Figure 2.8: VGG-16 Architecture from [BD19].

VGG-16 [SZ14] (2014) was one of the first feature extractors ever. The structure is very simple. Two blocks of two 3 × 3 convolution layers and three blocks of two 3 × 3 convolution layers followed by a 1 × 1 convolution layer. The depths of the 5 blocks are 64, 128, 256, 512, and 512. After each block is a max-pool layer. In total, there are 13 convolution layers. These 13 convolution layers are the feature extractors. It is called VGG-16 since three fully connected layers are used as the classifier, giving a grand total of 16 layers in the architecture. Softmax is used after the three fully connected layers to generate the class predictions.

CSPDarknet53 (see Figure 2.4) consists of a stem layer (convolution layer, k=3, s=2 and p=1), then 4 stages of a convolution layer (k=3, s=2 and p=1) followed by a C2f module, and finally a stage of a convolution layer (k=3, s=2 and p=1) followed by a C2f module and a SPPF (spatial Pyramid Pooling Fast) bottleneck. The stem reduces memory and computation costs, the convolution layer and C2f modules extract relevant features from the image, and the SPPF bottleneck processes these features, while also speeding up computation by pooling features of different scales into a fixed-size feature map. Each convolution layer is followed by BatchNorm and a SiLU activation. C2f modules consist of a cross-stage partial bottleneck with two convolutions. They are modified CSPLayer modules from the original CSPDarknet53 of YOLOv5. They combine high-level features with contextual information to improve detection accuracy.

The transformer-based methods ViT and Swin use transformer architectures to extract

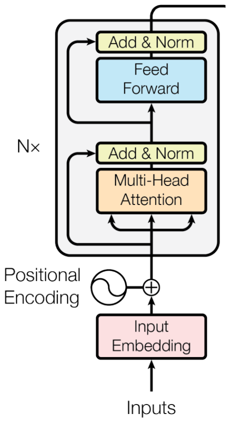


Figure 2.9: Transformer Encoder Architecture from [Vas+17].

features. Their classifiers are simple MLPs. ViT's backbone is the original transformer architecture from the original NLP transformer paper [Vas+17]. This transformer encoder consists of a normalization layer, a Multi-Head Attention layer, another normalization layer, and finally an MLP. Naturally, the most important part here is the Multi-Head Attention layer.

A Multi-Head Attention layer takes three inputs: query, key, and value. Queries and keys have a dimension of d_k, while values have a dimension of d_v. First, a learned linear layer is applied for each of the inputs, in order to linearly project h times. This is followed by a scaled dot product following the attention formula:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}}) \cdot V$$

This happens separately for each of the h heads. The h results are then concatenated and passed through a linear layer, creating the final feature. To understand this formula, one can think of the result of the softmax as a distribution curve. This distribution curve is then applied to the values V by the dot product. The elements of V with higher values get more attention and lower ones get less. In ViT, the queries, keys, and values are calculated as the dot product of the patch embeddings with the weights W_Q, W_K, and W_V.

The transformer encoder of Swin is generally the same as ViT's. The main difference, as mentioned in the previous subsection is that the attention is applied only for patches within each window and the usage of sliding windows to introduce cross-window connections. The Swin authors modified the Multi-Head Self Attention block into Window Multi-Head Self Attention (W-MSA) and Shifted Window Multi-Head Self Attention (SW-MSA) blocks for two consecutive Swin transformer blocks, where the window is displaced by $(\lfloor \frac{M}{2} \rfloor, \lfloor \frac{M}{2} \rfloor)$ pixels, for a window containing M x M patches. The formula of this modified Self Attention is:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d}} + B) \cdot V$$

This matrix B with the dimension M^2 x M^2 is the relative position bias. In this formula, Q, K, and V have the dimension M^2 x d, where d is the dimension of query and key, while M^2 is the number of patches in a window. The relative position on both axes lies between $M+1$ and $M-1$, so the Swin authors parameterized the smaller bias matrix $\hat{B}$ with the dimension $2M-1$ x $2M-1$. The values of B are taken from $\hat{B}$

2.2 Point Cloud 3D Object Detection

A large variety of methods exist for object detection in 3D point clouds. Currently, most of these methods are made for and trained on onboard (vehicle) data, which are far more widely available. As a result, they might need to be readjusted first in order to run with good performance on infrastructure data, which has a very different perspective to onboard data. Furthermore, these methods can be divided based on how they treat the raw 3D point cloud data before feature extraction. Generally speaking, there are point-based methods that take the raw 3D point cloud as is and voxel-based methods that first transform the raw 3D point cloud into segments called voxels (volume pixels).

2.2.1 Point-based Methods

As mentioned in the introduction, point-based methods use the raw 3D point cloud as their input. This is good since these methods are able to utilize the complete and more granular information provided by raw 3D point clouds. Unfortunately, this also means that they will have to work on a larger amount of data. This is another trade-off that needs to be considered when choosing a method to use as the 3D point cloud feature extractor in our system.

On top of this, point-based methods must consider the properties of raw 3D point cloud data. First, it is unordered. This means that, unlike image pixels that generally always come in the same order inside an array, the same point might be stored at a different index in two consecutive point clouds. As a result, point-based methods need to be permutation invariant. Second, points are not isolated. Neighboring points often form a certain object in the environment. Point-based methods need to capture this meaningful information. Finally, even when transformations are applied to a point cloud, changing the coordinates of a point, the point represented by the new coordinate is still the same point as pre-transformation. Point-based methods need to be transformation invariant.

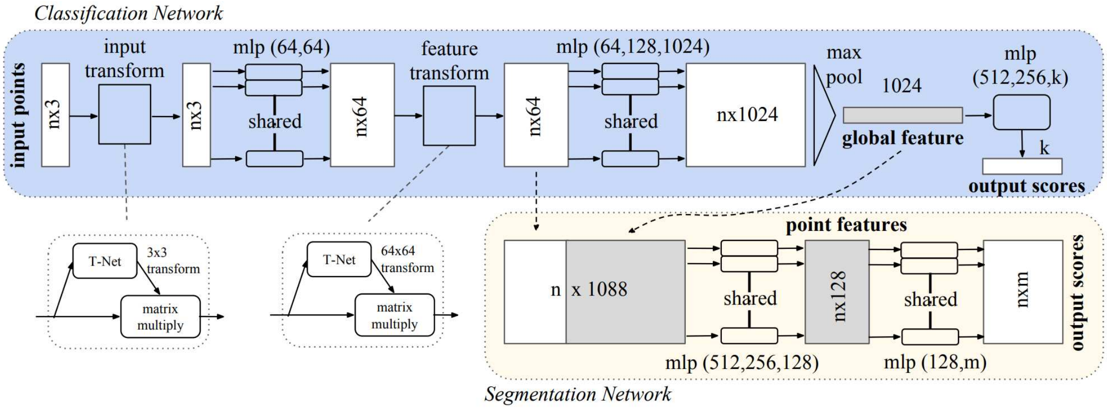


Figure 2.10: PointNet Architecture from [Qi+17a].

One of the most popular point-based methods is the PointNet family. The original PointNet [Qi+17a] was released in 2017. PointNet can be divided into classification and segmentation networks. In the classification network, per-point features are extracted by running the points through MLPs, which use symmetric operations. This symmetry, combined with the symmetric max-pool layer that is used at the end to combine the point features into a global feature, allows PointNet to be permutation invariant. This initial global feature is then sent to an MLP for classification.

In the segmentation network, this global feature is then concatenated with the original per-point feature and then used to generate a new per-point feature using MLPs. This new per-point feature is now aware of both local and global information, allowing PointNet to also be able to capture information about a point and its neighbors. This new per-point feature is then used for segmentation.

To ensure transformation invariance, in the initial steps of the classification network, the original points and the first features generated from them are passed through a "transformation network" or T-Net as they call it, which predicts an affine transformation matrix and applies this to the point and per point feature. The goal of this is to align points and features from different input point clouds.

However, PointNet has a weakness. Since data is extracted from individual points, the relation between points is ignored. PointNet++ [Qi+17b] attempts to solve this weakness. It introduces a multi-scale feature learning architecture called the Prediction Pyramid to enable it to capture these local structures/relations between the points. The core of PointNet++ is the Set Abstraction Layer. It consists of 2 layers: the Sampling Layer and the Grouping Layer. These layers pre-process the data by downsampling the points at the start of each layer and then grouping them into clusters Farthest Point Sampling (FPS) or Ball Query. They are then fed into a PointNet layer for feature extraction.

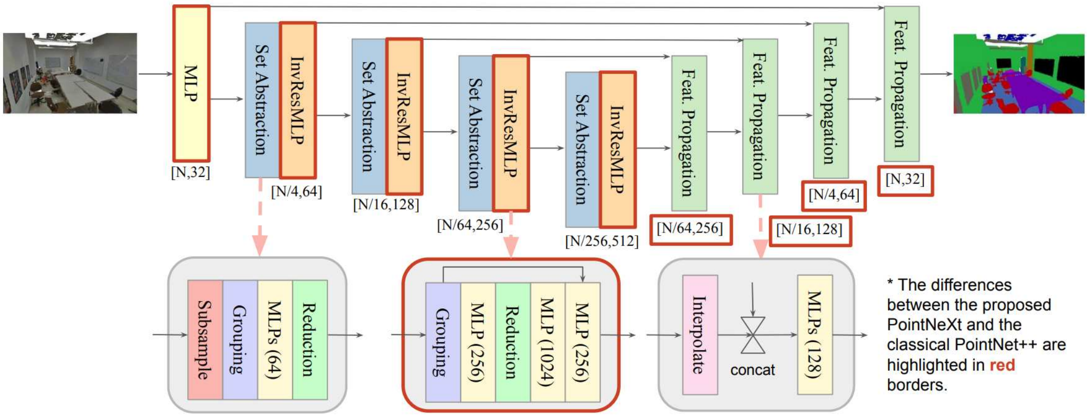


Figure 2.11: PointNext Architecture from [Qia+22].

The latest development of this family of methods is PointNeXt [Qia+22]. This method improves on PointNet++ in two ways: training and architecture. On the training side, it made changes to the data augmentation and optimization techniques. It uses a variety of data augmentation techniques depending on the dataset. They include point resampling, height appending, color dropping, and rotation. For optimization, it uses AdamW [5] instead of Adam optimizer, cosine instead of step decay, and Cross Entropy with label smoothing. For some datasets, the learning rate is increased to 0.01 from 0.001.

On the architecture side, it applies two modifications: receptive field scaling and model scaling. The receptive field is scaled by using a larger radius to query the neighborhood (the exact value is dataset-specific). To scale the model, the authors introduced the Inverted Residual MLP (InvResMLP) block. This block is built upon the SA layer and is appended after the first Set Abstraction layer in each stage. On top of this, they also used 4 SA blocks for both classification and segmentation, used a symmetric decoder with a matching channel size with the encoder, and they added an MLP at the very beginning to map the input to a higher dimension.

2.2.2 Voxel-based Methods

Another way to deal with 3D point cloud data is to first transform it into segments (voxels, pillars, frustums, etc.). Doing this introduces a structure into the previously unstructured 3D point cloud data. Voxelization also reduces the volume of data the methods have to work with since points are now grouped into these voxels and then treated as one input object. As a result, efficiency and inference times are improved. Two popular methods that use these principles are VoxelNet [ZT18] and PointPillars [Lan+19].

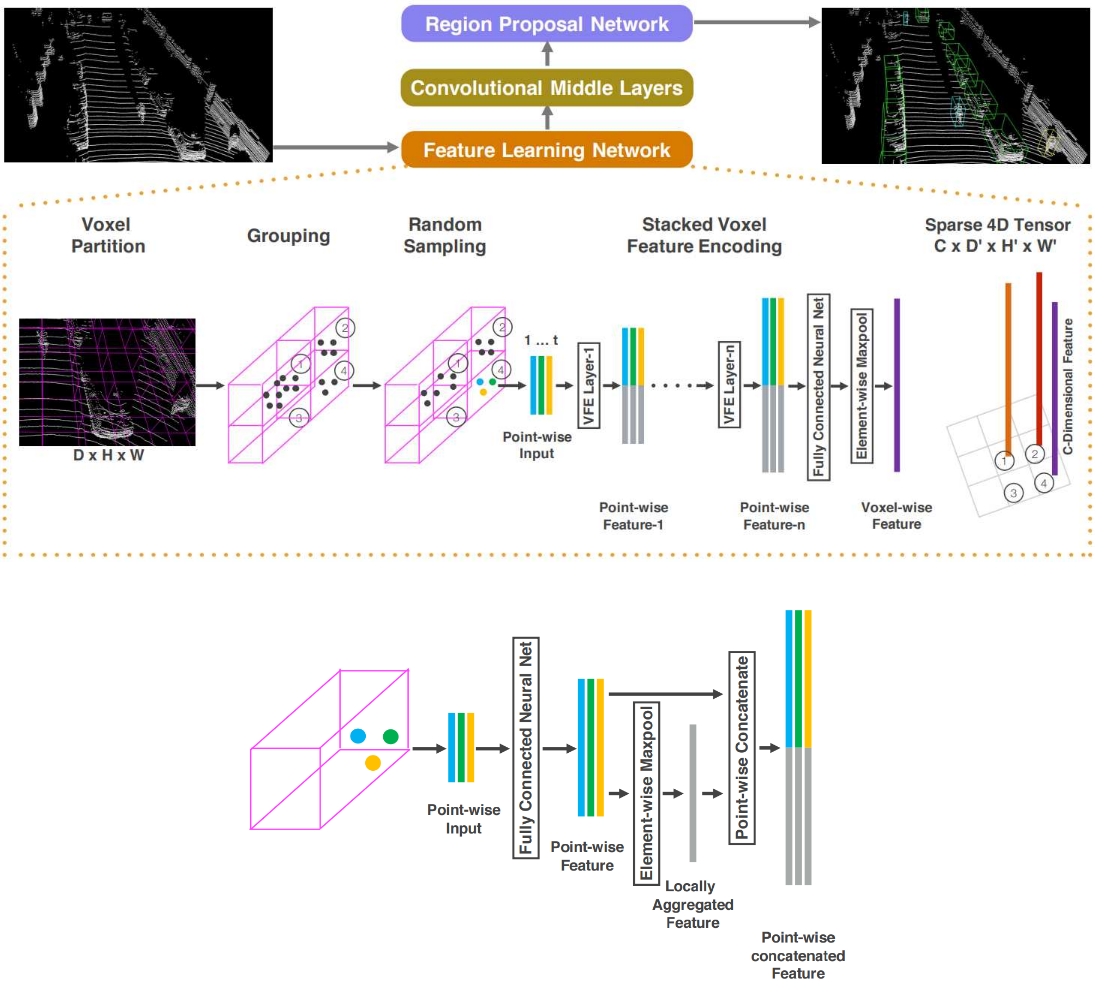


Figure 2.12: VoxelNet Architecture and Voxel Feature Encoder [ZT18].

In VoxelNet, points are first grouped into voxels and then handled per voxel. Due to the sparse nature of 3D point clouds, each voxel could have a significantly different amount of points in them. To save on computation effort and ensure that the amount of points processed from each vector is as close as possible, only non-empty voxels are processed, and random sampling of N points is applied to voxels containing more than N points.

These randomly sampled points contain the x, y, z, and reflectance information. They are then enriched with the x, y, and z values relative to the centroid of their voxel. They are then passed through a Feed Forward Network, containing a linear layer, batch normalization, and ReLU. These point-wise features of the voxel are passed through element-wise max-pooling to get locally aggregated features. The point-wise features are concatenated with the locally aggregated features for the voxel. All of this combined creates a Voxel Feature Encoding (VFE) layer. These layers are stacked several times and after the last one, a Fully Connected

Layer and element-wise max-pooling are applied to create the final voxel-wise features. This process is also applied to empty voxels.

On top of this, VoxelNet also applies a convolution middle layer. Each middle layer applies a 3D convolution layer, batch normalization, and ReLU in this exact order. The goal of this middle layer is to aggregate the voxel features in an increasingly larger receptive field and to add more information to the existing voxel-wise features.

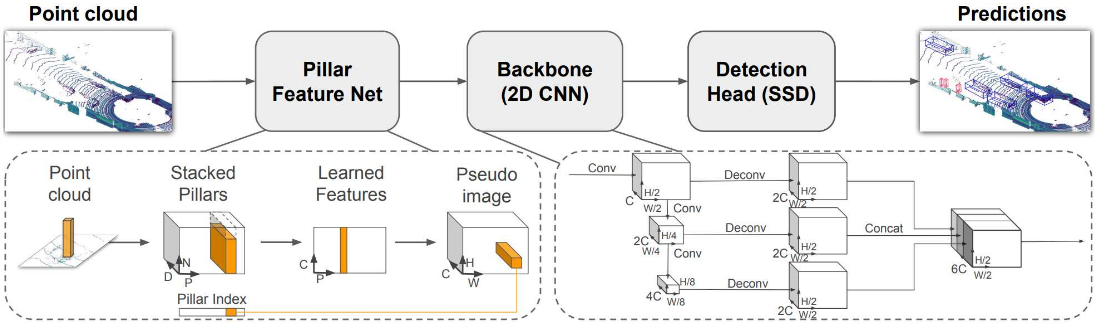


Figure 2.13: PointPillars Architecture from [Lan+19].

Different from VoxelNet, PointPillars segments the points in the point cloud into pillars instead of 3D cube voxels. The first step of PointPillars is to convert the 3D point cloud into a 2D pseudo-image of the point cloud to allow the application of 2D convolution. First, the point cloud is divided in the x and y axis into an evenly sized grid, creating a set of pillars. A pillar is basically a voxel of VoxelNet, just with an unlimited range in the z-axis. The points are also enriched with reflectance, the distance to the arithmetic mean of all points in the pillar, and the distance to the center of the pillar itself.

Same as the voxels of VoxelNet, the pillars are also mostly empty due to the sparsity of point cloud data. When sampling, PointPillar limits the number of non-empty pillars per sample and the number of points per pillar. If a pillar or sample has too much data, random sampling is used. If it doesn't have not enough data, zero padding is used. PointNet (linear, batch normalization, and ReLU) is then applied to each point, followed by maxpooling over the channel dimension. The resulting features are then spread back to the original locations, creating a 2D pseudo image.

The backbone of PillarNet is similar to VoxelNet's. The main difference is, that instead of applying 3D convolution layers in the middle layer, PillarNet applies 2D convolution layers instead. The backbone consists of 2 parts: top-down and bottom-up blocks. The top-down block applies the aforementioned 2D convolution layers in an increasingly small receptive field, followed by batch normalization, and ReLU. Meanwhile, the bottom-up blocks apply upsampling using 2D deconvolution layers, followed by batch normalization, and ReLU. The final feature is created by concatenating all features from different strides.

When used with infrastructure-based data, especially important if one were to attempt transfer learning from models pre-trained on vehicle data, models such as VoxelNet and PointPillars need to be adapted to the different perspectives the data is recorded from. A key difference is the shift in the z-axis (infrastructure data are recorded from a higher-up perspective compared to vehicle data). Cyber Mobility Mirror [Bai+23] was created to try to work around this issue.

In Cyber Mobility Mirror, the authors attempted to use a model pre-trained on vehicle data (NuScenes dataset [Cae+20]) on data generated from a LiDAR mounted on a traffic light. First, they geo-fenced the data generated by the LiDAR sensor to an area of $[-51.2m, 51.2m]$ in the x and y-axis, centered around the LiDAR sensor. The sensor is located at a height of 4,74m above ground, so the y-axis range is set to $[0m, 5m]$.

Then, they sent the point cloud data to the Roadside Point-cloud Encoder and Decoder (RPEaD), which transforms the point cloud to a perspective that is more similar to the vehicle perspective that the model was pre-trained on. To generate the transformation matrix, they used Least Squares Regression to auto-calibrate the LiDAR sensor and generate a rotation matrix. The translation part of the transformation is simply $[0, 0, \Delta z]$, where $\Delta z = z_{\text{roadside}} - z_{\text{vehicle}}$. To make the model less sensitive to this shift in the z-axis, they applied the voxelization strategy and feature extraction used by PointPillars [Lan+19] and then they followed by a Feature Pyramid Network (FPN) and anchor-based detection head of SSD [Liu+16]. The loss function used combines Smooth L1 loss for localization loss, softmax classification loss for direction loss (detecting flipped boxes), and focal loss [Lin+17] for classification loss. The authors also introduced improvements, such as 3D Multi-Object Tracking using 3DSORT (3D object matching on DeepSort), Geo-localization, Cloud Communication, and Multi-Object Reconstruction. But, they are not relevant to this work so they will not be covered here.

2.3 Camera-LiDAR Deep Fusion (Single Node)

Camera-only and LiDAR-only methods all have their own weaknesses due to the characteristics of the sensor that they are using. For example, cameras might produce poor-quality images when the lighting conditions are not ideal (too bright/dark, lens glare, etc.). Meanwhile, LiDARs might produce noisy point clouds when it is raining heavily or foggy. As a result, it makes sense to fuse both sensor modalities to introduce redundancy and allow the sensors to try to cover for each other's weaknesses. There are multiple approaches when doing this. For example, there are methods that seek to apply late fusion to fuse camera and LiDAR data, e.g. CLOCs [PMR20] and InfraDet3D [Zim+23a]. However, this work will only focus on methods that utilize the same approach as itself, deep fusion. In this section, we will cover three such methods: TransFusion [Bai+22a], DeepFusion [Li+22], and BEVFusion [Liu+23b].

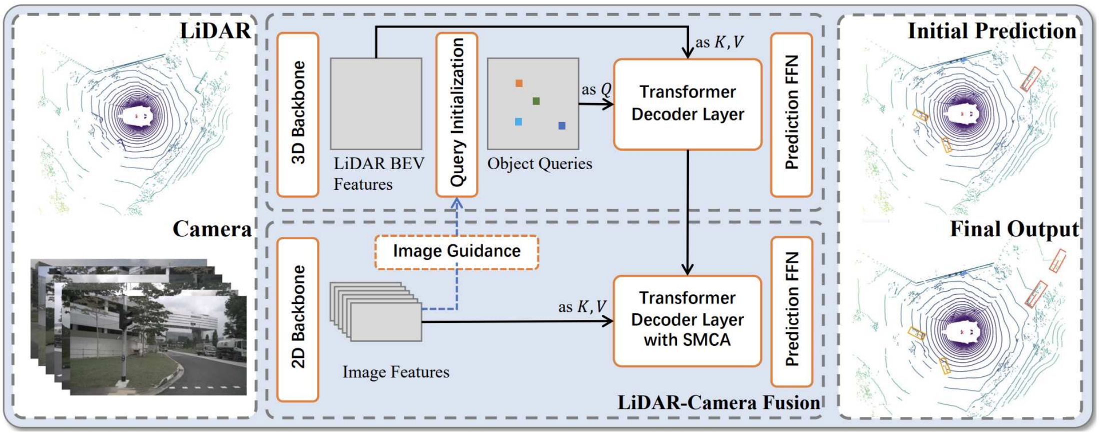


Figure 2.14: TransFusion Architecture from [Bai+22a].

TransFusion [Bai+22a] is a camera-LiDAR deep fusion method made for vehicle use cases. It is trained and tested on the nuScenes [Cae+20] and Waymo [Sun+20] datasets. TransFusion works by first extracting a BEV feature map from the LiDAR point cloud and feature maps from the images using standard 3D and 2D convolution backbones. Then these features are processed in two separate layers.

In the first layer, the LiDAR BEV feature map serves as the key and value for the transformer decoder of that layer. Meanwhile, the image feature maps serve as a guide to initialize the queries in an input-dependent and category-aware fashion. Input dependence is ensured by generating K heatmap for K object categories from the BEV LiDAR feature map. The image feature maps are then projected into BEV perspective to create BEV image feature maps. The LiDAR and images BEV feature maps are then averaged to create a final heatmap, which is then used to generate object candidates and the top-N candidates are used as initial queries. This allows image information to be used to guide the query initialization. Category awareness is achieved by adding a category embedding to each query. By doing this, the initial queries are already at or close to potential object centers, side information when modeling the object-object and object-context relations is provided, and it delivers prior knowledge of the object, improving property prediction. This means TransFusion now only needs one layer of transformer decoder.

In the second layer, the image feature maps serve as the key and value for the transformer decoder of that layer, while the results of the transformer decoder layer in the first layer serve as the queries. In this layer, the transformer decoder is also equipped with a Spatially Modulated Cross Attention (SMCA) module. This module weighs the cross attention with a 2D circular Gaussian mask around the projected 2D center of each query. This mask is generated similarly to CenterNet [ZWK19]. This means that each query only attends to the region around the projected 2D box, helping the network to learn where to select image features based on LiDAR features better and faster. The transformer decoders used in both layers follow DETR [MGJ21]. The detection head is similar to CenterPoint [YZK21].

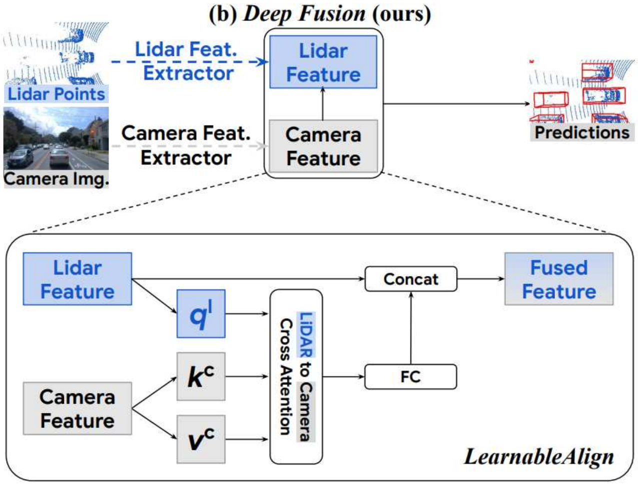


Figure 2.15: DeepFusion Architecture from [Li+22].

DeepFusion [Li+22] is also created for vehicle use cases and it is trained and tested on the Waymo Open Dataset [Sun+20]. Similar to TransFusion [Bai+22a], it is also based on the transformer architecture (cross attention) to capture the relations between the camera and LiDAR features. This part of the architecture is named *LearnableAlign* by the authors.

The image features are extracted using ResNet [He+16] and used as the keys and values, while the LiDAR features are extracted using PillarNet [Lan+19] and used as the queries. The result of the cross-attention module is normalized using softmax and then fed through a fully connected network (FCN). The resulting fused features are then concatenated with the original LiDAR features. These are the final fused features that are then detected by the 3D object detection head, e.g. PointPillars [Lan+19].

Another improvement DeepFusion has is *InverseAug*. Point clouds are usually first augmented before being fed into the model during training. The problem is, that this means that the correlation between points and pixels can no no longer be found using the original camera and LiDAR parameters. To solve this, *InverseAug* saves all of the augmentation transformation matrices applied to the point cloud and then uses this information to inverse the augmentations in order to find the original coordinates of the points. This improves the alignment between the points and pixels.

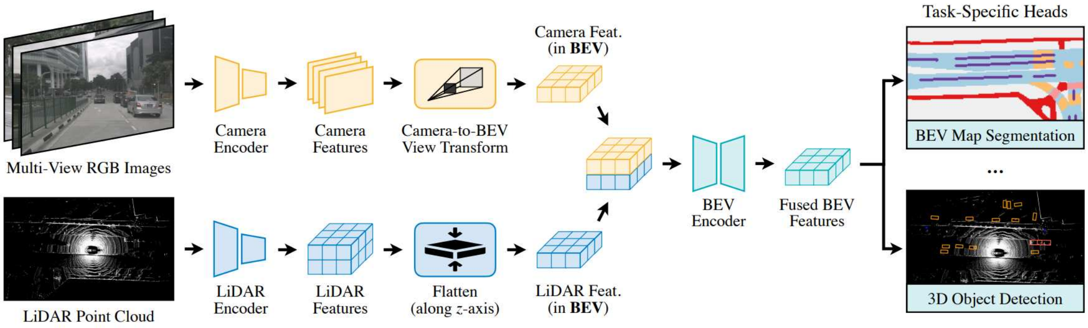


Figure 2.16: BEVFusion Architecture from [Liu+23b].

BEVFusion [Liu+23b] is another method created for vehicle use cases. It is trained and tested only in the nuScenes dataset [Cae+20]. However, it takes a different approach to the two methods above. Instead of utilizing cross-attention to fuse the camera and LiDAR features, BEVFusion uses a convolution-based fuser to do this. To enable this, BEVFusion first brings both of the features into one representation: bird's eye view (BEV). They chose this representation because it preserves both geometric (from LiDAR) and semantic information (from camera). For the LiDAR features, this is relatively simple. BEVFusion uses VoxelNet [ZT18] to extract per voxel LiDAR features. Then, it flattens these features along the z-axis to get the BEV representation.

For the camera features, BEVFusion first uses Swin ([Liu+21] to extract per-pixel features. To transform these features to BEV, it uses a modified version of the method introduced in the Lift, Splat, Shoot (LSS) paper [PF20]. First, one could re-project each pixel into a line in the 3D space. On this line, D discrete points exist and the probability that the pixel exists on each point (depth) is calculated. Then, the pixel features are rescaled by each related point's depth probabilities, creating a feature point cloud in the shape of a 3D frustum. This feature point cloud is then quantized along the x and y axis into $r \times r$ grids and BEVpooling is used to aggregate features in each $r \times r$ grid. Then the result is flattened in the z-axis creating the camera BEV representation.

BEVFusion improves on this camera-to-BEV process by introducing three improvements. First, to speed up BEVpooling, each point in the camera feature point cloud is associated with a BEV grid. This is possible as long as the intrinsic and extrinsic parameters of the camera and LiDAR do not change. The points are then sorted based on grid indices and the rank of each point is recorded. During inference, all feature points are reordered based on the precomputed ranks. This reduced the latency of grid association from 17 ms to 4 ms according to the authors.

The next improvement is done in the feature aggregation step of BEVpooling. The authors implemented a specialized GPU kernel. They assign a GPU thread to each grid. It calculates its interval sum and writes the result back. This removes the dependency between outputs and avoids writing partial sums to the DRAM. This reduced the latency of feature aggregation from 500ms to 2ms. Combined, both improvements speed up camera-to-BEV transformation by 40x.

The final modification is introducing a 2.5D depth stream into the LSS process. Utilizing the 3D point cloud, they first inverse all of the augmentations applied to it before transforming it into pixel coordinates and applying the image augmentations. Using this information, they created a new depth information, which is then used during the image feature extraction process.

After getting both camera and LiDAR BEV features, BEVFusion stacks them on top of each other and then passes them through a convolution-based BEV encoder/fuser, which creates the final fused feature map. This feature map is then passed to the head for the task at hand. For 3D object detection, BEVFusion uses the head from TransFusion [Bai+22a]. BEVFusion is one of the state-of-the-art models for 3D object detection. The ensemble version of BEVFusion (BEVFusion-e) is 7th place in the 3D object detection leaderboard of nuScenes, while a modified version of BEVFusion, which utilizes Edge Aware LSS (EA-LSS) [Hu+23] is first place in the same leaderboard [Inc23].

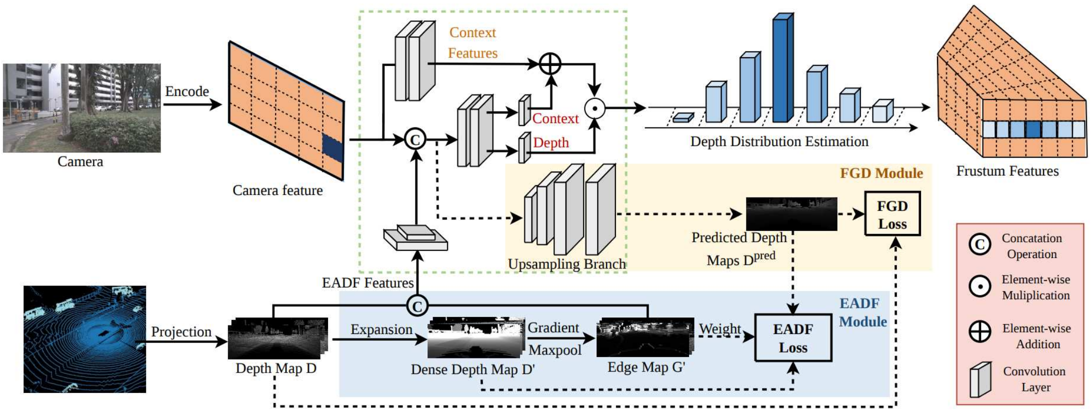


Figure 2.17: EA-LSS Architecture from [Hu+23].

The final single node camera-LiDAR deep fusion method covered is Edge Aware - Lift, Splat, Shoot (EA-LSS) [Hu+23]. As mentioned above, this method is currently the best method on the nuScenes 3D object detection leaderboard [Inc23]. The main contribution of this method is the introduction of an edge-aware version of the Lift, Splat, Shoot (LSS) [PF20] method to transform the camera features into a frustum of features, with improved depth estimation, especially in regions where the depth changes significantly (depth jump problem).

The core of this method lies in two modules: Fine-grained Depth (FGD) and Edge-aware Depth Fusion Module (EAGD) module. In the FGD module, an upsampling branch is used to predict depth maps from the camera features. Projected depth maps from the LiDAR point clouds are used as the ground truth. A fine-grained depth (FGD) loss is used to calculate the loss between the non-zero pixels in the projected depth maps and their equivalent in the predicted depth maps. This helps the method to retain accurate depth information.

The EAGD module is used to solve the depth jump problem. In this module, the projected depth maps from the LiDAR point cloud are first expanded into dense depth maps by expanding the maximum depth value of each block to fill the whole block. These dense depth maps are then used to create edge maps by calculating the gradients of dense depth maps and then using max pooling on the last dimension and normalization on the gradients. Edge-aware depth fusion (EADF) loss is used to focus on the edge of each object. The dense depth maps are used as the ground truth and the edge maps are used as weights.

The improved camera frustum features are then flattened and fused with the flattened BEV LiDAR features creating the final feature map. This feature map is then used by the detection head to perform the 3D object detection task. The model was trained and tested on the nuScenes Dataset [Cae+20]. It achieved the best BEV mAP result (the mAP nuScenes uses) in the leaderboards at 76.5 mAP.

2.4 Multi Node Deep Fusion

Currently, multi-node deep fusion is still under-researched compared to single-node (mostly vehicle) deep fusion methods. There are multiple ways to fuse data from multiple nodes as mentioned in the introduction, early, deep, and late fusion. However, based on the trade-offs, in this work deep fusion is used and as such, only deep fusion methods will be covered in this section.

2.4.1 LiDAR-only Vehicle-Infrastructure Deep Fusion

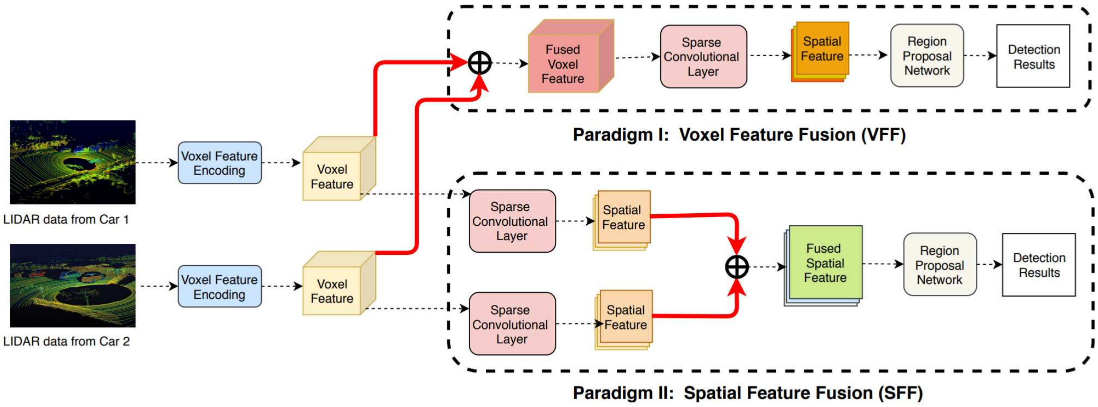


Figure 2.18: F-Cooper Architecture from [Che+19a].

The first method in this subsection is F-Cooper [Che+19a], which itself is a further development of an early fusion method by the same author Cooper [Che+19b]. F-Cooper aims to fuse the features from two vehicle nodes. In F-Cooper, there are two paradigms introduced: Voxel Feature Encoder (VFE) and Spatial Feature Encoder (SFE). The assumption is that the point clouds are already aligned in the same coordinate system before they are turned into voxels and the voxel features are extracted using VoxelNet's [ZT18] VFE layer. This way, when the voxel features from two vehicles are fused, the same voxel in both features represents the same 3D location. The authors then use element-wise Maxout operation to fuse both voxel features.

In the VFE paradigm, the voxel features are immediately fused before being passed to a sparse convolutional layer to create an already fused spatial feature. In the SFE paradigm, the separate voxel features are passed separately to a sparse convolutional layer to create separate spatial features. These spatial features are then the ones fused using the element-wise Maxout operation. In both paradigms, these fused spatial features are then passed to a Region Proposal Network (RPN) of VoxelNet to create the final detections.

The next method covered in this section is PillarGrid [Bai+22b]. Different from the previous two methods, PillarGrid works from the BEV perspective and it works by fusing vehicle and infrastructure nodes. PillarGrid works in the coordinate system of the infrastructure system, so before the features are extracted on the vehicle side, the vehicle LiDAR point cloud is first transformed into the infrastructure coordinate system and geo-fenced appropriately.

Then, the features are extracted from the point clouds separately in the vehicle and infrastructure side using PillarNet from PointPillars [Lan+19]. The resulting pillar-wise grid image feature from the vehicle side is then transmitted to the infrastructure. They are then stacked on top of each other and concatenated. 3D MaxPooling is then used to fuse both features. This works because the features and their grids are already aligned since the vehicle point

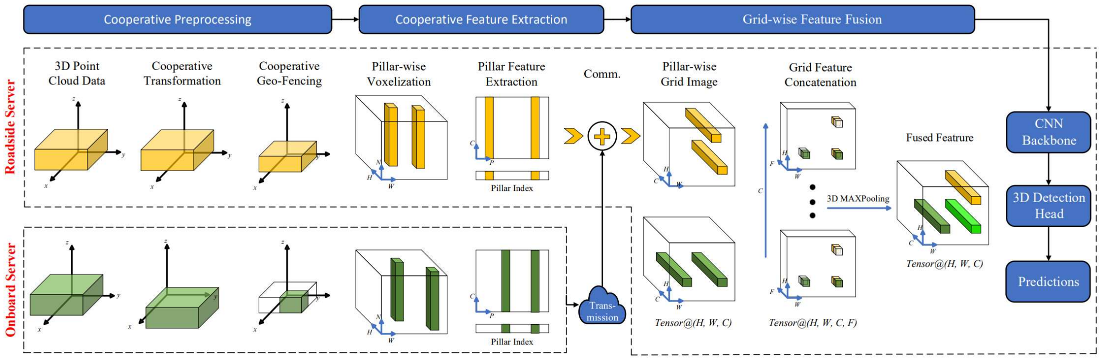


Figure 2.19: Architecture of PillarGrid module from [Bai+22b].

cloud is already transformed into the infrastructure coordinate system prior to feature extraction. The resulting fused features are sent to the Region Proposal Network (RPN) of VoxelNet [ZT18], which serves as the CNN backbone. Finally, the results are sent to the detection head of PointPillars [Lan+19] to generate the 3D detections.

One of the state-of-the-art methods in multi-node fusion is V2X-ViT [Xu+22]. V2X-ViT is designed to fuse features from multiple vehicle and infrastructure nodes. It is based on transformers in order to capture the heterogeneous nature of V2X systems with strong robustness against various noises. Similar to every other method covered in this section, there is an assumption that the point clouds are all aligned in one coordinate system. In the case of V2X-ViT, this coordinate system is the ego-vehicle's.

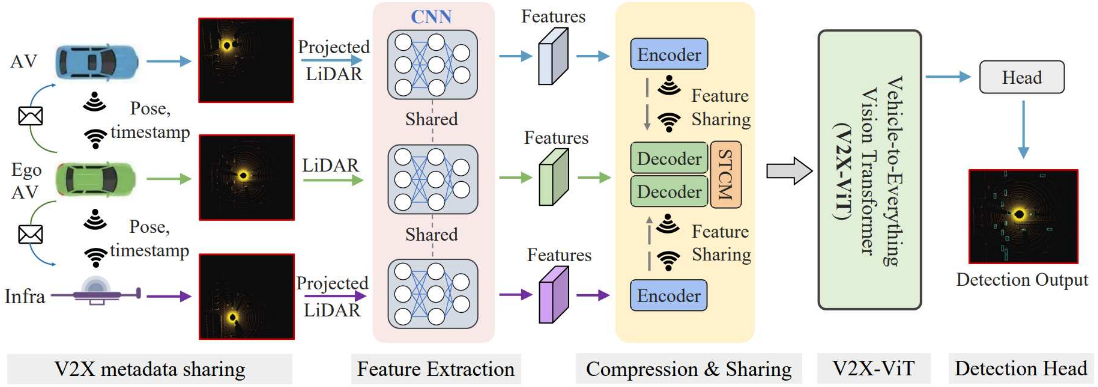


Figure 2.20: Overall Architecture of V2X-ViT from [Xu+22].

First, features are extracted from the point clouds of the nodes involved in the system after they are transformed into the ego-vehicle coordinate system using PointPillars' PillarNet [Lan+19]. These aligned features are then transmitted to the ego-vehicle. V2X-ViT utilizes an encoder and decoder to apply compression on these features during transmission to the ego-vehicle to reduce latency. Once they arrive at the ego vehicle, they are sent to a Spatial-Temporal Correction Module (STCM), which warps the feature maps to correct for the inherent misalignment that occurs because of the latency during transmission. Then, the features are sent to the V2X-ViT module, in which several V2X-ViT blocks are stacked in a series to learn inter-agent interaction and per-agent spatial attention.

The goal of the V2X-ViT block is to capture the heterogeneous graph representation between all nodes using the Heterogeneous Multi-Agent Self-Attention (HMSA) block and to capture long-range interactions at various scales using a Multi-Scale Window Attention (MSwin) block. For HMSA, one could visualize all involved participants as a directed graph

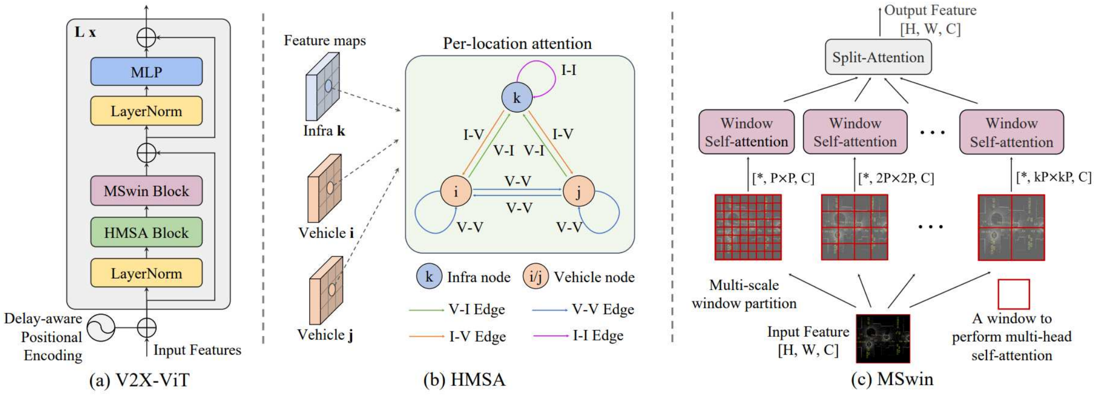


Figure 2.21: Architecture of V2X-ViT module from [Xu+22].

where each participant is a node and they are connected to each other by edges. HMSA encodes the relations in this graph. MSwin is based on the SWin [Liu+21] architecture covered in the image object detection section. It is a pyramid of windows, each of which captures a different attention range. Using different window sizes improves the detection robustness against localization errors. Larger windows capture long-range visual cues to compensate for large localization errors, while smaller windows perform attention at finer scales to preserve local context.

The output of the V2X-ViT module is fused features, which are then used as the input of the detection head. The detection head consists of 1 × 1 convolution layers for box regression and classification. Regression outputs the location, size, and yaw of the box, while classification outputs the confidence score of it being an object or background. Smooth L1 loss is used for regression, while focal loss [Lin+17] is used for classification.

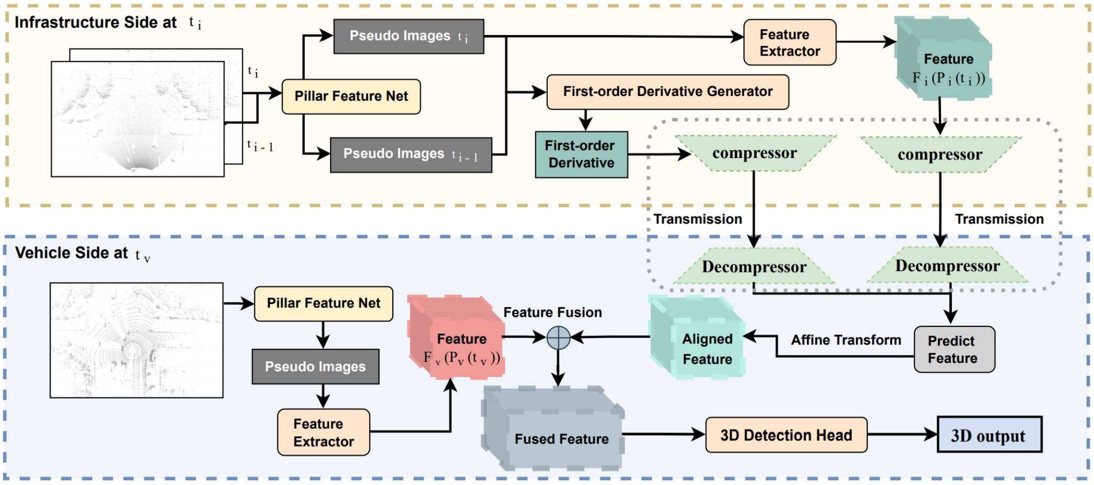


Figure 2.22: Architecture of FFNet from [Yu+23].

The next state-of-the-art method in the LiDAR-only, vehicle-infrastructure fusion category is Feature Flow Net (FFNet) [Yu+23]. The goal of this method is to address the temporal asynchrony between infrastructure and ego-vehicle sensor data, which could cause misalignment during the fusion process, negatively affecting performance. This method is centered around the ego-vehicle.

On the infrastructure side, bird's eye view (BEV) features are extracted from the LiDAR point cloud using PillarNet [Lan+19]. The extracted feature maps from the current and previous frames are then passed through the backbone and feature pyramid network (FPN) of SECOND [YML18] to estimate the first-order derivative. This and the current frame's BEV feature map are then each compressed using three ConvBn-ReLU blocks and transmitted to the ego-vehicle.

On the ego-vehicle side, the received first-order derivative and BEV feature map are then decompressed using three Deconv-Bn-ReLU blocks. The aligned infrastructure feature map at the current vehicle timestamp is then predicted using the formula:

$$\tilde{F}_i(t_v) \approx F_i(P_i(t_i)) + (t_v - t_i) * \tilde{F}_i'(t_i)$$

where $\tilde{F}_i(t_v)$ is the infrastructure feature at the vehicle timestamp, $F_i(P_i(t_i))$ is the BEV feature map extracted from infrastructure point cloud at infrastructure timestamp, $(t_v - t_i)$ is the delta of vehicle and infrastructure timestamp, and $\tilde{F}_i'(t_i)$ is the first order derivative at infrastructure timestamp. This aligned infrastructure feature map is then concatenated with the vehicle feature map and fused using a Conv-Bn-Relu block. Finally, the head of Single Shot Detector (SSD) [Liu+16] is used to perform the 3D object detection task.

This method is trained on the DAIR V2X Dataset [Yu+22a] and tested against the baseline methods: PointPillars[Lan+19], early fusion, late fusion (TCLF [Yu+22a]), and middle fusion (DiscoNet [Li+21]) and V2VNet [Wan+20]). This method outperformed all baseline methods with less than 1/10 the transmission cost of other methods when asynchrony exceeds 200 ms.

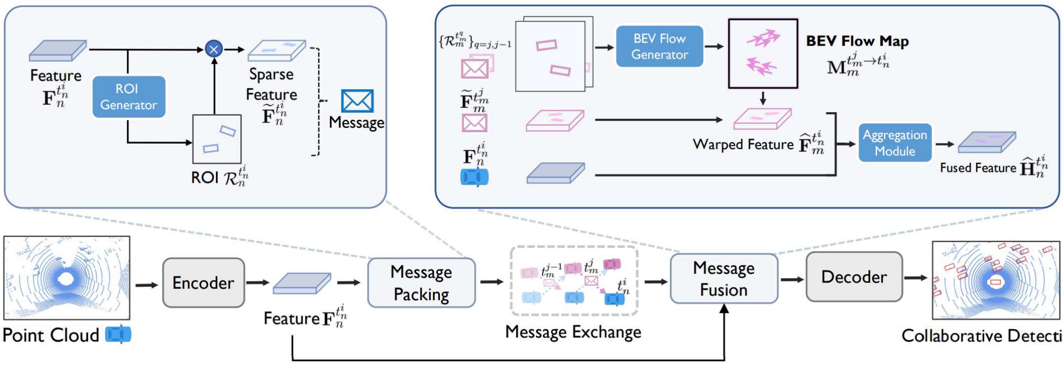

Feature $\mathbf{F}_n^{t_n^i}$
ROI Generator
Sparse Feature $\widetilde{\mathbf{F}}_n^{t_n^i}$
Message
ROI $\mathcal{R}_n^{t_n^i}$
$\{\mathcal{R}_m^{t_m^q}\}_{q=j,j-1}$
BEV Flow Generator
BEV Flow Map
$\mathbf{M}_m^{t_m^j \to t_n^i}$
$\widetilde{\mathbf{F}}_m^{t_m^j}$
$\mathbf{F}_n^{t_n^i}$
Warped Feature $\widehat{\mathbf{F}}_m^{t_n^i}$
Aggregation Module
Fused Feature $\widehat{\mathbf{H}}_n^{t_n^i}$
Point Cloud
Encoder
Feature $\mathbf{F}_n^{t_n^i}$
Message Packing
t_m^{j-1}
t_m^j
t_n^i
Message Exchange
Message Fusion
Decoder
Collaborative Detections

Figure 2.23: Architecture of CoBEVFlow from [Wei+23].

The final state-of-the art method is CoBEVFlow [Wei+23]. This method uses BEV flow instead of feature flow. BEV flow is a collection of motion vectors corresponding to each spatial location. In this method, each agent produces regions of interest (ROI) and a feature map. Using this ROI, each agent produces a sparse feature. This sparse feature and ROI are then sent to the ego-vehicle.

In the ego-vehicle, each non-ego agent's ROI is matched with the ego-vehicle's ROI. Using these matched ROIs, a BEV flow map is generated. This BEV flow map is then used applied to its corresponding non-ego agent sparse feature to align it with the ego-vehicle's feature map. These aligned feature maps are then stacked and fused using a fusion operation. The fusion operation used in the paper's experiments was a multi-scale max operation.

The paper also introduces the Irregular V2V (IRV2V) Dataset. This is a synthetic collaborative perception dataset with various temporal asynchronies that simulate different real-world scenarios. These asynchronies are: a uniformly sampled time shift and a uniformly sampled time turbulence. The method is trained and tested on both the IRV2V and DAIR-V2X [Yu+22a] datasets against the baseline methods: late fusion, DiscoNet [Li+21], V2VNet [Wan+20], V2X-ViT [Xu+22], and Where2Comm [Hu+22]. This method outperforms the baselines and is robust even in heavy asynchrony.

Chapter 3

Dataset

This chapter covers the details of the dataset used in this work. Both the TUMTraf Intersection Dataset [Zim+23b] and a newly created TUMTraf Cooperative Dataset are used. This chapter will mostly cover how the TUMTraf Cooperative Dataset was recorded, how the data was extracted, how the different modalities are matched into one frame, the preprocessing methods applied to it, how the vehicle and infrastructure point clouds are registered to each other, how it is labeled, and finally how the dataset instances in PyTorch are structured. For the same information regarding the TUMTraf Intersection Dataset, refer to its paper [Zim+23b].

3.1 Location

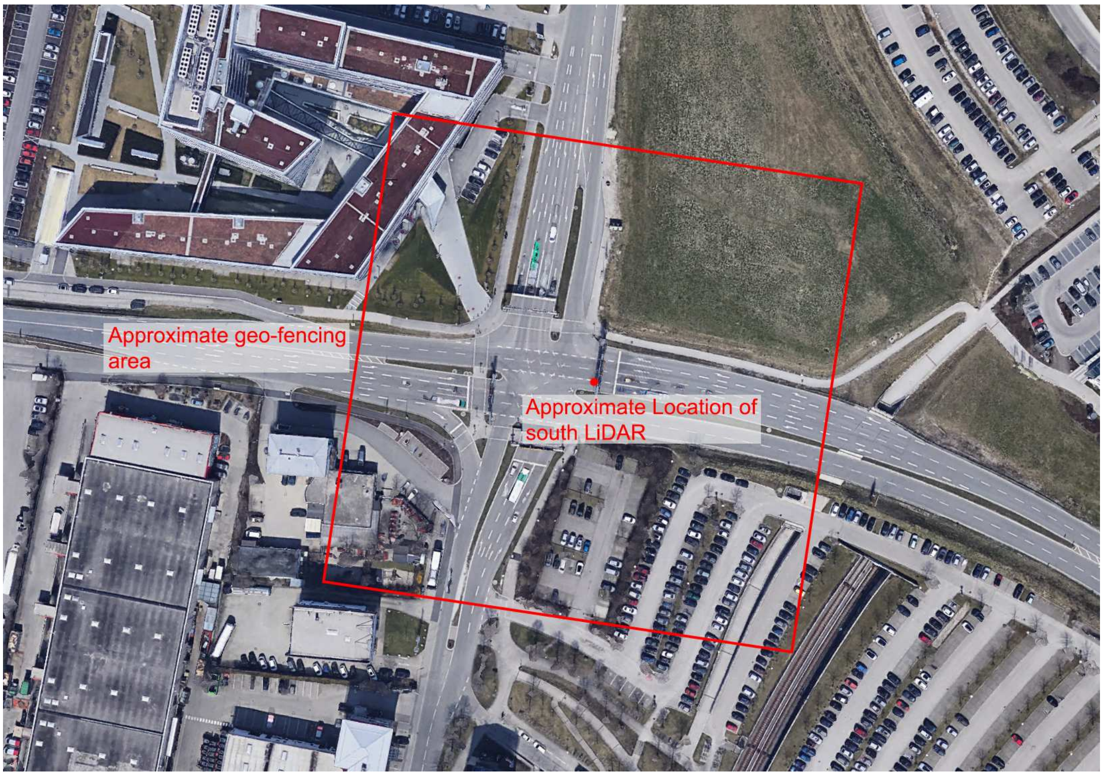

Approximate geo-fencing area
Approximate Location of south LiDAR

Figure 3.1: S110 intersection with the approximate point cloud geo-fencing area and the approximate location of the south LiDAR, which serves as the origin of the infrastructure coordinate system.

The dataset was recorded at the S110 intersection (48.249371, 11.630779) of the Providentia++ project test bed, located in Garching-Hochbrück, Germany. It is the intersection between Zeppelinstraße (north-south direction) and Schleißheimer Straße (east-west direction). There are four traffic signs and lights at each side of the intersection, and on the eastern side, the signs and lights are mounted on a gantry bridge, on which the sensors (cameras and LiDARs) used to collect the data of the datasets are mounted. There is a large variety of traffic participants going through the area since it is an industrial area, including relatively rare vehicles such as car-carrier trailers, tractors, etc.

This intersection was used to collect the data of release R2 (intersection dataset) of the TUMTraf Dataset (formerly A9-Dataset) [Zim+23b]. As mentioned above, this is also one of the datasets that will be used in this work. The dataset consists of 4800 images collected from two cameras and 4800 point clouds collected from two LiDARs. In the experiments, the point cloud created by fusing both LiDAR point clouds will be used. This means that there is a final count of 2400 labeled frames, each containing 2 images from 2 cameras and 1 fused point cloud. In total the dataset contains 38045 registered 3D objects (482 unique objects).

3.2 Roadside Infrastructure

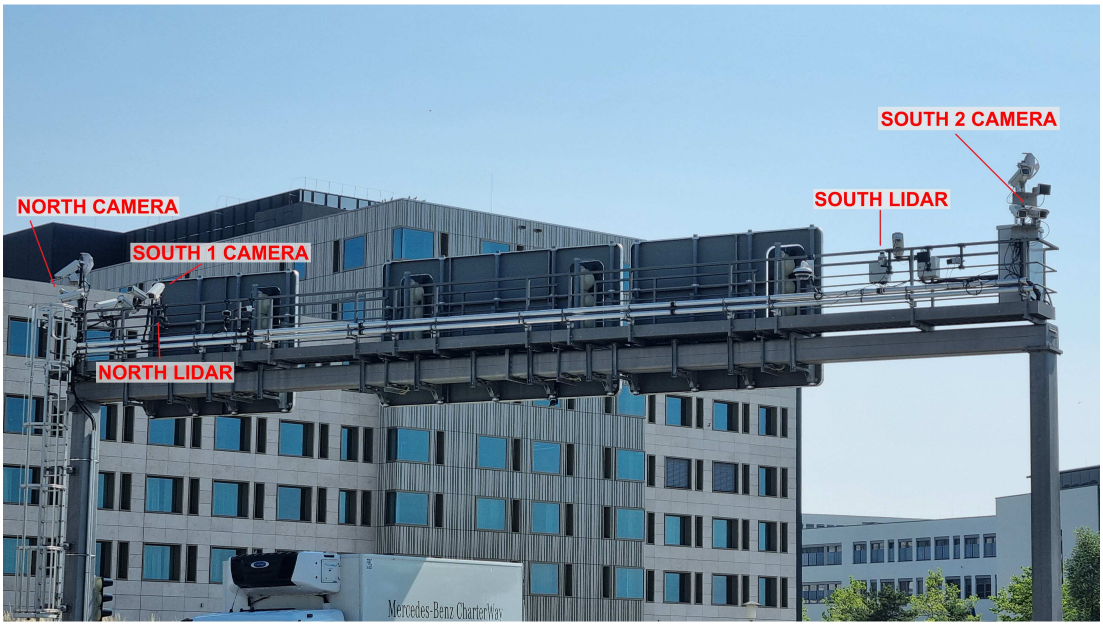


Figure 3.2: S110 gantry bridge with the locations of all of the sensors used in the TUMTraf Cooperative Dataset.

The role of the (roadside) infrastructure node in the TUMTraf Cooperative Dataset is played by the sensors mounted on the gantry bridge at the eastern side of the S110 intersection. There are a lot of sensors mounted on this gantry bridge, e.g. radar, event-based cameras, RGB cameras, and LiDARs. However, the work will utilize 3 cameras (north, south 1, and south 2) and two LiDARs (north and south). On top of this gantry bridge on the side of the south 2 camera and south LiDAR is a switch box that handles the routing of data from the sensors. At the bottom of the gantry bridge, near the northern feet of the gantry bridge is the Road Side Unit (RSU), which serves as the brain of the infrastructure node.

The coordinate system of the infrastructure node is based on and originates from the south LiDAR sensor. As such, the front direction (x-axis) roughly points to the west, the y-axis to the south, and the z-axis points upwards. The cameras have an x-axis pointing to its forward, a y-axis pointing to its right, and a z-axis pointing downward.

3.3 Recording Vehicle

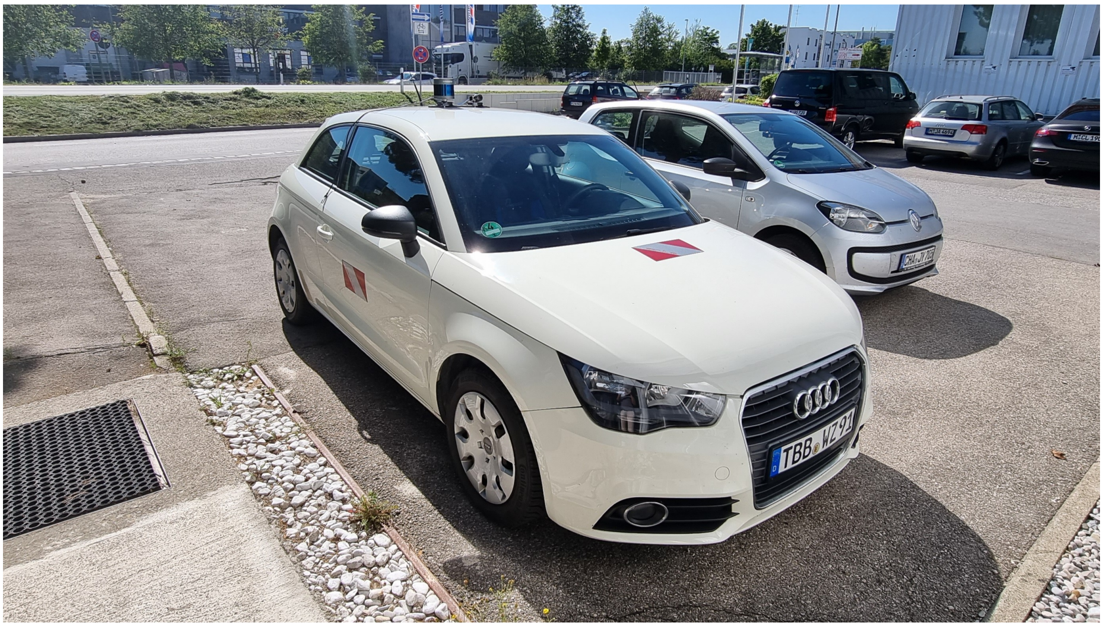


Figure 3.3: The vehicle used to record vehicle data for the TUMTraf Cooperative Dataset.

The role of the vehicle node in the TUMTraf Cooperative Dataset is played by an Audi A1. Two sensors are mounted on the roof of the vehicle, one LiDAR sensor at the center right side, and one camera at the center left side of the vehicle. Magnetic markings are placed around the vehicle to make it easier to locate the recording vehicle in the infrastructure camera images when extracting sequences from the recordings.

The coordinate system of the vehicle node is based on and originates at the vehicle LiDAR sensor. The front direction (x-axis) points to the front of the car, the y-axis points to the left of the car, and the z-axis points upward. The cameras have an x-axis pointing to the front of the car, a y-axis pointing to the right of the car, and a z-axis pointing downward.

3.4 Sensors

There is a wide variety of sensors that are used in autonomous/cooperative driving today. Some of the common ones are the following: RGB camera, radar, LiDAR, GPS, and IMU. The method works using RGB camera and LiDAR by fusing them together. As a result, the data are recorded using these two sensor types.

3.4.1 Camera

A camera is one of the most common types of sensors used in autonomous driving. They are cheap and one of the smaller sensor types by dimension. Cameras can provide a stream of high-resolution images. They contain information about the shapes and textures of the objects in the scene. As such, it is useful in identifying types of objects, such as vehicles, buses, trucks, bicycles, etc. In the case of RGB cameras, these images also contain information about colors, which could be useful for tasks such as traffic light detection. Based on the lens used by the camera, one could also customize the field of view. A narrow field of view gives us more zoom and details of the objects in the scene, while a wide field of view allows us to detect objects in a larger area.

Figure 3.4: Basler ace acA1920-50gc cameras.

One of the main disadvantages of cameras is the lack of depth information. Of course, one could estimate depth from images using a variety of techniques based on the amount of cameras. But these are still estimated depth values. Another weakness of the camera is that it is affected by lighting. As shown in the Introduction, when an image is overexposed or underexposed, a lot of information will be lost. Shadows, lens glare, etc. could also adversely affect the quality of information contained in the image. Another weakness is the weather. Rain or fog will affect the distance objects would be visible. As a result of these weaknesses, cameras are mostly used together with other sensors such as LiDARs.

The cameras used in the TUMTraf Cooperative Dataset are Basler ace acA1920-50gc cameras with Sony IMX174 sensor, generating 1920×1200 color images. The intersection images are recorded using 8 mm lenses (south 1 and 2, and north), while the vehicle images are recorded using a 16 mm lens.

3.4.2 LiDAR

Different from cameras, LiDAR (Light Detection and Ranging) uses laser beams in order to generate a scan of its surroundings. The LiDAR generates a point cloud, where each point is the location where a laser beam hits an object. The distance is calculated based on the time of flight, using the formula $d = ct/2$, where c is the speed of light and t is the time difference between when the laser beam is shot and when its return is received. As such, unlike cameras, LiDAR point clouds have very accurate depth information since it is measured and not merely estimated like with images. On top of this, LiDAR returns also include intensity information. This information can be used to infer the type of material the laser beam hits as different material produces returns with different intensities. This accuracy and intensity information is very helpful in 3D object detection and tracking tasks, especially in the context of autonomous driving.

On top of this, LiDARs (depending on the model used) are able to create highly dense point clouds. This high density/resolution means that more granular details, such as edges and corners, are available for the 3D object detections to work with. Certain models of LiDAR sensors are also able to generate point clouds in a 360-degree field of view. Finally, LiDARs are unaffected by lighting conditions, meaning they work equally well both during the day and at night, no matter how bright or dark. This is a very advantageous characteristic in the context of autonomous driving.

However, LiDARs are not perfect and they still have weaknesses. First, LiDAR sensors are generally more expensive than cameras. Some LiDAR models (e.g. compact LiDARs like the Blickfeld Cube-1) also have a comparatively limited field-of-view (100 × 30 degrees). Another weakness of LiDAR is related to its usage of laser beams. Lasers, high energy light, are affected by reflection and other physical phenomena due to atmospheric conditions. Black vehicles have low reflectivity and as such are often difficult to detect using LiDARs. Rain, fog, etc. will affect detection quality since laser beams hitting water particles/droplets will create returns, which in turn create noisy point clouds. This is why, as good as LiDAR sensors are, it is still a good idea to create redundancy by pairing them with other sensors, such as cameras.

Figure 3.5: Ouster OS1-64 (gen. 2) and Robosense RS-LiDAR-32.

There are two types of LiDARs used in the TUMTraf Cooperative Dataset. The infrastructure part of the TUMTraf Cooperative Dataset is recorded using two Ouster OS1-64 (gen. 2) LiDARs. They have a 360 × 45 degree FOV, 64 vertical layers, 120 m range, and 1.5 - 10 cm accuracy. Additionally, they are set up for below-horizon operation because they are mounted high up and have to record objects that are located below them. The vehicle part of the cooperative dataset is recorded using a Robosense RS-LiDAR-32. It has a 360 × 40 degree FOV, 32 vertical layers, up to 200 m range, and ca. 5 cm accuracy.

3.5 Data Collection, Preparation, and Labeling

The data for the TUMTraf Cooperative Dataset was recorded on July 6th, 2023. To do the recording, ROS and open-source ROS drivers for the sensors were used. To help with synchronization, all sensors are synchronized to a NTP server. Multiple drives through the intersection were recorded with the vehicle going in different directions and recording different scenarios (such as highly occluded scenarios and u-turns) in each drive. The results arrive as two rosbags, each containing the images and point clouds of the infrastructure and vehicle nodes.

The images and point clouds are then extracted from these rosbags. The images and point clouds are matched to create data frames using soft synchronization based on the timestamp in their file names. Out of all the drives recorded, 5 × 10 seconds scenes are selected based on how interesting and challenging they are. The selected scenes all feature occlusion and difficult-to-detect objects such as pedestrians, bicycles, and trucks pulling trailers carrying multiple cars.

The infrastructure and vehicle point clouds of each scene are then registered to each other to create a registered point cloud and per-frame transformation matrix from vehicle to infrastructure perspective. The registration is done using a combination of automatic registration using the Open3D library and its point-to-point ICP function, manual registration by hand using Blender, and interpolation. Automatic registration was done for the keyframes with manual registration to fix the sub-optimal results, while the in-between frames were registered using interpolation based on the keyframes. The point clouds are then cleaned from noise, while the images are undistorted using the TUMTraf Dataset Devkit [Zim+23b]. Then, the 500 frames are divided between 5 people, each labeling 100 frames manually using the proAnno annotation tool, which is based on the 3D-BAT annotation tool [ZRT19]. The labels are created based on the registered vehicle-infrastructure point cloud and guided by the images. This creates the final ASAM OpenLABEL format point cloud label files.

3.6 Dataset Statistics

This section covers the statistics of both datasets used in this work. Since the TUMTraf Cooperative Dataset is set to be as similar to other TUMTraf datasets (including the TUMTraf Intersection dataset) as possible, the TUMTraf Cooperative Dataset was also labelled using the same classes from TUMTraf datasets: CAR, VAN, BICYCLE, PEDESTRIAN, MOTORCYCLE, TRAILER, TRUCK, BUS, EMERGENCY_VEHICLE, and OTHER.

3.6.1 TUMTraf Intersection Dataset

The TUMTraf Intersection Dataset is the third release of the TUMTraf datasets (previously the A9-Dataset) [Zim+23b]. This dataset consists of 2400 data frames, each containing two LiDAR point clouds, two camera images, and a label file. The data was collected using two Ouster OS1-64 (gen. 2) LiDARs and two Basler ace acA1920-50gc cameras with Sony IMX174 sensor. The data is also collected at the S110 intersection on the Providentia++ testbed.

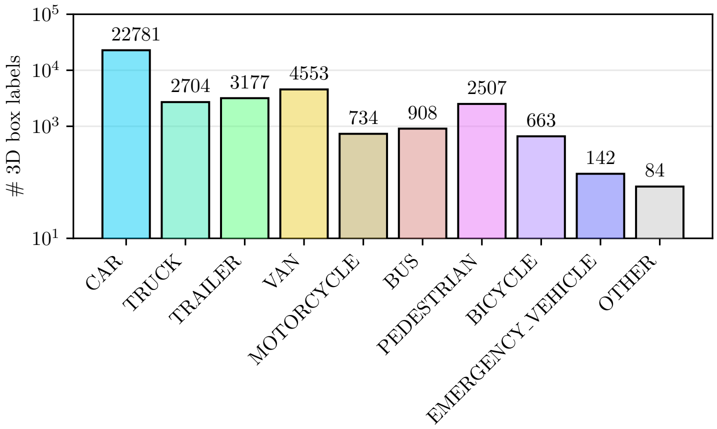


Figure 3.6: Class distribution of the entire TUMTraf Infrastructure Dataset.

The dataset is split into train, validation, and test sets in an 80-10-10 split. Considering the full amount of frames and this split, the train set contains 1920 data frames, the validation set 240 data frames, and the test set 240 data frames. Because this dataset is used to compare the proposed method with the existing infrastructure-only camera-LiDAR late fusion method used that is the current best method for the TUMTraf Intersection Dataset

(InfraDet3D [Zim+23a]). The split used will be this paper's original train, validation, and test split. This split was created using random sampling.

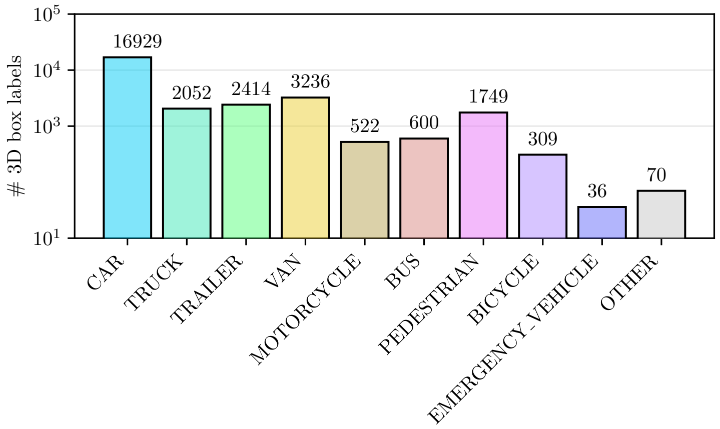


Figure 3.7: Class distribution of the train set of the TUMTraf Infrastructure Dataset.

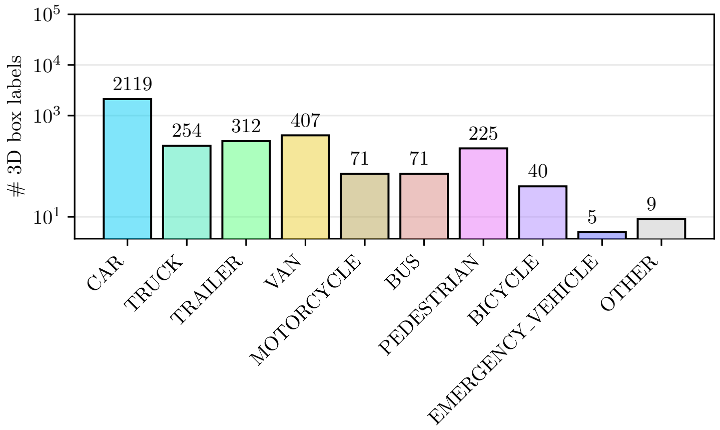


Figure 3.8: Class distribution of the validation set of the TUMTraf Infrastructure Dataset.

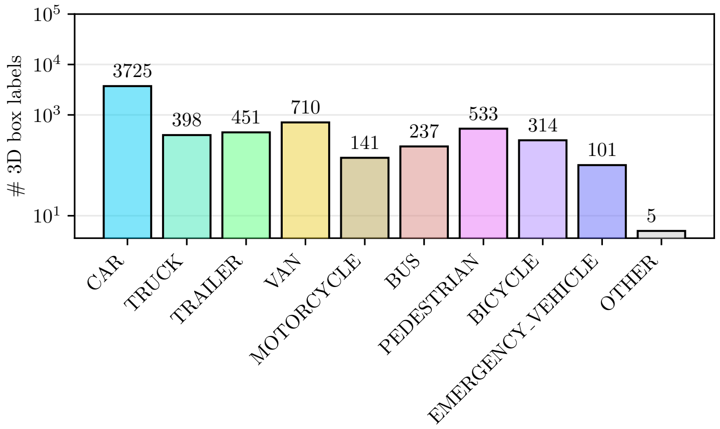


Figure 3.9: Class distribution of the test set of the TUMTraf Infrastructure Dataset. Note the noticeably different distribution in the test set for the classes BICYCLE, BUS, and EMERGENCY_VEHICLE compared to the train and validation sets because of the random sampling.

Figure 3.6 shows the class distribution in the whole TUMTraf Infrastructure Dataset, including the train, validation, and test sets. Figures 3.7 to 3.9 show the class distributions of the train, validation, and test sets. Note the very different distributions for the classes BICYCLE, EMERGENCY_VEHICLE, and to some extent BUS in the train and val sets compared to the test set. This is caused by the random sampling used when splitting the dataset. This can be solved by using a different sampling method like stratified sampling [Wik23].

3.6.2 TUMTraf Cooperative Dataset

The TUMTraf Cooperative Dataset was created from scratch for this work. The dataset consists of 500 data frames, each containing two LiDAR point clouds (infrastructure and vehicle), three infrastructure camera images, one vehicle camera image, and a label file, which also contains the vehicle-to-infrastructure point cloud transformation matrix.

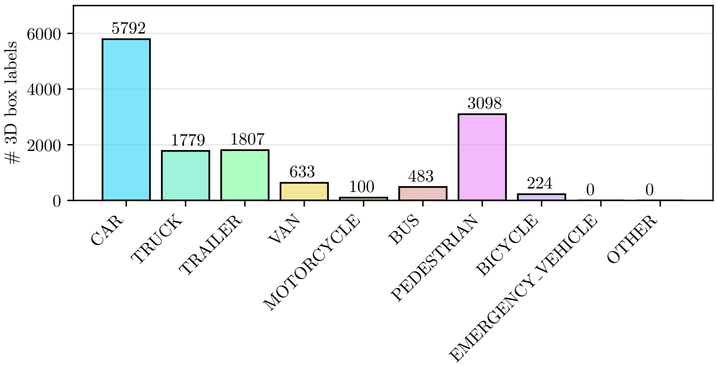


Figure 3.10: Class distribution of the entire TUMTraf Cooperative Dataset.

The dataset is also split into train, validation, and test sets in an 80-10-10 split. Considering the full amount of frames and this split, the train set contains 400 data frames, the validation set 50 data frames, and the test set 50 data frames. This dataset is entirely split using stratified sampling [Wik23], ensuring a similar class distribution between the train, validation, and test sets. Figure 3.10 shows the class distribution in the whole TUMTraf Infrastructure Dataset, including the train, validation, and test sets. Figures 3.11 to 3.13 show the class distributions of the train, validation, and test sets. Note the very similar class distributions between the train, validation, and test sets, unlike the split of the TUMTraf Infrastructure Dataset.

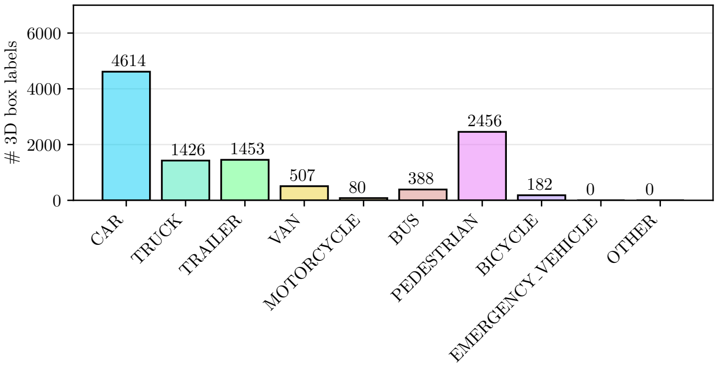


Figure 3.11: Class distribution of the train set of the TUMTraf Infrastructure Dataset.

Unlike the TUMTraf Infrastructure Dataset, this dataset is very limited by the very small amount of data that it contains in the context of deep learning model training. As a result,

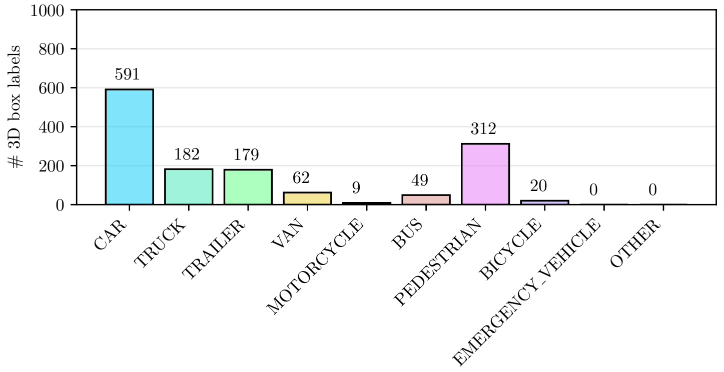


Figure 3.12: Class distribution of the validation set of the TUMTraf Infrastructure Dataset.

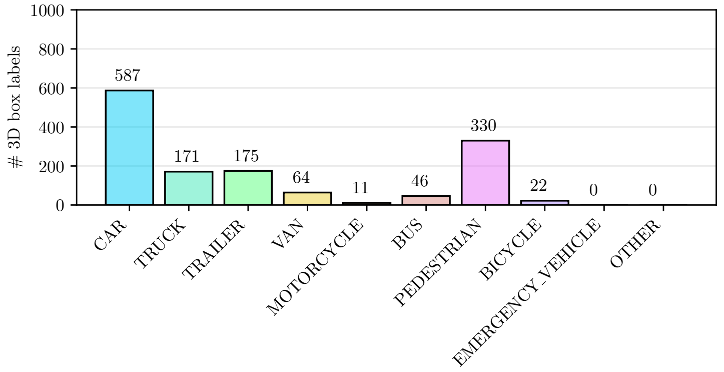


Figure 3.13: Class distribution of the test set of the TUMTraf Cooperative Dataset. Note the very similar class distribution in this test set to the train and validation sets thanks to the usage of stratified sampling [Wik23].

the results that will be shown in Chapter 5 are under the assumption of heavy overfitting by the model, the solutions to which will be presented in Chapter 7. On top of that, two classes do not exist in this dataset (EMERGENCY_VEHICLE and OTHER), while the MOTORCYCLE class has such a small population that the model proposed in this work are not able to learn it well enough, resulting in all configurations of the proposed method to have 0 class mAP during testing. As a result, all of the experiments in this work are done while excluding the three classes.

Chapter 4

Implementation

This chapter provides an explanation of the implementation of the method proposed in this work. Since the method is a cooperative multi-modal deep fusion method, it can be divided into the multi-modal deep fusion part, which will happen separately in the (theoretical) vehicle and infrastructure nodes, and the cooperative deep-fusion part, which happens after in the infrastructure node. The baseline and starting code base of this proposed method is BEVFusion [Liu+23b] for multi-modal (camera-LiDAR) deep fusion, with vehicle-infrastructure deep fusion method inspired by PillarGrid [Bai+22b]. BEVFusion's code base was used as the starting point to save up on implementation time. Because of this, this code is written using MMCV, MMDetection, and MMDetection3D, which are built on top of PyTorch since BEVFusion's code base uses them.

Ideally, this all happens live where both a real vehicle and a real infrastructure node (at its RSU) do their part of camera-LiDAR fusion separately. Then the vehicle sends the fused camera-LiDAR feature to the infrastructure node's RSU, where the vehicle-infrastructure fusion and 3D object detection happen. Finally, the resulting list of detection is then sent back to the vehicle node.

However, due to time constraints and the significant amount of extra problems that need to be solved before this could happen, e.g. live estimation of vehicle-infrastructure transformation matrix, fast enough V2I (and I2V) communications between vehicle and infrastructure to allow real-time (higher than 10 FPS) operation, synchronization between vehicle and infrastructure, etc., the current implementation is simplified and written for offline operation using dataset and data loaders, where everything happens in the same computer and the same GPU.

4.1 Data Preparation

As the first step, the data contained in the datasets need to be formatted into a .pkl file that will then be used by the dataset and data loader. BEVFusion was originally trained on the nuScenes [Cae+20] dataset and as such expects .pkl files and data frames that are formatted in nuScenes format. The datasets are in the ASAM OpenLABEL format, so the first step is to create a converter that converts OpenLABEL to nuScenes format. The ASAM OpenLABEL format is selected because it is the best format that can be used in the future to get rid of all the conversions, parsings, and transformations that are still necessary today due to the lack of format standardization.

The main challenge for doing this is finding a replacement for the ***token*** system used in nuScenes to uniquely identify frames, which in turn is also used in everything nuScenes does, including critical parts such as the nuScenes evaluation protocol used by BEVFusion to calculate loss, mAP, etc. during training and evaluation. The solution is to replace tokens with ***timestamps***. Each frame in the data set has a unique timestamp instead of tokens. Other than this, this conversion is relatively straightforward where the right data simply needs to be placed in the right location in the nuScenes format data frame.

4.1.1 TUMTraf Intersection Dataset

For the TUMTraf Intersection Dataset (previously A9-Dataset R2), this is done using the script *tools/create_a9_data.py*, which uses the converter class *tools/data_converter/a9_converter.py* and the ground truth database creator *tools/data_converter/create_a9_gt_database.py*. The database is created for one of the augmentations used by BEVFusion during training: *Object-Paste*, which will be covered in more detail in the Data Pipeline subsection.

The goal of this script is to convert the ASAM OpenLABEL format into a nuScenes-like format for the TUMTraf Intersection Dataset's case, which is named A9NuscDataset. The old name A9 was used since this code was written prior to the recent renaming of the dataset. The structure of a data frame of this type of dataset is shown in Listing 4.1.

```
{
    "lidar_path": pcd_path,
    "lidar_anno_path": pcd_label_path,
    "sweeps": [],
    "cams": {
        "s110_camera_basler_south1_8mm": {
            "data_path": img_south1_path,
            "type": "s110_camera_basler_south1_8mm",
            "lidar2image": lidar2s1image,
            "sensor2ego": south12ego,
            "sensor2lidar": south12lidar,
            "camera_intrinsics": south1intrinsics,
            "timestamp": timestamp_from_label,
        },
        "s110_camera_basler_south2_8mm": {
            "data_path": img_south2_path,
            "type": "s110_camera_basler_south2_8mm",
            "lidar2image": lidar2s2image,
            "sensor2ego": south22ego,
            "sensor2lidar": south22lidar,
            "camera_intrinsics": south2intrinsics,
            "timestamp": timestamp_from_label,
        }
    }
    "lidar2ego": lidar2ego,
    "timestamp": timestamp_from_label,
    "location": location_from_label,
    "gt_boxes": list_of_gt_boxes,
    "gt_names": list_of_gt_names,
    "gt_velocity": list_of_gt_velocities,
    "num_lidar_pts": list_of_num_lidar_pts,
    "num_radar_pts": list_of_num_radar_pts,
    "valid_flag": list_of_valid_flags,
}
```

Listing 4.1: Example data frame of a A9NuscDataset dataset

In this data frame, there is only one point cloud per data frame since the fused/registered point cloud from north and south LiDARs contained in the dataset is used. This point cloud is also first converted from .pcd files to .bin files since this is what BEVFusion used. During this conversion, only the x, y, z, and intensity values of each point are saved since those are the only information used in this method. As such, this point cloud path points to the newly created .bin files. This fused point cloud is centered around the south LiDAR. Sweeps will always be an empty list, since the TUMTraf Intersection Dataset does not have extra LiDAR sweeps associated with a LiDAR point cloud like nuScenes.

lidar2s1image and *lidar2s2image* are projection matrices from the south LiDAR to the south 1 and 2 images. *south12ego* and *south22ego* are transformation matrices from south 1 and 2 cameras to the ego, which is S110 base. *south12lidar* and *south22lidar* are transformation matrices from south 1 and 2 cameras to the south LiDAR. *south1intrinsics* and *south2intrinsics* are the intrinsic matrices of the south 1 and 2 cameras. Finally, *lidar2ego* is the transformation matrix from south LiDAR to the ego, which is the S110 base. All of these matrices are currently hard-coded in the converter class, but in the future, they should be readable from the point cloud label files.

The fields *gt_boxes*, *gt_names*, *gt_velocity*, *num_lidar_pts*, *num_radar_pts*, *valid_flag* are all related to ground truth information and, as such, only available in the data frames of the training and validation sets. All of this information are either obtainable from the point cloud label files or hard-coded in the converter class. The hard-coded ones are *gt_velocity*, which is a list filled with [0, 0]s since velocity information is not used, *num_radar_pts*, which is a list of zeros since we only work with LiDAR data, and *valid_flag*, which is a list of Trues since all labels are used.

```
{
    "class_name_1": [{
        "name": class_of_object,
        "path": rel_filepath_to_bin,
        "gt_idx": i,
        "box3d_lidar": LiDARInstance3DBoxes_instance,
        "num_points_in_gt": num_points_in_object,
        "difficulty": difficulty_of_object,
    }],
    "class_name_2": [{
        "name": class_of_object,
        "path": rel_filepath_to_bin,
        "gt_idx": i,
        "box3d_lidar": LiDARInstance3DBoxes_instance,
        "num_points_in_gt": num_points_in_object,
        "difficulty": difficulty_of_object,
    }]
}
```

Listing 4.2: Example data frame of a A9NuscDataset ground truth database

Other than creating .pkl files that contain the appropriate data frames, as mentioned before, this script also creates a ground truth database .pkl file and directory filled with .bin files of each ground truth object for use by the *ObjectPaste* augmentation. This ground truth database creator works by first loading the dataset as an A9NuscDataset. It then loads the annotations of this particular data frame, which contains information like *gt_bboxes_3d*, which is a list of *LiDARInstance3DBoxes*, which is a data type in MMDetection3D for the 3D bounding boxes of LiDAR labels, *gt_labels_3d*, which is a list of integers representing the classes of the labels, and *gt_names_3d*, which is a list of the classes of the labels as strings.

Using this information, the database .bin files are created by using the *gt_bboxes_3d* information of the current label file, finding the points inside the 3D bounding boxes and storing them as separate .bin files with *{point_cloud_index}_{class}_{i}.bin* as the naming convention, where i is the current index of the for loop over all bounding boxes in *gt_bboxes_3d* of the label file. Meanwhile, the ground truth database .pkl file contains frames for each current label file, which are structured as shown in Listing 4.2.

Each frame will contain 9 key-value pairs. The keys are the 9 class names in the dataset, while the values are lists of JSON objects containing the class of a ground truth object, the path to the .bin file, its index in the list of *gt_bboxes_3d* of the label file, the LiDARInstance3DBoxes instance taken from *gt_bboxes_3d*, the number of points in this ground truth object and the difficulty, which will be zero for everything since the label files do not contain difficulty information.

4.1.2 TUMTraf Cooperative Dataset

For the TUMTraf Cooperative Dataset, a similar protocol to the TUMTraf Intersection Dataset version is followed. Creating .pkl files for the train, validation, and test sets, and the ground truth database .pkl file plus the ground truth object .bin files on top. The script used is *create_a9coop_data.py*, which uses the converter class *a9coop_converter.py* and the ground truth database creator *create_a9_gt_database.py*, which is the same one as TUMTraf Intersection Dataset. These scripts are all located in the same directory as their TUMTraf Intersection Dataset equivalents shown in the previous subsection. The same nuScenes style format as before is also used but with some changes to the data frame layout. This version of the dataset is called A9NuscCoopDataset. The modified layout is shown in Listing 4.3.

```
{
    "vehicle_lidar_path": vehicle_pcd_path,
    "vehicle_sweeps": [],
    "infrastructure_lidar_path": infrastructure_pcd_path,
    "infrastructure_sweeps": [],
    "registered_lidar_path": registered_pcd_path,
    "registered_sweeps": [],
    "lidar_anno_path": pcd_label_path,
    "vehicle2infrastructure": vehicle2infrastructure
    "vehicle_cams": {
        "vehicle_camera_basler_16south1_8mm": {
            "data_path": img_vehiclesouth1_path,
            "type": "vehicle_camera_basler_16south1_8mm",
            "lidar2image": vehiclelidar2s1image,
            "sensor2lidar": vehiclecamsouth12lidar,
            "camera_intrinsics": vehiclecamsouth1intrinsics,
            "timestamp": timestamp_from_label,
        }
    }
    "infrastructure_cams": {
        "s110_camera_basler_south1_8mm": {
            "data_path": img_south1_path,
            "type": "s110_camera_basler_south1_8mm",
            "lidar2image": infralidar2s1image,
            "sensor2lidar": south12infralidar,
            "camera_intrinsics": south1intrinsics,
            "timestamp": timestamp_from_label,
        },
        "s110_camera_basler_south2_8mm": {
            "data_path": img_south2_path,
            "type": "s110_camera_basler_south2_8mm",
            "lidar2image": infralidar2s2image,
            "sensor2lidar": south22infralidar,
            "camera_intrinsics": south2intrinsics,
            "timestamp": timestamp_from_label,
        },
        "s110_camera_basler_north_16mm": {
            "data_path": img_snorth_path,
            "type": "s110_camera_basler_north_16mm",
            "lidar2image": infralidar2n1image,
            "sensor2lidar": north2infralidar,
            "camera_intrinsics": northintrinsics,
            "timestamp": timestamp_from_label,
        }
    }
    "timestamp": timestamp_from_label,
    "location": location_from_label,
    "gt_boxes": list_of_gt_boxes,
    "gt_names": list_of_gt_names,
    "gt_velocity": list_of_gt_velocities,
    "num_lidar_pts": list_of_num_lidar_pts,
    "num_radar_pts": list_of_num_radar_pts,
    "valid_flag": list_of_valid_flags,
}
```

Listing 4.3: Example data frame of a A9NuscCoopDataset dataset

The structure of the cooperative data frame is generally the same as the TUMTraf Intersection Dataset version. It can be thought of as combining two such data frames into one, one for the vehicle node and one for the infrastructure. Another difference is that transformation matrices to ego (previously S110 base) are removed. This is because the TUMTraf Cooperative Dataset doesn't have it and BEVFusion doesn't actually use it. It was originally included because it is just something nuScenes data frames include, which was inherited by BEVFusion, and as a result, also inherited by the TUMTraf Intersection Dataset version of this method.

The ground truth database of the TUMTraf Cooperative Dataset is effectively the same as the previous version, so this will not cover it again. The only difference is that now the points to create the ground truth .bin files come from the point cloud shown by the path *registered_lidar_path*, which is the registered vehicle and LiDAR point clouds to make sure that the ground truth objects contain the most complete set of points, and thus information available.

4.2 Data Loading

The next things that needed to be created were the PyTorch MMDetection3D dataset classes. BEVFusion uses a nuScenes dataset class that is adapted from the one provided by MMDetection3D. The dataset classes are in turn adapted from the one used by BEVFusion, both modified to fulfill the datasets' specific use cases.

4.2.1 TUMTraf Intersection Dataset

The first modification that needs to be done for the TUMTraf Intersection Dataset is to change the object classes considered in the dataset class. Instead of the nuScenes classes, they were changed to the classes of the datasets used in the experiments: CAR, TRAILER, TRUCK, VAN, PEDESTRIAN, BUS, MOTORCYCLE, OTHER, BICYCLE, and EMERGENCY_VEHICLE. nuScenes originally uses settings defined in [Inc19a] for the 3D object detection task. The same values are used, except for class ranges. CAR, TRUCK, BUS, TRAILER, VAN, and EMERGENCY_VEHICLE are set at 50 m. PEDESTRIAN, MOTORCYCLE, and BICYCLE are set to 40 m. Finally, OTHER is set at 30 m.

The function *get_data_info* is also modified to work with the fields *lidar_path*, *sweeps*, *lidar2ego*, *timestamp*, and *location* for LiDAR and general information; *image_paths*, *camera2lidar*, *lidar2camera*, *lidar2image*, *camera2ego*, *camera_intrinsics*, and *camera2lidar* for the cameras; and *ann_info* for the annotation info if not in testing mode. *lidar2camera* is obtained by inversing the *camera2lidar* information from the data frame.

ann_info is obtained using the function *get_ann_info*. This function is relatively simple. First, it checks if *valid_flag* is used. If yes, a mask for filtering the annotations based on the information contained in *valid_flag* is created. Otherwise, it will choose annotations with *num_lidar_pts* larger than zero. *valid_flag* is always used and since it was set to contain only Trues earlier, all annotations are always loaded.

This function then loads the fields *gt_bboxes_3d*, *gt_labels_3d*, and *gt_names*. *gt_bboxes_3d* is a list of LiDARInstance3DBoxes obtained using the information contained in the *gt_boxes* and *gt_velocity* information contained in the data frame. These new LiDARInstance3DBoxes instances are set up with origin at (0.5, 0.5, 0.5). *gt_labels_3d* is a list of the labels of the annotations as integers based index of the class inside the tuple CLASSES containing the classes in the order CAR, TRAILER, TRUCK, VAN, PEDESTRIAN, BUS, MOTORCYCLE, OTHER, BICYCLE, and EMERGENCY_VEHICLE. *gt_names* is a list of the labels of the annotations as strings.

Finally, the evaluation protocol used in the dataset class is also modified. BEVFusion uses the nuScenes evaluation protocol for this purpose. nuScenes calculates the per-class average precision, translation error, scale error, orientation error, and velocity error. On top of that, it also calculates the means of these metrics across all classes. Finally, it also calculates the nuScenes Detection Score. The explanation of these metrics will be covered in Chapter 5.

As mentioned earlier, nuScenes uses the token system in everything it does and it includes the evaluation protocol. As such, it was necessary to reverse-engineer the entire evaluation protocol and modify it to work using timestamps instead. All of the functions mentioned next are contained in the nuScenes dataset dev-kit repository [Inc19b]. The functions that needed to be modified were *load_prediction*, *load_gt*, *accumulate*, and *filter_eval_boxes*. The changes were minimal (changing every usage of token to timestamp) and the main bulk of the work was retracing everything to find all the functions used.

Other than these modified functions, the functions *_evaluate_a9_nusc*, *serializeMetricData*, *calc_ap*, *calc_tp*, *cummean*, *center_distance*, *velocity_l2*, *yaw_diff*, *angle_diff*, and *scale_iou* are copied into the dataset class. This is done because due to the modifications, everything now needs to run inside the dataset class instead of being able to import an existing implementation. With all of these modifications, a TUMTraf Intersection Dataset that works in ways similar to nuScenes is now created.

4.2.2 TUMTraf Cooperative Dataset

The dataset class for the TUMTraf Cooperative Dataset is almost exactly the same as the dataset class for the TUMTraf Intersection Dataset. The overall settings, the classes being detected, the *get_ann_info* function, and the evaluation protocol are the same. The only difference is in how the dataset handles data because now there are both vehicle and infrastructure data instead of just infrastructure.

The *get_data_info* function now loads these fields for general and LiDAR data: *timestamp*, *location*, *vehicle2infrastructure*, and the vehicle, infrastructure, and registered versions of *lidar_path* and *sweeps*. For camera data, the function now loads *image_paths*, *camera2lidar*, *lidar2camera*, *lidar2image*, *camera2ego*, *camera_intrinsics*, and *camera2lidar* for the four cameras of vehicle and infrastructure nodes. This function also loads *ann_info* for the annotation info if not in testing mode.

4.3 Data Pipeline

There are two versions of the data pipeline used in the proposed method: one for the TUMTraf Intersection Dataset and one for the TUMTraf Cooperative Dataset. They need to be differentiated because everything (loading, augmentation, preprocessing, etc.) in the cooperative version needs to handle two nodes of data at once (vehicle and infrastructure) instead of just one for the TUMTraf Intersection dataset. There is also one data augmentation method that couldn't be brought to a usable state due to time limitations. All of this information will be covered in their respective subsections below.

4.3.1 TUMTraf Intersection Dataset

The TUMTraf Intersection Dataset's data pipeline is practically the same as BEVFusion's pipeline with some alterations to the parameters of certain data augmentation methods. There are three variants of this data pipeline, one each for training, validation, and testing.

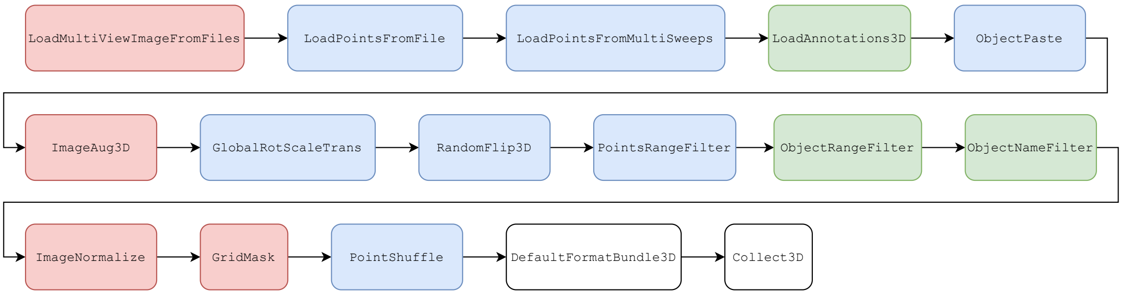


Figure 4.1: TUMTraf Intersection Dataset training data pipeline. The red methods handle image data, the blue methods handle point cloud data, the green methods handle the annotations, and the white methods are general methods that format and collect the necessary information for the model.

The training pipeline starts with loading LiDAR, image, and annotations with the functions *LoadMultiViewImageFromFiles*, *LoadPointsFromFile*, *LoadPointsFromMultiSweeps*, and *LoadAnnotations3D*. *LoadPointsFromFile* is to load the points of the point cloud from the .bin file. During loading, we set five as the amount of dimensions to be loaded and used. The amount of beams is also reduced to 32. *LoadPointsFromMultiSweeps* can basically be ignored since there are no extra sweeps. *LoadAnnotations3D* loads the annotation data minus attributes.

After all the images and point clouds are loaded, data augmentation comes next. The first augmentation used is *ObjectPaste*. This augmentation loads ground truth objects from the ground truth database created earlier and pastes them inside the original point cloud. Before pasting, several objects are sampled from the database. Then, based on the sampled object's 3D bounding box, the method checks if the sampled database object's bounding box collides with any original ground truth bounding box or not. Only sampled objects with no collisions will be used. Then, during pasting, any existing points from the original point cloud that exists within the sampled object's bounding box are removed and the points from the sampled object are concatenated into the original point cloud.

In this method, while only objects with 5 points or more are pasted similar to BEVFUsion, the sampling parameters are modified from BEVFusion's original parameters. The parameters used CAR: 2, TRAILER: 5, TRUCK: 3, VAN: 3, PEDESTRIAN: 7, BUS: 5, MOTORCYCLE: 5, OTHER: 2, and BICYCLE: 7. These values are selected because there are enough CAR, TRUCK, and VAN objects in the original data. The performance in OTHER objects was also good enough as is during preliminary testing. Meanwhile, more samples of TRAILER, BUS, PEDESTRIAN, MOTORCYCLE, and BICYCLE are needed based on the per-class performance results during preliminary testing using the original BEVFusion values [Liu+23a].

The next three augmentations involve affine transformations of the image and point cloud data. *ImageAug3D* is used to augment image data. The operations applied here are resize, crop, flip, and rotate. Resize is limited to $[0.38, 0.55]$ times for height and $[0.48, 0.48]$ times for width. Rotation is limited to $[-5.4, 5.4]$ degrees. This augmentation also calculates the 3D transformation matrix caused by these operations and saves it as *img_aug_matrix* for usage further down the line. This method also ensures that the image is resized from 1200 × 1920 to 256 × 704, These values were selected by the BEVFusion authors after they found that performance plateaus when using images larger than 256 × 704.

GlobalRotScaleTrans and *RandomFlip3D* augment the point cloud data. *GlobalRotScaleTrans* applies random global rotation, scaling, and translation to the point cloud, while *RandomFlip3D* randomly flips the point cloud along the x and y-axis. Scaling is limited to $[0.9, 1.1]$, rotation to $[-0.78539816, 0.78539816]$ radians, and translation to 0.5 m. All of these methods also save the 3D transformation matrices caused by these operations as *lidar_aug_matrix* to be used further down the line.

The next three methods filter the point cloud and annotations based on range and the class names to geo-fence the point cloud and labels into the selected values of $[-75m, 75m]$ in the x- and y-axis and $[-8m, 0m]$ in the z-axis. After these methods, the image data is normalized to a mean of $[0.485, 0.456, 0.406]$ and a standard deviation of $[0.229, 0.224, 0.225]$. Next comes the augmentation method *GridMask*. This technically exists and is usable but it is not used in the experiments and thus a probability of 0.0 is set for it. After this, the points are shuffled using *PointShuffle*. The final two methods: *DefaultFormatBundle3D* and *Collect3D* formats the resulting information into a format that MMDetection3D models use and collects them for further usage.

The validation and test pipelines use the same building blocks as the training pipeline. The only difference is that no augmentations are applied for either the image or point cloud data. The only kind of change applied is geo-fencing the point cloud and normalizing the images to the values also set in the training pipeline. Test pipeline also differs from training and validation pipeline in that it doesn't load the annotations using *LoadAnnotations3D*.

4.3.2 TUMTraf Cooperative Dataset

The cooperative uses the same data pipeline as the TUMTraf Intersection Dataset version but with modified versions of many modules adapted to the now cooperative data that has to

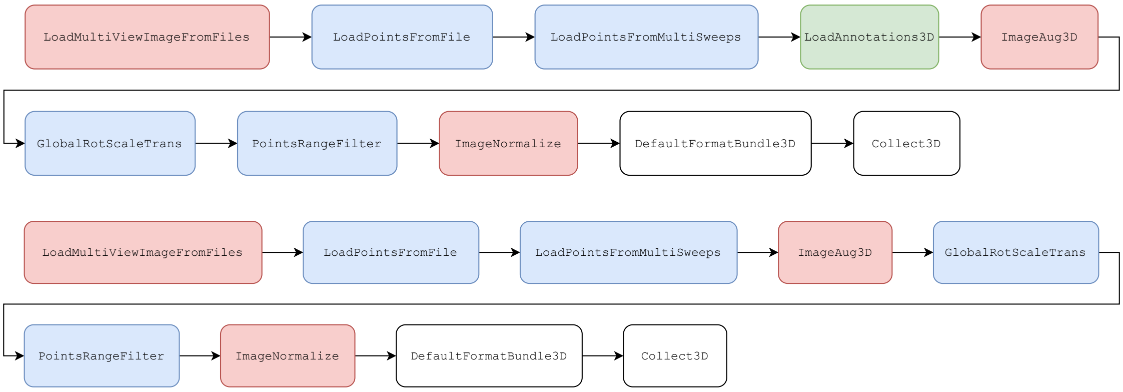


Figure 4.2: TUMTraf Intersection Dataset validation (top) and test (bottom) data pipelines. The main difference of validation to the training data pipeline is the removal of ObjectPaste, RandomFlip3D, GridMask, and the annotation filters, followed by disabling all remaining augmentations. From validation to test, LoadAnnotations3D is removed.

be processed. Generally speaking, the modifications involve adapting the methods to handle data from and for two nodes (vehicle and infrastructure) instead of one. From Figure 4.3, all of the methods with the Coop suffix in their name are the methods that were modified to handle both vehicle and infrastructure data at once. Methods without this suffix behave exactly the same as in the TUMTraf Intersection Dataset, so they will be skipped.

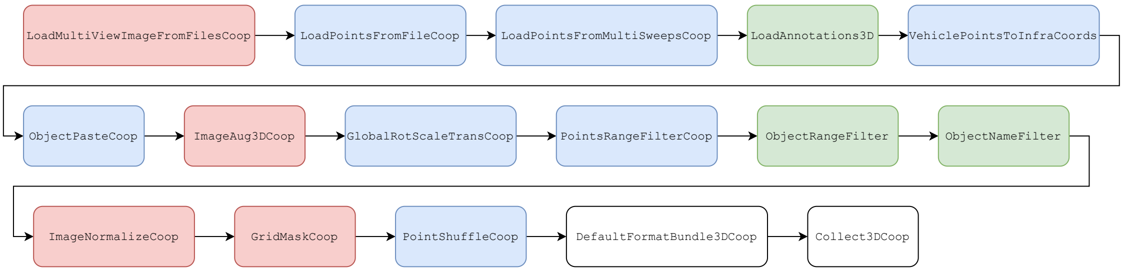


Figure 4.3: TUMTraf Cooperative Dataset training data pipeline. The red methods handle image data, the blue methods handle point cloud data, the green methods handle the annotations, and the white methods are general methods that format and collect the necessary information for the model.

The first three "coop" methods: *LoadMultiViewImageFromFilesCoop*, *LoadPointsFromFileCoop*, and *LoadPointsFromMultiSweepsCoop* behave basically the same as their non-coop counterparts. The only difference is that they now load both vehicle and infrastructure images and point clouds. *VehiclePointsToInfraCoords* transforms the vehicle LiDAR point cloud into the infrastructure coordinate system using the *vehicle2infrastructure* information from the dataset. The transformation happens here because it is assumed that in real life this transformation happens in a separate module to the deep learning model and it is assumed that point clouds taken by the model as input are already in the infrastructure coordinate system.

In *ObjectPasteCoop*, the sampling part is the same as the non-coop version. The modification is in the part where points inside the bounding boxes of the sampled objects are removed and the points of the sampled objects are pasted into the original point cloud. This now needs to be done on both vehicle points and infrastructure points. The affine augmentation methods *ImageAug3DCoop* and *GlobalRotScaleTransCoop* behave the same as in the non-coop version. Only they now apply the exact same randomly sampled transformations to both vehicle and infrastructure images and point cloud, and also save the transformation matrices separately as *vehicle_lidar_aug_matrix*, *infrastructure_lidar_aug_matrix*, *vehicle_img_aug_matrix*, and *infrastructure_img_aug_matrix*. Note that *RandomFlip3D* is not used here. This is because the module cannot be brought up to a usable state when handling both vehicle and infrastructure data at once, causing bad augmented data.

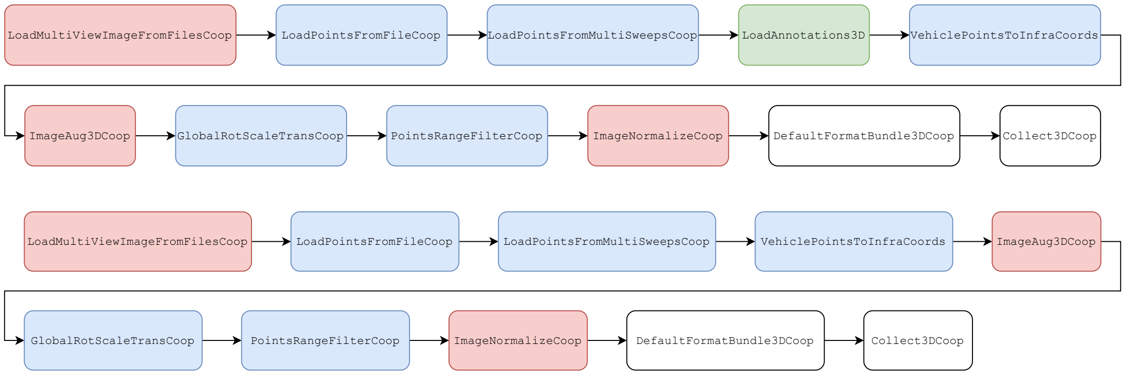


Figure 4.4: TUMTraf Cooperative Dataset validation (top) and test (bottom) data pipelines. Similar to the TUMTraf Intersection Dataset version, the main difference of validation to the training data pipeline is the removal of ObjectPaste, RandomFlip3D, GridMask, and the annotation filters, followed by disabling all remaining augmentations. From validation to test, LoadAnnotations3D is removed.

The next four "coop" methods function exactly the same as their non-coop counterpart, just applying the operations to both the vehicle and infrastructure versions of the data they work on. Finally *DefaultFormatBundle3DCoop* and *Collect3DCoop* do the exact same task as their non-coop counterpart, only now they have different and a larger number of information to format and collect. The validation and test pipelines from Figure 4.4 follow the same principle as their TUMTraf Intersection Dataset counterparts. They use the same modules as their dataset's training data pipeline but not all of them.

4.4 Infrastructure only Multi-modal 3D Object Detection Model

This model is created solely to compare the infrastructure-only version of the method of this work to InfraDet3D [Zim+23a] in the research group's toolchain. The reason is that the cooperative method requires both infrastructure and vehicle data to be available, and currently, the vehicle data is not available yet in the live system. As mentioned at the very beginning, the method of this work is a modified version of BEVFusion [Liu+23b] combined with a vehicle-infrastructure fusion method inspired by PillarGrid [Bai+22b]. Note that no piece of code from PillarGrid is used since the code of PillarGrid is not available publicly.

As such, this method is a version of BEVFusion, modified to use different point cloud and camera backbones that the proposed method uses. For point cloud data, PointPillars' PillarNet [Lan+19] is used instead of VoxelNet [ZT18]. Meanwhile, for image data, a modified version of MMYOLO's [Con22] implementation of YOLOv8 [Ult23] is used. The details of the implementations of these modifications will be covered in the next section, which covers the cooperative multi-modal 3D object detection model proposed in this work.

4.5 Cooperative Multi-modal 3D Object Detection Model

In this section, the implementation details of the method proposed in this work will be covered. This method utilizes BEVFusion as its baseline and as the method to fuse camera and LiDAR data of the node, separately on each node, then finally fuses the vehicle and infrastructure data using the method proposed by PillarGrid (element-wise max pooling). This section covers the architecture, the backbones implemented, the modifications to the camera-to-BEV transformation module that need to be done to allow cooperative operation, the camera-LiDAR fusion, the vehicle-infrastructure fusion and why it works, and finally the 3D object detection head.

4.5.1 Model Architecture

In the original BEVFusion, the top level of the model is located in *mmdet3d/models/fusion_ models/bevfusion.py*. This file is where the camera encoder (backbone, neck, and vtransform), LiDAR encoder (voxelization and backbone), the camera-LiDAR fuser, the decoder (backbone and neck), and the head are created/built. In the cooperative model, this is split into two parts since two separate instances of "headless" BEVFusion (which does camera-LiDAR fusion for the vehicle and infrastructure nodes) now need to be created, which are then followed by a vehicle-infrastructure fuser, the decoder, and the head.

The "headless" BEVFusion instance is defined in *mmdet3d/models/fusion_models/bevfusion _headless.py*. This file is similar to the original *bevfusion.py* but it defines and builds only the camera encoder (backbone, neck, and vtransform), LiDAR encoder (voxelization and backbone), and the camera-LiDAR fuser. The top level of the model is now located in *mmdet3d/models/coop_ fusion_models/bevfusion_coop.py*. In this file, the vehicle and infrastructure nodes are built, which are both instances of "headless" BEVFusion, the vehicle-infrastructure fuser, the decoder (backbone and neck), and the head.

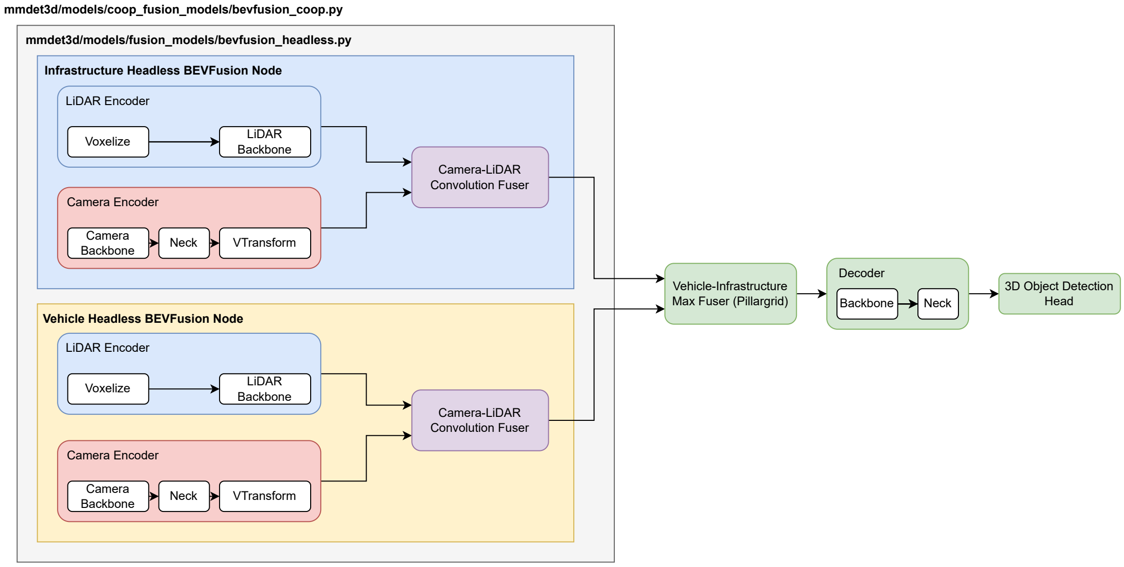

mmdet3d/models/coop_fusion_models/bevfusion_coop.py
mmdet3d/models/fusion_models/bevfusion_headless.py
Infrastructure Headless BEVFusion Node
LiDAR Encoder
Voxelize
LiDAR Backbone
Camera Encoder
Camera Backbone
Neck
VTransform
Camera-LiDAR Convolution Fuser
Vehicle Headless BEVFusion Node
LiDAR Encoder
Voxelize
LiDAR Backbone
Camera Encoder
Camera Backbone
Neck
VTransform
Camera-LiDAR Convolution Fuser
Vehicle-Infrastructure Max Fuser (Pillargrid)
Decoder
Backbone
Neck
3D Object Detection Head

Figure 4.5: A detailed architecture of the proposed deep multi-modal cooperative method. The infrastructure and vehicle nodes are represented by an instance of "headless" BEVFusion [Liu+23b] each, which fuses the camera and LiDAR features of each node. The fused vehicle and infrastructure camera-LiDAR features are then fused using a PillarGrid-style [Bai+22b] element-wise max fuser to create the final fused feature for detection.

The parameters of these vehicle and infrastructure nodes are the parameters that BEVFusion originally takes as input. For example, for the vehicle node the inputs are: *vehicle_img*, *vehicle_points*, *vehicle_lidar2camera*, *vehicle_lidar2image*, *vehicle_camera_intrinsics*, *vehicle_camera2lidar*, *vehicle_img_aug_matrix*, *vehicle_lidar_aug_matrix*, *vehicle2infrastructure*, *node*, and *metas*.

This restructuring is done so that it is technically possible to create a version of the cooperative model that consists only of a headless BEVFusion instance using configuration files for both the vehicle and infrastructure nodes to fuse their own camera and LiDAR features, and an instance of the cooperative model without headless BEVFusion instances for the theoretical RSU to run the vehicle-infrastructure fusion and 3D object detection in. However, this is currently not used since it is not possible to run live cooperative tests yet.

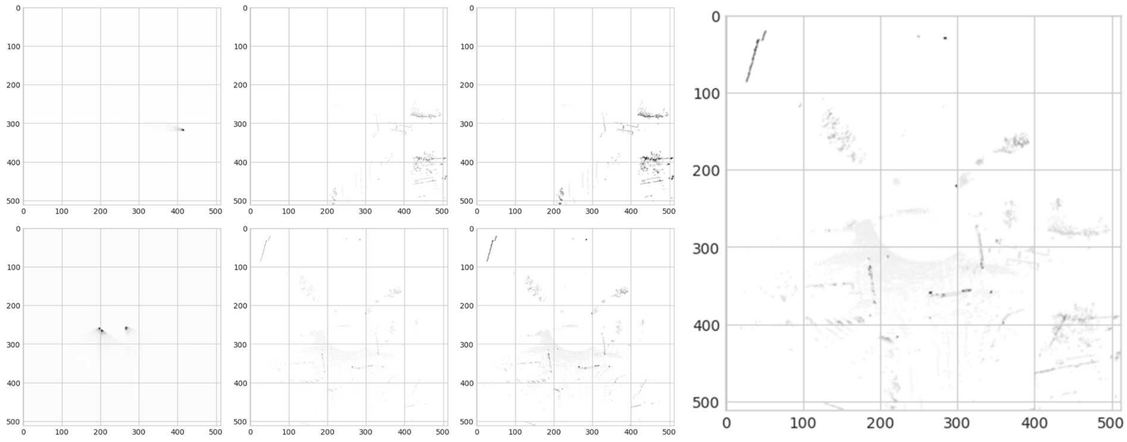


Figure 4.6: A visualization of the evaluation of the feature maps as it goes deeper into the architecture. At the top are vehicle features and at the bottom are infrastructure features. From left to right in each row are: camera features, LiDAR features, and fused camera-LiDAR features. The large visualization the furthest to the right is the final fused vehicle-infrastructure camera-LiDAR features.

On top of restructuring the method for cooperative operation, the extra camera backbone (YOLOv8), the camera vtransform to allow cooperative operation, and the vehicle-infrastructure fuser are implemented. The other parts are generally using BEVFusion's (which uses MMDetection and MMDetection3D) original implementations. The camera neck is BEVFusion's implementation of the Feature Pyramid Network (FPN) from LSS [PF20]. MMDetection's implementation of the SWin transformer [Liu+21] is used as one type of camera backbone. The LiDAR voxelization module is MMDetection3D's hard voxelization implementation. While the LiDAR backbones used are MMDetection3D's implementation of VoxelNet and PointPillars. The camera-LiDAR fuser is BEVFusion's implementation of a convolution-based fuser. The decoder backbone and neck are MMDetection3D's implementation of SECOND [YML18] and its FPN. Finally, the head is BEVFusion's implementation of TransFusion [Bai+22a] for LiDAR and camera-LiDAR 3D object detection or CenterPoint's [YZK21] head for camera-only 3D object detection.

YOLOv8 Camera Backbone

The YOLOv8 implementation that is used as the camera backbone is taken from MMYOLO's implementation of the same backbone. The modification is done in order to allow it to run on BEVFusion's environment. BEVFusion requires MMCV 1.4.0 and MMDetection 2.20.0. Meanwhile, MMYOLO requires MMCV 2.0.0rc4 to 2.1.0, MMDetection 3.0.0 to 4.0.0, and MMEngine 0.6.0 or higher. MMEngine replaces MMCV in the latest libraries from the creators of MMCV, MMDetection, and MMDetection3D: OpenMMLab.

It is clear that the environment BEVFusion runs in is considerably older than what MMYOLO requires. As a result, all the files required for YOLOv8 need to be copied to the code base and restructured. The resulting files in the code base consist of: *mmdet3d/models/backbones/ base_backbone.py*, *mmdet3d/models/backbones/mmdet_darknet .py*, and *mmdet3d/models/backbones/yolov8.py*. Then the MMEngine, MMCV, or MMDetection functions that these files use are replaced with equivalent functions that are available in MMCV 1.4.0 or MMDetection 2.20.0. For example, we replaced MMEngine's BaseModule with MMCV's BaseModule.

Finally, since the COCO pre-trained models that MMYOLO published are used for transfer learning, the different key names inside the .pth files published on MMYOLO's Github repository have to be adapted. To change the name of the keys inside the .pth file to the appropriate key names for the model architecture, the script *tools/convert_yolo_checkpoint.py* is used. After doing all of these modifications, MMYOLO's COCO pre-trained implementation of YOLOv8 can now run in the proposed method's considerably older BEVFusion-based environment.

Camera to BEV Transformation

The main modification to enable the cooperative operation of BEVFusion lies in the vehicle-side camera-to-BEV transformation, which happens in the VTransform modules of BEVFusion. As mentioned in Chapter 2, this method is a modified version of the method introduced by LSS [PF20]. The goal is to scale the image features with its discrete pixel depth probabilities and transform it into a 3D frustum point cloud. BEVFusion introduces several modifications, such as precomputing the association of each feature point to a BEV grid, improving the feature aggregation step of BEVPooling, and introducing a 2.5D depth stream to the process. This original process is used by the infrastructure side of the cooperative method. An overview of this process is shown in Figure 4.7.

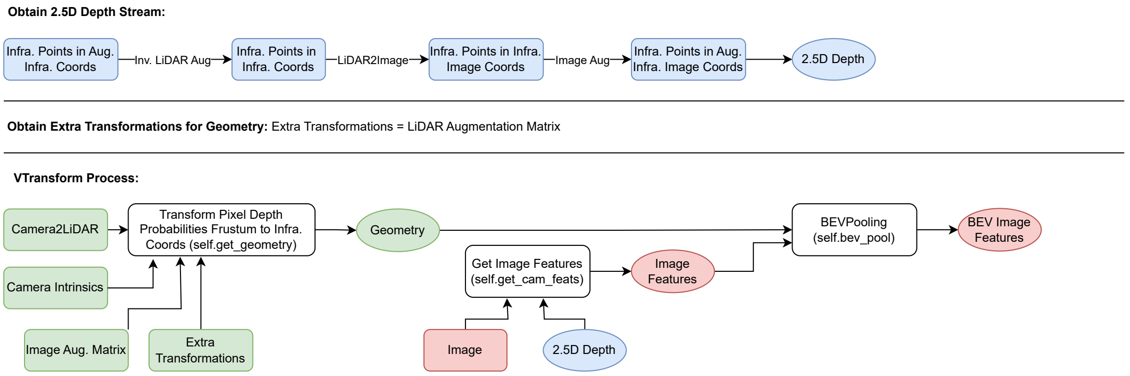

Obtain 2.5D Depth Stream:
Infra. Points in Aug. Infra. Coords
Inv. LiDAR Aug
Infra. Points in Infra. Coords
LiDAR2Image
Infra. Points in Infra. Image Coords
Image Aug
Infra. Points in Aug. Infra. Image Coords
2.5D Depth
Obtain Extra Transformations for Geometry: Extra Transformations = LiDAR Augmentation Matrix
VTransform Process:
Camera2LiDAR
Transform Pixel Depth Probabilities Frustum to Infra. Coords (self.get_geometry)
Geometry
BEVPooling (self.bev_pool)
BEV Image Features
Camera Intrinsics
Get Image Features (self.get_cam_feats)
Image Features
Image Aug. Matrix
Extra Transformations
Image
2.5D Depth

Figure 4.7: Infrastructure-side VTransform process. In this process, the images are already in the infrastructure perspective. This means the default VTransform of BEVFusion [Liu+23b] can be directly used.

The modification done is related to the transformations applied in this process. This is especially true in two parts: obtaining the 2.5D depth stream and the extra transformations for geometry. For the vehicle node, the vehicle points were transformed into the infrastructure coordinate system in the data pipeline. Meanwhile, the images are still from the vehicle's perspective. So, first, the augmentations applied to the vehicle points in the data pipeline have to be inversed, obtaining the original vehicle points in the infrastructure coordinate system. Then the inverse of *vehicle2infrastructure* is used to obtain vehicle points in the vehicle coordinate system. Then *lidar2image* is used to obtain our vehicle points in image (pixel) coordinates. Finally the image augmentations are applied to get the final coordinates that are augmented similarly to the vehicle images. Using these coordinates, the 2.5D depth stream is obtained.

Extra transformations are used in *self.get_geometry* to transform the 3D frustum of pixel depth probabilities to the augmented infrastructure coordinate system. The frustum starts in the vehicle LiDAR coordinate system. So, it needs both *vehicle2infrastructure* to transform it to the infrastructure coordinate system and the LiDAR augmentation transformation matrices to bring it to the augmented infrastructure coordinate system. This modified VTransform process is shown in Figure 4.8.

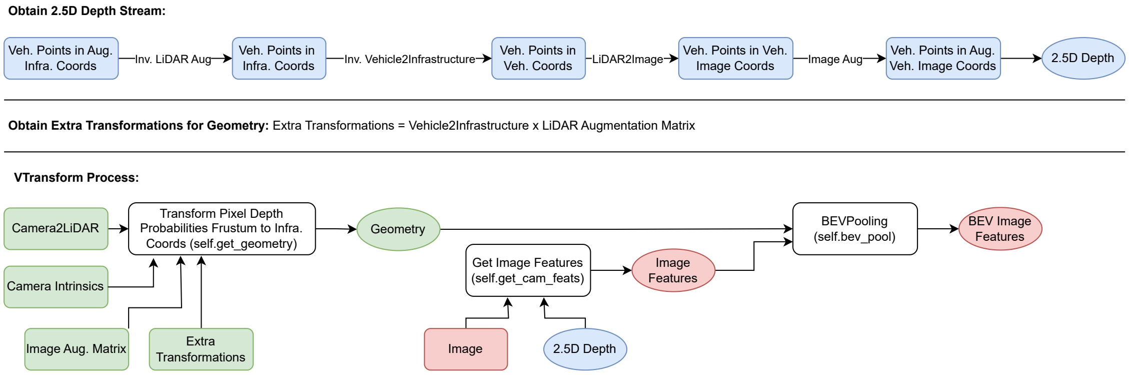

Obtain 2.5D Depth Stream:
Veh. Points in Aug. Infra. Coords
Inv. LiDAR Aug
Veh. Points in Infra. Coords
Inv. Vehicle2Infrastructure
Veh. Points in Veh. Coords
LiDAR2Image
Veh. Points in Veh. Image Coords
Image Aug
Veh. Points in Aug. Veh. Image Coords
2.5D Depth
Obtain Extra Transformations for Geometry: Extra Transformations = Vehicle2Infrastructure x LiDAR Augmentation Matrix
VTransform Process:
Camera2LiDAR
Transform Pixel Depth Probabilities Frustum to Infra. Coords (self.get_geometry)
Geometry
BEVPooling (self.bev_pool)
BEV Image Features
Camera Intrinsics
Get Image Features (self.get_cam_feats)
Image Features
Image Aug. Matrix
Extra Transformations
Image
2.5D Depth

Figure 4.8: Vehicle-side modified VTransform process. The main difference here is that the cameras are at the vehicle coordinate system, while the points are at the infrastructure coordinate system. So, the vehicle-to-infrastructure matrix and its inverse are used to transform between coordinate systems.

Vehicle-Infrastructure Fusion

The method of vehicle-infrastructure fusion is inspired by the method proposed by Pillargird [Bai+22b]. This fuser is implemented from scratch for this work and no piece of code from the PillarGrid project is used since the code is not available publicly. This is done by first stacking both infrastructure and vehicle fused camera-LiDAR features on top of each other. These features are already aligned with each other since they were both already in the infrastructure coordinate system, geo-fenced the same way, and augmented the same way before being processed by the "headless" BEVFusion. As such, element-wise max operation can be used to obtain the final fused vehicle-infrastructure feature.

4.6 Label Export

A script is implemented to perform inference but instead of using the nuScenes evaluation protocol, it exports the detection results in the ASAM OpenLABEL format. This is done to enable the usage of the evaluation protocol and the visualization scripts provided by the TUMTraf Dataset Devkit [Tea23]. This script is *tools/inference_to_openlabel_coop.py*.

This script works by first initializing the CUDA and CUDNN settings. Next, it builds the dataset, dataloader, and cooperative model. Then, it loads the selected pre-trained checkpoint into the model. Using this dataloader and model, it performs inference. The inference produces bounding boxes, scores, and class labels as outputs. Then, it transforms each bounding box into a *Detection* class object with a randomly generated uuid, its category from the label output, its x, y, z center location, its x, y, z dimensions, its yaw angle, the number of LiDAR points inside it (obtained using Open3D functions), its score, and finally its sensor ID (s110_lidar_ouster_south) is selected since it is the center of the infrastructure coordinate system. Then, the list of these Detection objects is transformed into ASAM OpenLABEL format .json files using the function *detections_to_openlabel*.

Chapter 5

Experiments

This chapter covers the experiments performed to find out the performance of the method proposed in this work. The topics covered are the test sets used and the differences between them, the metrics chosen and why they were chosen, the test system used, the limitation/caveat that applies to one of the experiment types, and finally a list of the experiments themselves and their results will be shown.

5.1 Test Sets

As mentioned in Chapter 3, two datasets were used in the experiments: the TUMTraf Intersection Dataset and the TUMTraf Cooperative Dataset created for this work. The dataset that is actually core to the proposed method is the TUMTraf Cooperative Dataset. The TUMTraf Intersection Dataset is included mainly to compare the proposed method to an existing state-of-the-art late fusion method that is currently running in the research group's toolchain: InfraDet3D [Zim+23a] and InfraDet3D was only trained and tested on this dataset and not the TUMTraf Cooperative Dataset.

As such, two different experiment types were performed. The TUMTraf Intersection Dataset experiments were performed with the test set of the dataset, which was split using random sampling and a separate test sequence. As a result, it has a class distribution imbalance between the training-validation and the test sets. For these experiments, only the classes used by the latest paper with InfraDet3D results, which is the TUMTraf Intersection Dataset paper [Zim+23b], were used. This consists of six classes: CAR, TRUCK, BUS, MOTORCYCLE, PEDESTRIAN, and BICYCLE. For these experiments, the 3D mAP values offered by the TUMTraf Dataset Devkit, which were also used by InfraDet3D, were used as the metric.

For the TUMTraf Cooperative Dataset, a different setup was used. The test set was split using stratified sampling and as a result, it has a more equal class distribution across training, validation, and test sets. In these experiments, seven classes were considered: CAR, VAN, TRUCK, BUS, TRAILER, PEDESTRIAN, and BICYCLE. The comparisons between the various configurations tested in this type of experiment used the BEV mAP numbers produced by the BEVFusion-based [Liu+23b] test script, which uses a nuScenes style evaluation protocol. On top of that, the frames per second (FPS) and VRAM numbers were also considered as well.

5.2 Metrics

As a reminder, the goal of this work is to prove the benefits of cooperative driving over vehicle-only or infrastructure-only, to prove the benefit of using both camera and LiDAR instead of only one of them, to prove deep fusion is better than late fusion, to prove that the method can run in real-time (at least 10 FPS) in cooperative camera-LiDAR mode, and to see if it uses low enough VRAM to technically be able to run on mainstream GPUs (ignoring their performance levels). This is why the metrics below were selected: mAP (BEV and 3D), frames per second (FPS) numbers, and VRAM usage.

5.2.1 Precision and Recall

Although these metrics were not actually used, since they are parts of a metric used (mean Average Precision (mAP), these two metrics are still important to understand. Based on the work of Olson and Delen [OD08], precision shows the number of true positive (TP) over all positive results (true positive (TP) + false positive(FP)). This shows the quality of positive predictions by the method. Recall is the number of true positive (TP) over all actual positive results (true positive (TP) + false negative (FN)). This shows the ability of the method to correctly detect all positive results in the data. As formulas, they can be shown as follows:

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

In the experiments, true positives (TP) are instances where the method correctly detects an object in the point cloud and/or image. False positives (FP) are instances where the method detects an object in the point cloud and/or image when there was none. Finally, false negatives (FN) are instances where the method fails to detect an object that exists in the point cloud and/or image.

Generally, in autonomous driving, high recall is preferable since it is better to be more cautious and avoid false negatives. False negatives are worse for the safety of traffic participants than false positives (although false positives are still bad).

5.2.2 Mean Average Precision

Mean Average Precision is a commonly used metric to measure performance in the field of computer vision and 3D object detection. This metric is popular because it shows the performance (accuracy) of the evaluated method while including both precision and recall into consideration [HF17].

Mean Average Precision is the mean of the Average Precision of the different object classes. For example, in the TUMTraf Intersection Dataset experiment, it is the mean over the six classes (CAR, TRUCK, BUS, MOTORCYCLE, PEDESTRIAN, and BICYCLE). The average precision of each object class is calculated by averaging the precision over different recall values. This is calculated by creating a precision-recall curve. This curve shows precision and recall values at different detection confidence threshold values (the threshold for True Positive). Average Precision is the area below this curve.

Because mean Average Precision is the mean over different object classes, a high mAP score means that the method performs well over a large variety of object classes, while maintaining a good balance between precision and recall. This is important since methods in the field of autonomous driving need to perform well across different object classes to ensure the safety of all traffic participants.

In the experiments, there are two types of mAP numbers produced by the two different evaluation scripts used. The evaluation script that is modified from BEVFusion's [Liu+23b] evaluation script (and by extension nuScenes) produces BEV mAP because it uses BEV center distance between prediction and ground truth to decide if the prediction is correct. The evaluation script from the TUMTraf Dataset Devkit [Zim+23b][Tea23] produces 3D mAP numbers because they use 3D center distance instead.

5.3 Runtime Evaluation

Frames per second (FPS) to measure real time-ness of the method in the experiments. As mentioned in Chapter 1 and papers such as [Liu+22], a minimum of 10 FPS is needed for the method to be considered in real-time. The model's FPS numbers were measured using the scripts *tools/benchmark.py* and *tools/benchmark_coop.py*. These scripts count the time needed by the model to run one full inference, including all steps, at certain intervals. The first five iterations are skipped as a warmup since the first few iterations are usually considerably slower than the average.

5.4 Resource Consumption

VRAM usage is based on the numbers shown by *nvidia-smi* (NVIDIA System Management Interface) during training and testing. During training, multiple GPUs were used and the GPUs might have different VRAM usage numbers. The highest peak number across all GPUs was chosen as the result. This ensures that the method uses the amount of VRAM listed in the tables or less. This metric is selected because this method needs to not only be able to run on very expensive deep learning specialist GPUs with lots of VRAM.

5.5 Limitation

A large limitation in the experiments concerns the newly created TUMTraf Cooperative Dataset. This dataset consists of only 500 data frames. This is very small for 3D object detection tasks in autonomous driving scenarios. As a result, the results shown in the tables in this chapter are results under heavy overfitting. The possible solutions to this issue will be presented in Chapter 7. This will also be proven in a set of infrastructure-only experiments where the different results when an extra augmentation (random flip) is used and not used are shown. This means that the numbers can't be taken at face value, but the general trends shown by the numbers can be used to prove or disprove the core idea of this work.

5.6 Test System

The system used for the experiments listed in the tables in the next section are all tested on a GPU cluster equipped with an AMD EPYC 7282 16-Core Processor, 32 GB RAM, three NVIDIA RTX 3090 with 24 GB VRAM, and 1+4 TB SSDs for data storage. For environment setup, two versions of the Docker image provided by BEVFusion were used. The original (for experiments that use SPConv), which uses Python 3.8, PyTorch 1.10.1 with CUDA 11.3, Torchvision 0.11.2, Torchaudio 0.10.1, Pillow==8.4.0, tqdm, torchpack, MMCV 1.4.0, MMDetection

2.20.0, OpenMPI 4.0.4, mpi4py 3.0.3, and the nuScenes devkit; and the modified version, which adds the latest versions of numba, torchsparse, pypcd, and Open3D.

5.7 Results

In this section, all of the experiments performed and their results are presented. The experiments were performed using two different datasets: the TUMTraf Cooperative Dataset to prove performance of the method proposed in this work in an offline cooperative scenario and the TUMTraf Intersection Dataset [Zim+23b] to compare the best overall infrastructure-only LiDAR-only and camera-LiDAR fusion configurations of the proposed method with the existing state-of-the-art in the research group's toolchain (InfraDet3D [Zim+23a]).

5.7.1 TUMTraf Cooperative Dataset

The TUMTraf Cooperative Dataset experiments consist of a small group of experiments to find the correct training schedule, a small group of experiments to test the effect of adding one extra augmentation (random flip) and confirming the overfitting assumption, all the various combinations of backbones tested to find the best-performing configuration, and finally listing the results of the best combination alongside its 3D mAP results to provide the most complete picture. In all of these experiments, a score threshold of 0.1 is used to determine if a prediction is a true positive (TP). Although a score threshold of 0.1 is very low compared to the usual 0.5-0.7, it was found during the initial feasibility tests that at score thresholds higher than this, too many false negatives (FP) are introduced. Because there are not many false positives (FP) at this value anyway, it was decided to use 0.1 as the score threshold.

Training Schedule Search

The first step in the experiments was to find the correct training schedule to train with. BEVFusion [Liu+23b] originally uses this schedule: train LiDAR only 3D object detection for 20 epochs from scratch, then fine-tune camera-LiDAR fusion using the pre-trained weights from the LiDAR only training and the weights of SwinT [Liu+21] pre-trained on the nuImages [Cae+20] dataset for 6 epochs. During LiDAR-only training, all augmentations, including *ObjectPaste* were used. Meanwhile, during camera-LiDAR finetuning, *ObjectPaste* was not used. For camera-only training, the strategy is similar to LiDAR-only training with some changes: use the nuImages pre-trained weights, and train for 20 epochs using all augmentations, except ObjectPaste.

The goal was to check if this schedule is still the best for cooperative training, meaning cooperative camera-only training lasted for 20 epochs with the pre-trained weights and all augmentations except ObjectPaste activated, for cooperative LiDAR only training lasts for 20 epochs from scratch, and in cooperative camera-LiDAR fusion the weights from cooperative LiDAR-only training were used together with the same pre-trained camera backbone weights for the vehicle and infrastructure nodes (SwinT pre-trained on nuImages and YOLOv8 pre-trained on COCO) to fine-tune on; or is it better to use an alternative schedule of taking the weights of vehicle- and infrastructure-only camera-only, LiDAR-only, and camera-LiDAR fusion models, fusing these weights, then during their respective cooperative training freezing the encoders and camera-LiDAR fuser weights and fine-tuning only the decoder and 3D object detection head weights.

Due to time limitations, it was decided to check this on camera-only vehicle, infrastructure, and cooperative training first. Only if this alternative schedule shows a benefit there, would this alternative schedule be used on LiDAR-only and camera-LiDAR fusion training. All of these camera-only training used SwinT pre-trained on nuImages, with an output size of 250, train, and grid size of 2000. They were trained for 20 epochs with a learning rate of 2e−4, cyclic momentum, and cycling learning rate configuration. The last 3 epochs were saved and the epoch with the best performance was chosen. The results of these camera-only experiments are shown in Table 5.1:

Configuration	BEV mAP	FPS	VRAM (Train)	VRAM (Test)
Vehicle camera-only	40.187	16.3	N/A	2867 MiB
Infrastructure camera-only	65.74	13.5	N/A	3675 MiB
Cooperative camera-only, alt. schedule	**68.536**	10.74	N/A	3857 MiB
Cooperative camera-only, ori. schedule	**68.912**	10.74	N/A	3857 MiB

Table 5.1: Training schedule search experiments, vehicle, infrastructure, and cooperative camera-only training

As seen in these results, the alternative schedule does not bring any benefit to cooperative training. It actually made the performance slightly worse (although still well within the margin of error). On top of this, freezing the backbone broke the MMCV learning rate scheduler hook and the gradient clipping hook. As a result, a lot of the training attempts for cooperative LiDAR-only training used the wrong learning rate schedule and often suffered gradient explosion, causing the training results to be considerably worse. This is technically solvable with enough time digging into MMCV library code but it was deemed to be not worth it considering the trade-off between time and benefit. So, for every other training in the upcoming sections, the original training schedule was used with slight changes: for camera and LiDAR training, train for 20 epochs and save the last 3 epochs. For the camera-LiDAR fusion training, fine-tune for 8 epochs and save the last 3 epochs.

Effect of Adding Random Flip and Confirming Overfitting

The next experiment performed was done because of the problem faced while trying to get random flip to work with point cloud data from the vehicle node. There was consistently some amount of misalignment when the *lidar_aug_matrix* that includes this augmentation was used to align the BEV camera features. It was decided to disable this augmentation in the experiments using the TUMTraf Cooperative Dataset. However, it was still useful to check the effect of this one augmentation on the overall performance of the method in this dataset.

Since this augmentation only works properly on the infrastructure side, two experiments were performed using infrastructure-only, LiDAR-only models that used VoxelNet with 0.075× 0.075 voxel size. Since the point cloud was geo-fenced to $[-75m, 75m]$ in the x- and y-axis, the sparse shape is 2000 × 2000. One training was done with random flip on and another with random flip off. They were trained for 20 epochs with a learning rate of 1e−4, cyclic momentum, and cycling learning rate configuration. The results were shown in Table 5.2 below:

Nodes and Sensors Used	BEV mAP	FPS	VRAM (Train)	VRAM (Test)
Infrastructure LiDAR-only w/ flip	74.3345	16.9	N/A	5771 MiB
Infrastructure LiDAR-only w/o flip	**89.156**	16.7	N/A	5771 MiB

Table 5.2: Effect of random flip and confirmation of overfitting

As shown above, adding an extra augmentation in the form of random flip bizarrely drops the mAP performance of the method by almost 15 points. This is counter-intuitive to the common understanding that adding augmentations should make the models more generalized and as a result perform better in the test set. This result confirms overfitting, the deeper explanation of which will be written in Chapter 6. With this confirmation and since this augmentation is broken for the vehicle node anyway, random flip is disabled in all other experiments shown below.

Vehicle-only, Camera-only

The first type of main experiments was for vehicle-only, camera-only models. This means that the method was set up with only one vehicle node (one instance of headless BEVFusion) that handles only the vehicle image data. Two backbone types, SwinT [Liu+21] and YOLOv8 [Ult23] were tested.

For SwinT, two setups were used: 250 output size with 2000 grid size and 375 output size with 1500 grid size. For YOLOv8, two versions of YOLOv8 were used: nano (n) and small (s), each with the same output and grid size variants as SwinT. The goal of having these output and grid size variants was to see what is more important: a larger grid size or a larger output size. They were all trained for 20 epochs with a learning rate of 2e−4, cyclic momentum, and cycling learning rate configuration. The results are shown in Table 5.3:

Backbone Configuration	BEV mAP	FPS	VRAM (Train)	VRAM (Test)
Swin T, 375 out, 1500 grid	31.75	19.58	8915 MiB	2789 MiB
Swin T, 250 out, 2000 grid	**44.84**	15.9	11479 MiB	2835 MiB
YOLOv8 n, 375 out, 1500 grid	27.51	23.66	6691 MiB	2657 MiB
YOLOv8 n, 250 out, 2000 grid	36.18	18.82	9179 MiB	2733 MiB
YOLOv8 s, 375 out, 1500 grid	30.99	23.12	6503 MiB	2703 MiB
YOLOv8 s, 375 out, 1500 grid (retrain)	28.5	23.12	6503 MiB	2703 MiB
YOLOv8 s, 250 out, 2000 grid	**42.98**	18.42	10009 MiB	2815 MiB
YOLOv8 s, 250 out, 2000 grid (retrain)	**46.83**	18.42	10009 MiB	2815 MiB

Table 5.3: Vehicle-only, camera-only experiments

As seen in the table, it is shown that YOLOv8 backbones are generally faster than SwinT, reaching around 3 FPS higher. There are two types of YOLOv8 s 250 out, 2000 grid shown in the table because the data not marked as "retrain" were achieved using a broken YOLOv8 .pth file with mismatched keys to the model. This was only discovered near the end of the working period because the overfitting masked a lot of the issue as the models reached (relatively) high-performance numbers anyway. As a result, only the data marked as "retrain" are retrained and utilize transfer learning. The unmarked ones are effectively trained from scratch.

The YOLOv8 models trained from scratch are still usable to show the fact that a larger grid size is more important than a larger output size and the fact that YOLOv8 is generally faster than SwinT. SwinT is more performant than YOLOv8 trained from scratch (44.84 vs 42.98). However, the best YOLOv8 s configuration, which utilized transfer learning from COCO is more performant than SwinT (46.83 vs 44.84). This does not mean that transfer learning always results in a better YOLOv8 s model, however, as the 375 out, 1500 grid new result is worse than the old result (28.55 vs 30.99). It is also shown by the unmarked data that YOLOv8 s managed to get a considerably higher mAP (42.98, 6.8 higher at 250 out, 2000 grid configuration) than YOLOv8 n with a negligible hit to FPS (0.4-0.5 FPS lower) at equivalent output and grid sizes. This and time-limitation are the reason why only YOLOv8 s models were retrained with the corrected .pth file as the unmarked results already showed that YOLOv8 n is simply inferior to YOLOv8 s in every way.

This result is also consistent in infrastructure-only and cooperative camera-only experiments as will be shown later. YOLOv8 m was not tested because this adds 3-3.2 GB VRAM usage on top of YOLOv8 s [Ope23] and the cooperative model needs to be able to run in the cluster's GPU during training and still be small enough to run on GPUs with less than 24 GB

VRAM during inference. This is the reason why SwinT and YOLOv8 s are chosen for training fusion and cooperative model variants.

Vehicle-only, LiDAR-only

The next experiment was for vehicle-only LiDAR-only models. Only the LiDAR point cloud from the vehicle was used and the method was set up with only one vehicle node (one instance of headless BEVFusion) that handles only the vehicle point cloud data. Two backbone types were tested, VoxelNet [ZT18] and PointPillars [Lan+19] with two hard voxelization methods: deterministic, which guarantees that the points sampled from the same point cloud in the same voxel will be the same points; and non-deterministic, which samples the points for the voxels randomly.

For VoxelNet, there were three variants: all use a sparse shape of 250 with a grid size of 2000, but one used SPConv v2 and deterministic hard voxelization, one used SPConv v2 and non-deterministic hard voxelization, and the last used a new backend for VoxelNet developed by the authors of BEVFusion: Torchsparse [Tan+22] and non-deterministic hard voxelization. Torchsparse promises a speedup of 1.7x over SPConv v2 while using less VRAM. For PointPillars a variety of output, training, and test grid sizes were tested. 512, 768, and 1024 output size with the same training grid size with two variants of test grid size: the same and 2x of train grid size (which is identifiable with the _2x suffix in later mentions of such models).

All were trained for 20 epochs with a learning rate of 1e–4, cyclic momentum, and cycling learning rate configuration, except for VoxelNet non-deterministic (and by extension VoxelNet Torchsparse) which used a cosine annealing learning rate configuration and PointPillars 768 out with 2x test grid which used a learning rate of 2e–4. The results are shown in Table 5.4:

Backbone Configuration	BEV mAP	FPS	VRAM (Train)	VRAM (Test)
VoxelNet deterministic	74.755	14.38	17027 MiB	5775 MiB
VoxelNet non-deterministic	**77.9**	15.9	17025 MiB	5775 MiB
VoxelNet Torchsparse	**77.9**	25.58	N/A	3513 MiB
PointPillars 512	**84.33**	56.82	6003 MiB	2561 MiB
PointPillars 512_2x	**85.33**	55.04	6003 MiB	2561 MiB
PointPillars 768	84.86	38.68	9503 MiB	3105 MiB
PointPillars 768_2x	84.39	38.24	9515 MiB	3105 MiB
PointPillars 1024	78.76	27.6	14815 MiB	3819 MiB
PointPillars 1024_2x	73.72	27.62	14815 MiB	3819 MiB

Table 5.4: Vehicle-only, LiDAR-only experiments

These results show that for VoxelNet, the best configurations use non-deterministic voxelization, with both the SPConv v2 and Torchsparse-backed model achieving 77.9 mAP. The difference between the two lies in their speed. The Torchsparse-backed model is considerably faster (15.9 vs 25.58 FPS) and lighter (5775 MiB vs 3513 MiB). The Torchsparse model didn't need retraining as converting a non-deterministic SPConv VoxelNet model to Torchsparse only requires the .pth model to be reformatted using a script that the BEVFusion authors provided.

Meanwhile, the PointPillars models outperform VoxelNet in every metric. The 512 and 768 output models all beat VoxelNet by around 7 mAP while crushing VoxelNet in speed, especially the 512 output models (56.82 and 55.04 vs 25.58 FPS). The PointPillars models are also all lighter than VoxelNet, except the 1024 output model. These results are replicated in the infrastructure and cooperative experiments, which will be shown later. As such, only

VoxelNet non-deterministic, VoxelNet Torchsparse, PointPillars 512 and 512_2x will be used when training fusion and cooperative model variants.

Vehicle-only, Camera-LiDAR Deep Fusion

The final vehicle-only experiment was for the camera-LiDAR fusion configuration. In this experiment, both the LiDAR point cloud and camera image from the vehicle were used and the method was set up with only one vehicle node (one instance of headless BEVFusion) that handles both data types. As mentioned in the previous subsections, the combination was limited to SwinT and YOLOv8 s for the camera backbone and VoxelNet non-deterministic, VoxelNet Torchsparse, and PointPillars 512 and 512_2x for the LiDAR backbone. The setup of the camera backbones was always matched to their LiDAR backbone: 250 output size with 2000 grid size for VoxelNet and the appropriate sizes for the two PointPillars variations.

All of them were fine-tuned for 8 epochs with a learning rate of 1e–4, cyclic momentum, and cosine annealing learning rate configuration. The results are shown in Table 5.5:

Backbone Configuration	BEV mAP	FPS	VRAM (Train)	VRAM (Test)
VoxelNet non-deterministic + SwinT	80.91	11.2	18875 MiB	6069 MiB
VoxelNet non-deterministic + YOLOv8 s	79.84	12.26	18399 MiB	5959 MiB
VoxelNet Torchsparse + SwinT	80.91	14.9	N/A	3861 MiB
VoxelNet Torchsparse + YOLOv8 s	**80.09**	17.22	N/A	3751 MiB
VoxelNet Torchsparse + YOLOv8 s (retrain)	**80.1**	17.22	15669 MiB	3751 MiB
PointPillars 512 + Swin T	84.67	17.88	13771 MiB	3949 MiB
PointPillars 512 + YOLOv8 s	**85.35**	20.36	12437 MiB	3835 MiB
PointPillars 512 + YOLOv8 s (retrain)	**85.485**	20.36	N/A	3835 MiB
PointPillars 512_2x + Swin T	84.545	17.66	13771 MiB	3949 MiB
PointPillars 512_2x + YOLOv8 s	**84.42**	20.5	12437 MiB	3835 MiB
PointPillars 512_2x + YOLOv8 s (retrain)	**84.9**	20.5	N/A	3835 MiB

Table 5.5: Vehicle-only, camera-LiDAR fusion experiments

From the results, it is shown that the PointPillars 512 variants perform better than VoxelNet. They achieve higher mAP scores and FPS numbers while being lighter during training and inference. VoxelNet is slightly helped in FPS and VRAM usage results when it uses Torchsparse instead of SPConv v2 as its backend. YOLOv8 s is generally better than SwinT since it achieves similar mAP numbers while running at 2.5-3 FPS higher than its SwinT equivalent on both VoxelNet Torchsparse and PointPillars 512 variants. These results are also replicated in both infrastructure and cooperative experiments.

Infrastructure-only, Camera-only

The first set of infrastructure-only experiments used only camera image data. The setup for these experiments was exactly the same as the vehicle-only, camera-only experiments. The same backbones and the same number of epochs, training schedule, learning rate, momentum, etc. were used. The results are shown in Table 5.6 below:

Backbone Configuration	BEV mAP	FPS	VRAM (Train)	VRAM (Test)
Swin T, 375 out, 1500 grid	61.06	15.82	16833 MiB	3509 MiB
Swin T, 250 out, 2000 grid	**67.15**	13.54	17423 MiB	3675 MiB
YOLOv8 n, 375 out, 1500 grid	52.13	20.26	11181 MiB	3329 MiB
YOLOv8 n, 250 out, 2000 grid	53.82	16.68	11713 MiB	3483 MiB
YOLOv8 s, 375 out, 1500 grid	58.14	20.02	11655 MiB	3337 MiB
YOLOv8 s, 375 out, 1500 grid (retrain)	56.505	20.02	11655 MiB	3337 MiB
YOLOv8 s, 250 out, 2000 grid	**61.0475**	16.6	12277 MiB	3459 MiB

YOLOv8 s, 250 out, 2000 grid (retrain)	**61,98**	16.6	12277 MiB	3459 MiB

Table 5.6: Infrastructure-only, camera-only experiments

These results show the reason why it is beneficial to combine both vehicle and infrastructure in cooperative driving. The dataset contains a lot of challenging scenarios such as heavy occlusion. This is why the mAP numbers listed here are considerably higher than the vehicle-only, camera-only results. The infrastructure cameras have a considerably better perspective of the scenario that is mostly free of occlusion, which allows it to gain around 15-20 mAP over the vehicle-only results in Table 5.3.

Another interesting point is the fact that these results are lower when it comes to FPS numbers and higher when it comes to VRAM usage compared to the vehicle-only results. This is because the infrastructure side consists of three images compared to the vehicle one. All things considered, the FPS results shown by SwinT and YOLOv8 s at 250 output and 2000 grid size for these metrics are still relatively good, losing only around 1.8 FPS and 2.4 FPS respectively. For VRAM usage, YOLOv8 is better with only around 2.2 GB increase in VRAM usage during training for YOLOv8 s, while SwinT has around 5.9 GB increase in VRAM usage. These results also support the decision to only consider SwinT and YOLOv8 s at 250 output and 2000 grid size for fusion and cooperative training.

Infrastructure-only, LiDAR-only

The infrastructure-only, LiDAR-only experiments had the exact same setup as their vehicle-only equivalents. The same backbones and the same number of epochs, training schedule, learning rate, momentum, etc. were used. The results are shown in Table 5.7 below:

Backbone Configuration	BEV mAP	FPS	VRAM (Train)	VRAM (Test)
VoxelNet deterministic	91.56	14.6	16313 MiB	5771 MiB
VoxelNet non-deterministic	**89.72**	16.58	15311 MiB	5771 MiB
VoxelNet Torchsparse	**89.72**	25.78	N/A	3551 MiB
PointPillars 512	**90.91**	57.28	6003 MiB	2793 MiB
PointPillars 512_2x	**92.86**	57.76	6005 MiB	2793 MiB
PointPillars 768	91.123	39.6	9503 MiB	3393 MiB
PointPillars 768_2x	92.07	39.58	9501 MiB	3393 MiB
PointPillars 1024	90.965	27.7	14869 MiB	4177 MiB
PointPillars 1024_2x	92.02	27.44	14913 MiB	4177 MiB

Table 5.7: Infrastructure-only, LiDAR-only experiments

These results also show that infrastructure has a better perspective of the scenario. The mAP results are all better than their vehicle-only equivalents shown in Table 5.4. Their FPS and VRAM results are also generally the same (within the margin of error) as their vehicle-only equivalents. One interesting thing shown by these results is that VoxelNet non-deterministic and Torchsparse are worse than VoxelNet deterministic for the infrastructure-only scenario.

The performance of the different PointPillars models is also much more similar to each other compared to the vehicle-only equivalents. However, they still show a diminishing return when increasing PointPillar's output size beyond 512. These results also support the decision to only consider VoxelNet non-deterministic, VoxelNet Torchsparse, PointPillars 512, and PointPillars 512_2x in fusion and cooperative training.

Infrastructure-only, Camera-LiDAR Deep Fusion

The final set of infrastructure-only experiments fused both camera image and LiDAR point cloud data. The setup for these experiments was exactly the same as their vehicle-only equivalents. The same backbones and the same number of epochs, training schedule, learning rate, momentum, etc. were used. The results are shown in Table 5.8 below:

Backbone Configuration	BEV mAP	FPS	VRAM (Train)	VRAM (Test)
VoxelNet non-deterministic + SwinT	90.26	10	23227 MiB	6551 MiB
VoxelNet non-deterministic + YOLOv8 s	91.55	11.42	19607 MiB	6329 MiB
VoxelNet Torchsparse + SwinT	90.26	12.68	N/A	4477 MiB
VoxelNet Torchsparse + YOLOv8 s	**91.56**	15.22	N/A	4255 MiB
VoxelNet Torchsparse + YOLOv8 s (retrain)	**90.42**	15.22	16433 MiB	4255 MiB
PointPillars 512 + Swin T	90.97	14.1	16687 MiB	4819 MiB
PointPillars 512 + YOLOv8 s	**90.71**	17.58	13229 MiB	4585 MiB
PointPillars 512 + YOLOv8 s (retrain)	**91**	17.58	13229 MiB	4585 MiB
PointPillars 512_2x + Swin T	92.765	14.16	16689 MiB	4819 MiB
PointPillars 512_2x + YOLOv8 s	**93.025**	17.28	13229 MiB	4585 MiB
PointPillars 512_2x + YOLOv8 s (retrain)	**92.92**	17.28	13229 MiB	4585 MiB

Table 5.8: Infrastructure-only, camera-LiDAR fusion experiments

These results again show that infrastructure has a better perspective of the scenario. The mAP results are all better than their vehicle-only equivalents shown in Table 5.5. Their FPS and VRAM results are also generally the same (within the margin of error) as their vehicle-only equivalents, except for the VoxelNet and SwinT combo. This combination takes a large hit to VRAM usage, increasing it by around 4.3 GB. During training, it already uses 23227 MiB, which almost maxes out the VRAM of the cluster's RTX 3090, and this is not even the cooperative version of the model with two nodes.

Other than this, these results also show that PointPillars 512 variants are still better than VoxelNet in all metrics, YOLOv8 is still generally better than SwinT in all metrics, and TorchSparse still improves VoxelNet-based models when it comes to FPS and VRAM usage. An interesting insight here is that retraining the models that use YOLOv8 s to actually utilize transfer learning from COCO actually reduces the performance slightly unlike in the vehicle-only experiments. This is likely caused by the fact that COCO and TUMTraf have very different Operational Design Domains (ODDs).

Cooperative, Camera-only

These were the first set of experiments, in which both the vehicle and infrastructure nodes of the method were built and active. For these experiments, only camera image data from the vehicle and infrastructure sensors were used. The setup for these experiments when it comes to the backbones used, number of epochs, training schedule, learning rate, momentum, etc. were exactly the same as their vehicle-only and infrastructure-only equivalents. The results are shown in Table 5.9 below:

Backbone Configuration	BEV mAP	FPS	VRAM (Train)	VRAM (Test)
Swin T, 375 out, 1500 grid	59.94	12.42	19721 MiB	3693 MiB
Swin T, 250 out, 2000 grid	**68.58**	10.76	22553 MiB	3857 MiB
YOLOv8 n, 375 out, 1500 grid	46.9	16.9	13263 MiB	3365 MiB
YOLOv8 n, 250 out, 2000 grid	61.14	13.82	15169 MiB	3485 MiB
YOLOv8 s, 375 out, 1500 grid	61.055	16.92	16343 MiB	3419 MiB
YOLOv8 s, 375 out, 1500 grid (retrain)	59.24	16.92	16343 MiB	3419 MiB
YOLOv8 s, 250 out, 2000 grid	**66.48**	13.9	15363 MiB	3541 MiB
YOLOv8 s, 250 out, 2000 grid (retrain)	**68,49**	13.9	15363 MiB	3541 MiB

Table 5.9: Cooperative, camera-only experiments

From these results again a repeat of the trends shown in the vehicle-only and infrastructure-only experiments is shown. The results of YOLOv8 s at 250 output and 2000 grid size for these metrics are still the best among YOLOv8 models. They have the highest mAP, an FPS number that is still comfortably above real-time (10 FPS), and use low enough VRAM during training that it is feasible to expect cooperative fusion models that use it to be trainable using the RTX 3090 without gradient accumulation.

Meanwhile, it looks slightly worse for SwinT. At an output size of 250 and grid size of 2000, it is still the best SwinT configuration when it comes to mAP, but the FPS number is now just slightly above real-time at 10.76 FPS. VRAM usage is also getting close to maxing out the VRAM of the RTX 3090. This indicates that some cooperative fusion models that use it might only be trainable with gradient accumulation.

The cooperative models are also shown to have better mAP than their vehicle-only and infrastructure-only equivalents shown in Tables 5.3 and 5.6, except in the case of SwinT and YOLOv8 n at 375 output size and 1500 grid size. At these sizes, the mAP actually dropped by 1.12 for SwinT and 5.23 for YOLOv8 n compared to their infrastructure-only equivalents, which is quite significant.

Cooperative, LiDAR-only

In this set of experiments, both the vehicle and infrastructure nodes of the method were built and working. Both the LiDAR point cloud data from vehicle and infrastructure were used. The setup for these experiments when it comes to the backbones used, number of epochs, training schedule, learning rate, momentum, etc. were exactly the same as their vehicle-only and infrastructure-only equivalents. The results are shown in Table 5.10 below:

Backbone Configuration	**BEV mAP**	**FPS**	**VRAM (Train)**	**VRAM (Test)**
VoxelNet deterministic	94.7	8.88	17097 MiB	5641 MiB
VoxelNet non-deterministic	**93.31**	10.54	17095 MiB	5641 MiB
VoxelNet Torchsparse	**93.31**	20.58	N/A	3639 MiB
PointPillars 512	**94.47**	48.12	7147 MiB	2857 MiB
PointPillars 512_2x	**93.925**	47.26	7147 MiB	2857 MiB
PointPillars 768	94.32	34.26	11389 MiB	3537 MiB
PointPillars 768_2x	91.4	34.26	11411 MiB	3537 MiB
PointPillars 1024	94.85	24.26	17013 MiB	4433 MiB
PointPillars 1024_2x	92,38	24.16	17013 MiB	4433 MiB

Table 5.10: Cooperative, LiDAR-only experiments

The results show that the cooperative LiDAR-only models are generally better than their vehicle- and infrastructure-only equivalents by up to almost 4 mAP. The trends when it comes to the backbones from the vehicle- and infrastructure-only experiments also apply here. Deterministic VoxelNet is slightly better than non-deterministic and Torchsparse, but considerably slower with the cooperative deterministic VoxelNet model already running below the 10 FPS target. Meanwhile, VoxelNet non-deterministic is barely above this target at 10.54 and Torchsparse is comfortably over this target at 20.58.

For PointPillars models, the diminishing returns of increasing output size over 512 also apply here. But, all PointPillars models are still faster than even the fastest VoxelNet variant (Torchsparse) and comfortably faster than the 10 FPS target. These results again support the decision to only consider VoxelNet non-deterministic, VoxelNet Torchsparse, PointPillars 512, and PointPillars 512_2x for the most difficult experiment: cooperative, camera-LiDAR fusion.

Cooperative, Camera-LiDAR Deep Fusion

This is the most challenging set of experiments performed. In these experiments, both the vehicle and infrastructure nodes of the method were built and working and both the camera images and LiDAR point clouds of both the vehicle and infrastructure sensors were used. The setup for these experiments when it comes to the backbones used, number of epochs, training schedule, learning rate, momentum, etc. were exactly the same as their vehicle-only and infrastructure-only equivalents. The results are shown in Table 5.11:

Backbone Configuration	BEV mAP	FPS	VRAM (Train)	VRAM (Test)
VoxelNet non-deterministic + SwinT	93.47	6.3	15169 MiB*	6689 MiB
VoxelNet non-deterministic + YOLOv8 s	92.94	7.24	22025 MiB	6385 MiB
VoxelNet Torchsparse + SwinT	93.51	8.84	N/A	4613 MiB
VoxelNet Torchsparse + YOLOv8 s	**92.94**	10.66	N/A	4277 MiB
VoxelNet Torchsparse + YOLOv8 s (retrain)	**94.305**	10.66	18651 MiB	4277 MiB
PointPillars 512 + Swin T	94.43	9	22033 MiB	4935 MiB
PointPillars 512 + YOLOv8 s	**94.27**	11.14	17307 MiB	4631 MiB
PointPillars 512 + YOLOv8 s (retrain)	**94.25**	11.14	17307 MiB	4631 MiB
PointPillars 512_2x + Swin T	92.79	9.06	22025 MiB	4935 MiB
PointPillars 512_2x + YOLOv8 s	**94.16**	11.2	17013 MiB	4631 MiB
PointPillars 512_2x + YOLOv8 s (retrain)	**94.22**	11.2	17013 MiB	4631 MiB

Table 5.11: Cooperative, camera-LiDAR fusion experiments

The results show that cooperative, camera-LiDAR fusion models are generally always better than their vehicle- and infrastructure-only equivalents as shown in the Tables 5.5 and 5.8. However, they are very close when it comes to mAP results to cooperative LiDAR-only models. The cooperative, fusion variants that use PointPillars 512 are all worse than their cooperative, LiDAR-only equivalents. While the cooperative, fusion PointPillars 512_2x variants are slightly better than their cooperative, LiDAR-only equivalents. For VoxelNet, only the VoxelNet Torchsparse cooperative, fusion variants are better than their cooperative, LiDAR-only variants.

When it comes to real-time operations, only three configurations meet the requirement of running at 10 FPS or more. VoxelNet Torchsparse + YOLOv8 s managed 10.66 FPS, PointPillars 512 + YOLOv8 s managed 11.14 FPS, and finally, PointPillars 512_2x + YOLOv8 s managed 11.2 FPS. All combinations listed on the table were light enough to run on mainstream GPUs with only 8 GB VRAM. All of them except one were also light enough to be able to be trained on the RTX 3090 without using gradient accumulation. Only VoxelNet non-deterministic + SwinT was too large to train without gradient accumulation. This is the reason why even when this combination is always the heaviest in previous experiments, in this list it only uses 15169 MiB VRAM during training.

Best Result

Based on the results of vehicle-only LiDAR-only plus camera-LiDAR fusion, infrastructure-only LiDAR-only plus camera-LiDAR fusion, and cooperative LiDAR-only plus camera-LiDAR fusion results, the combination PointPillars 51_2x and YOLOv8 s after retraining is chosen as the best. This is because it beats PointPillars 512 and YOLOv8 s in vehicle-only LiDAR-only by 1 mAP, infrastructure-only LiDAR-only by 1.9 mAP, infrastructure-only camera-LiDAR fusion by 1.92 mAP. It lost to PointPillars 512 + YOLOv8 s in vehicle-only camera-LiDAR fusion by 0.585 mAP, cooperative LiDAR-only by 0.555 mAP, and cooperative camera-LiDAR fusion by 0.03 mAP.

This shows that PointPillars 51_2x and YOLOv8 s after retraining win by comparatively much larger margins than it loses by. Most importantly, it is the second-best infrastructure-only camera-LiDAR fusion combination, losing only to itself before retraining by 0.105 mAP. This is important for the TUMTraf Intersection Dataset experiments. Combining mAP results with FPS and VRAM usage results in this combination providing the best trade-off across all metrics. The 3D mAP results are obtained while setting a minimum of 5 3D points in a detection to be considered and using either south 1 or south 2 camera field of view, while BEV mAP doesn't do any of this. The results of PointPillars 51_2x and YOLOv8 are shown in Tables 5.12 and 5.13:

		3D mAP (South 1 FOV)			
Configuration	**BEV mAP**	**Easy**	**Mod.**	**Hard**	**Overall**
Vehicle-only, camera-only	46.83	39.31	12.42	4.29	35.02
Vehicle-only, LiDAR-only	85.33	77.30	31.26	53.76	76.68
Vehicle-only, camera-LiDAR fusion	84.9	77.29	34.29	39.71	76.19
Infrastructure-only, camera-only	61,98	41.13	15.64	1.35	37.09
Infrastructure-only, LiDAR-only	92.86	82.16	45.14	46.56	81.07
Infrastructure-only, camera-LiDAR fusion	92.92	**85.43**	49.10	49.56	84.13
Cooperative, camera-only	68.94	52.04	29.26	10.28	49.81
Cooperative, LiDAR-only	93.925	84.61	50.00	53.78	**84.15**
Cooperative, camera-LiDAR fusion	**94.22**	84.50	**51.67**	**55.14**	84.05

Table 5.12: Summary of the best configuration with its 3D mAP results from south 1 FOV

		3D mAP (South 2 FOV)			
Configuration	**BEV mAP**	**Easy**	**Mod.**	**Hard**	**Overall**
Vehicle-only, camera-only	46.83	31.47	37.82	30.77	30.36
Vehicle-only, LiDAR-only	85.33	85.22	76.86	69.04	80.11
Vehicle-only, camera-LiDAR fusion	84.9	77.60	72.08	73,12	76.40
Infrastructure-only, camera-only	61,98	31.19	46.73	40.42	35.04
Infrastructure-only, LiDAR-only	92.86	86.17	88.07	75.73	84.88
Infrastructure-only, camera-LiDAR fusion	92.92	87.99	**89.09**	**81.69**	87.01
Cooperative, camera-only	68.94	45.41	42.76	57.83	45.74
Cooperative, LiDAR-only	93.925	92.63	78.06	73.95	85.86
Cooperative, camera-LiDAR fusion	**94.22**	**93.42**	88.17	79.94	**90.76**

Table 5.13: Summary of the best configuration with its 3D mAP results from south 2 FOV

5.7.2 TUMTraf Intersection Dataset

The TUMTraf Intersection Dataset experiments were used to create a final comparison between the best backbone configuration on the TUMTraf Cooperative Dataset against InfraDet3D for camera-LiDAR fusion. As mentioned in the beginning, the best overall camera-LiDAR fusion configuration was chosen as the representative of the proposed method. This configuration is PointPillars 512_2x and YOLOv8 s after retraining to utilize transfer learning from the COCO pre-trained YOLOv8 weights.

Two fields of view were considered: camera south 1 and south 2. In each perspective, two cases were evaluated: LiDAR-only and camera-LiDAR fusion. The values for InfraDet3D shown here are taken from the TUMTraf Intersection Dataset paper [Zim+23b], which provides the most up-to-date performance results of InfraDet3D. The results of the proposed method were obtained using the default settings of the evaluation script from TUMTraf Dataset Devkit and a minimum of 5 3D points in a detection. The results are shown in Table 5.14 below:

	3D mAP			
Configuration	**Easy**	**Mod.**	**Hard**	**Overall**
LiDAR only (InfraDet3D, south 1 FOV)	75.81	47.66	**42.16**	55.21
LiDAR only (proposed method, south 1 FOV)	**76.24**	**48.23**	35.19	**69.47**
LiDAR only (InfraDet3D, south 2 FOV)	38.92	46.60	**43.86**	43.13
LiDAR only (proposed method, south 2 FOV)	**74.97**	**55.55**	39.96	**69.94**
Camera-LiDAR fusion (InfraDet3D, south 1 FOV)	67.08	31.38	35.17	44.55
Camera-LiDAR fusion (proposed method, south 1 FOV)	**75.68**	**45.63**	**45.63**	**66.75**
Camera-LiDAR fusion (InfraDet3D, south 2 FOV)	58.38	19.73	33.08	37.06
Camera-LiDAR fusion (proposed method, south 2 FOV)	**74.73**	**53.46**	**41.96**	**66.89**

Table 5.14: Comparison of the proposed method to InfraDet3D

From the table, we can see that the proposed method is better than InfraDet3D both when LiDAR only and camera-LiDAR fusion was used in every metric, except in the case of LiDAR only, hard (50-64 m) detections. Since the proposed method is still capable of running in real-time (17.28 FPS), this proves that deep fusion is better a better fusion approach than late fusion as proposed in the introduction of the work.

Chapter 6

Analysis

This chapter provides an analysis of the data listed in Chapter 5. This analysis is divided into quantitative and qualitative analysis. Quantitative analysis covers both types of experiments and their results. Here it will be decided whether some of the goals of the thesis are achieved or not, and several other conclusions could be reached using this data. Qualitative analysis is analyzed using visualizations of the detection results of the proposed method and contains several conclusions that cannot be proven by raw data alone.

6.1 Quantitative Analysis

This section contains an analysis of all the data listed in the experiment results of Chapter 5. This covers both types of datasets used, proving overfitting, the effects of transfer learning, and the diminishing returns caused by decreasing the size of the voxel.

6.1.1 TUMTraf Cooperative Dataset

The goal of this analysis is to prove that cooperative driving is better than infrastructure-only and that is in turn better than vehicle-only. This should also prove that fusing the camera and LiDAR is better than using LiDAR only, which in turn should be better than using the camera only. On top of this, there should be configurations that can run in real-time on mainstream GPUs.

To prove the first and second goal, the data in Tables 5.12 and 5.12; and Tables 5.3 to 5.11 can be used. Looking at Table **??**, it is clearly visible, especially from the 3D mAP results, that the lowest three results are the three camera-only models, with vehicle-only (35.02 and 30.36 mAP) being the lowest and cooperative the highest (49.81 and 45.74 mAP) of the three. They are then followed by the three LiDAR-only models in the same order vehicle (76.68 and 80.11 mAP), infrastructure (81.07 and 84.88 mAP), and cooperative (84.15 and 85.86 mAP). Finally, the best are the camera-LiDAR fusion models, with the infrastructure-only (84.13 and 87.01 mAP) following the cooperative model (84.05 and 90.76 mAP).

The only exception to this rule is that the vehicle-only camera-LiDAR fusion model (76.19 and 76.40 mAP) is lower than the three LiDAR-only models. This is likely caused by the poor perspective and the really bad performance of the camera-only variant of the vehicle-only models. This could be caused by the fact that the vehicle uses only one camera, with a bad perspective, while infrastructure uses three cameras with a much better perspective. Regardless, these results mean that the first and second goals are achieved.

To prove that the proposed cooperative, camera-LiDAR fusion method can run in real-time and can (technically) run on mainstream GPUs (ignoring the fact that cheaper GPUs have worse performance than the RTX 3090 used in the experiments), we look at the frames per second (FPS) numbers and VRAM usage shown in Tables 5.3 to 5.11. These results clearly show that there are three different variations of cooperative camera-LiDAR fusion models that are capable of running in or more than real-time (10 FPS). They are as follows: VoxelNet Torchsparse + YOLOv8 s (10.66 FPS), PointPillars 512 + YOLOv8 s (11.14 FPS), and PointPillars 512_2x + YOLOv8 s (11.2 FPS). Three other cooperative camera-LiDAR fusion methods are close to real-time: VoxelNet Torchsparse + SwinT (8.84 FPS), PointPillars 512 + SwinT (9 FPS), and PointPillars 512_2x + SWinT (9.06 FPS). All of these results are achieved without using acceleration and optimization with TensorRT. Noncooperative and noncamera-LiDAR fusion methods are all lighter than cooperative, camera-LiDAR fusion methods, and so they all run comfortably above 10 FPS. These results mean that the goal of running in real-time is achieved.

Finally, based on the same tables, it is clearly shown that the heaviest configuration during inference is VoxelNet non-deterministic + SwinT at 6689 MiB. This is still comfortably lower than the VRAM of mainstream GPUs, such as the NVIDIA RTX 3070, which has 8 GB of VRAM. This configuration is also the only one that exceeds the 24 GB VRAM of the RTX 3090 during training, which requires the use of gradient accumulation to be trainable. Every other configuration uses less than 24 GB during training, with the highest recorded non-gradient accumulated configuration being infrastructure-only VoxelNet non-deterministic + SwinT at 23227 MiB. This means that the final goal for the experiments using the TUMTraf Cooperative Dataset is also achieved.

6.1.2 TUMTraf Intersection Dataset

The goal of the experiments using the TUMTraf Intersection Dataset is simple: achieve a better performance (3D mAP) on the test set of this dataset than the current state-of-the-art infrastructure-only, camera-LiDAR fusion used in the research group's toolchain: InfraDet3D [Zim+23a]. Based on the results in Table 5.14, this goal was comfortably achieved.

In both LiDAR-only and camera-LiDAR fusion modes, in both the south 1 and south 2 fields of view, the chosen best configuration of the proposed method comfortably beats InfraDet3D in all metrics, except in hard (50-64 m) 3D mAP of LiDAR only detection in both the south 1 and south 2 fields of view. This is likely caused by the fact that objects in this category are often very small and/or occluded. However, as mentioned earlier, the proposed method still handily beats the LiDAR-only easy (0-40 m), moderate (40-50 m), and overall 3D mAP results. As such, the model achieves the goal for the TUMTraf Intersection Dataset experiments.

6.1.3 Overfitting

One of the experiments performed on the TUMTraf Cooperative Dataset aims to prove the assertion that the proposed method suffers heavy overfitting due to the very small size of the dataset. This was proven using the data in Table 5.2. When random flip augmentation was used, the mAP of the method counterintuitively dropped instead of increasing. This goes against the conventional wisdom that adding augmentations should be able to create enough "new" data to allow the model to generalize even on small datasets.

This weird result seems to be caused because of a heavy overfitting situation. As such, even with augmentation, the model still could "memorize" the data and as a result, it overfits to the augmented data during training. The more augmentations are applied, the more different the data during training "looks" compared to its original, unaugmented form, which is used during testing. As a result, performance drops because it is now overfitted to these different-looking data and, as such, less capable of working on raw, unaugmented data.

The good results in the TUMTraf Intersection Dataset experiments also prove that the proposed method works properly and is able to learn the correct features needed to perform well in the 3D object detection task when it is provided with enough data to prevent overfitting. This shows that the overfitting faced in the TUMTraf Cooperative Dataset experiments was purely caused by a lack of data and not because the proposed method is unable to correctly learn the features needed for the task at hand.

6.1.4 Effect of Transfer Learning

Another interesting observation from the quantitative data shown in Chapter 5 is that retraining the models that use YOLOv8 s to actually utilize transfer learning from COCO sometimes actually reduces the performance. These outlier results with a drop of 1 mAP or more can be observed in all camera-only configurations using YOLOv8 s at 375 output and 1500 grid size and infrastructure-only VoxelNet Torchsparse + YOLOv8. However, based on all other results, it is generally proven that utilizing transfer learning from models pre-trained on other datasets is useful, even if the dataset is not necessarily autonomous driving-related (YOLOv8 was pre-trained on the COCO dataset).

6.1.5 Diminishing Returns of Decreasing Voxel Size

The final observation from the quantitative data in Chapter 5 is the fact that increasing output size beyond 512 (and as a result decreasing voxel size) is not beneficial. This can be obtained from the results of LiDAR-only experiments, where PointPillars with output sizes of 512 (voxel size of $[0.293, 0.293, 8]$), 768 (voxel size of $[0.1953, 0.1953, 8]$), and 1024 (voxel size of $[0.1465, 0.1465, 8]$) were tested.

The results from Table 5.7 and 5.10 showed that increasing output size (decreasing voxel size) is generally pointless as the performance of the different PointPillars configurations stays almost the same when increasing output size, with the best result achieved by PointPillars 512_2x. It is also sometimes harmful to performance (as shown in the vehicle-only results in Table 5.4), where the mAP drops as the output size increases. These observations also come while the FPS numbers consistently go down and VRAM usage goes up as the output size grows larger. The most extreme result was shown in the cooperative, LiDAR-only results where the PointPillars with 1024 output size used almost 10 GiB more VRAM than PointPillars than PointPillars with 512 output size while running at 23-24 FPS slower.

6.2 Qualitative Analysis

This section consists of an analysis of several characteristics of the proposed methods and particularities of the experiments performed using the visualizations of the detections from those experiments. The characteristics covered are the ability of the proposed method to overcome (heavy) occlusion, the performance with far away objects, how camera-LiDAR fusion helps, the performance of the proposed method in night scenes, and noting an issue that was discovered at the end regarding missing labels in one drive of the TUMTraf Cooperative Dataset.

6.2.1 Occlusion

The main premise of this proposed method is to help autonomous vehicles in (heavily) occluded scenarios by including infrastructure sensors, that have a much better view of the scenario than the vehicle. To prove this, two sets of images are chosen: the first (left image) in each set is the detection of a vehicle only, PointPillars 512_2x + YOLOv8 s fusion model post retraining and the second (right image) in each set is the detection of the cooperative version of the same model.

In the first set, there is a scenario where the ego-vehicle is behind three large vehicles, occluding almost everything in front of it. This is a heavy occlusion scenario, and the vehicle can see almost nothing in front of the three large vehicles using either camera or LiDAR. The images in Figure 6.1 show the detection results of vehicle-only and cooperative methods from the ego vehicle's point of view.

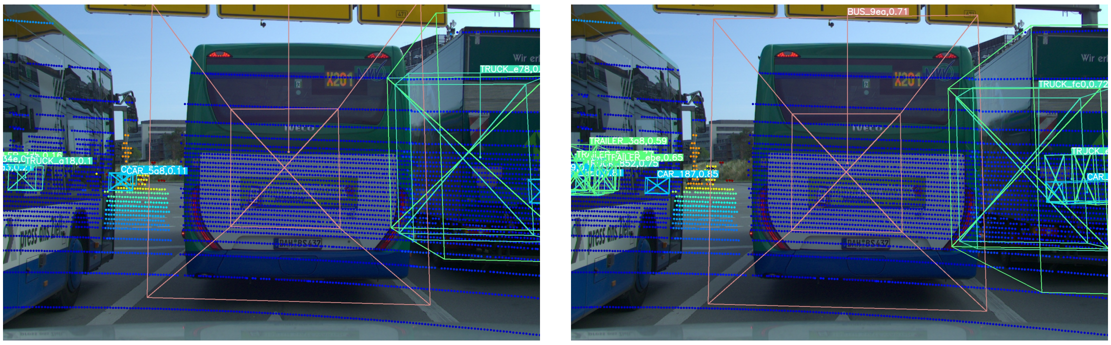


Figure 6.1: Occlusion set 1, vehicle POV: vehicle-only vs. cooperative. The detections beyond the three large vehicles shown in the vehicle-only case are all misplaced and inaccurate as shown in Figure 6.2. This is likely only possible due to the overfitting causing the model to guess/estimate object class and locations even when there are no points. Meanwhile, the low score threshold of 0.1 allows the objects to become false positives.

At a quick glance, it might seem that while the vehicle-only method has fewer detections in front of the three large vehicles, it still has some. However, the BEV perspective visualizations using the point clouds of vehicle-only and cooperative in Figure 6.2 tell a different story.

In these BEV images, white points show the vehicle point cloud and red points show the infrastructure point cloud. From here we can see that the "detections" beyond the three large vehicles detected by the vehicle-only model are merely guesses/estimations on areas without any points or camera image data and a lot of them are wrong. There are even missing objects, such as the vehicles in the middle-left area, the bottom-right area, and the car in the far right and top-right areas. The pedestrians "detected" by the vehicle-only model are also a lot more unstable, with multiple false positives around each, while the cooperative model is more stable and with fewer false positives (although there are still some. These false "detections" by the vehicle-only model are only possible because the model is currently overfitting very heavily and the visualization was done without filtering by a minimum number of points in a detection.

The second set contains a scenario where the ego-vehicle is waiting at a traffic light with (previously) a clear view of vehicles in the opposing direction. Then, a truck passed in front of those vehicles, occluding the view. The images shown in Figures 6.3 show the detection results of vehicle-only and cooperative methods from the ego vehicle's point of view.

Here, again, the vehicle-only results seem fine at a quick glance. It even managed to detect a car behind the right side of the trailer that is supposedly not detected by the cooperative model. However, again, two different perspectives of the same scene tell a different story.

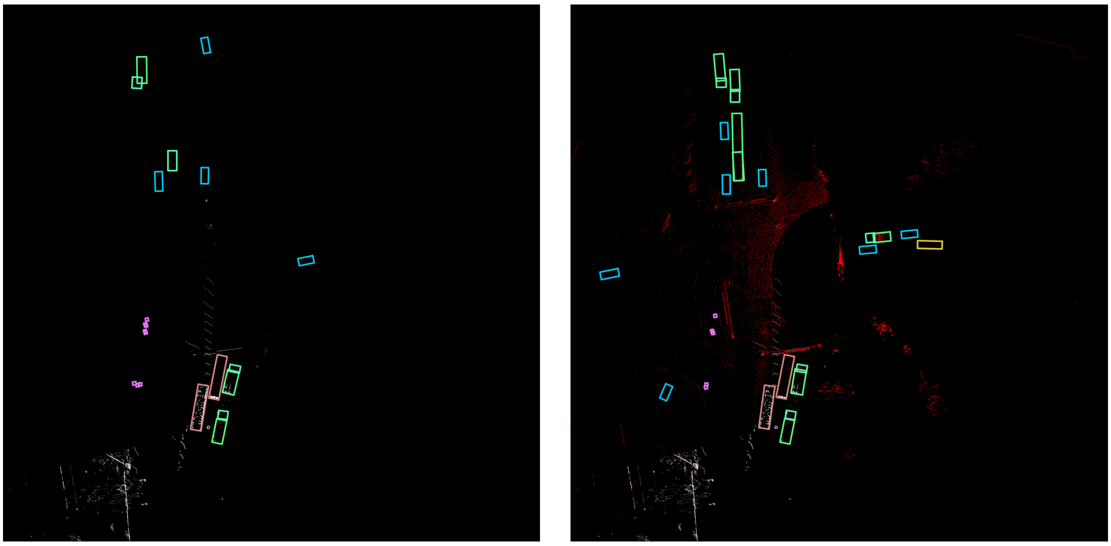


Figure 6.2: Occlusion set 1, BEV point cloud POV: vehicle-only vs. cooperative. From this BeV perspective, it is clear that the vehicle-only model only very inaccurately guessed/estimated the locations of the objects behind the three large vehicles. This is likely caused by the overfitting and low score threshold of 0.1.

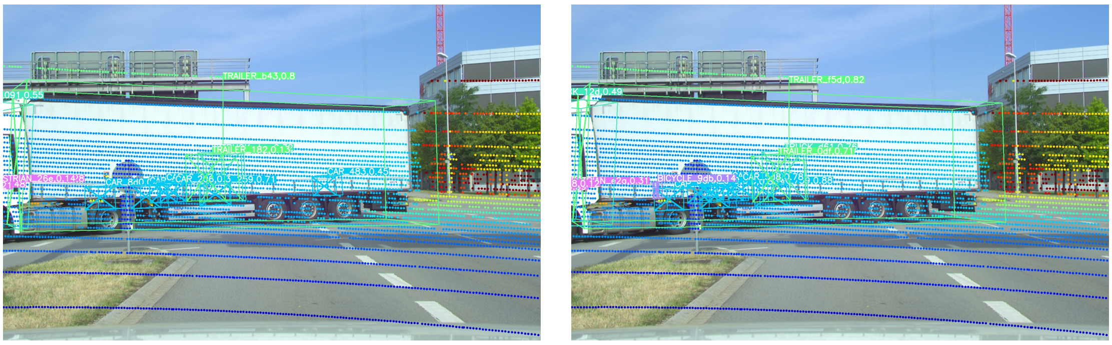


Figure 6.3: Occlusion set 2, vehicle POV: vehicle-only vs. cooperative. The vehicle-only image again suggests that it is able to detect objects behind the occluding white trailer. However, as will be shown in Figure 6.4, these are mere inaccurate guesses/estimations of the model, that are only possible due to the overfitting and low score threshold of 0.1. The vehicle-only image also shows a car behind the trailer to the right that is supposedly not detected by the cooperative model. However, this is a false positive by the vehicle-only model as shown in Figure 6.5.

The four images in Figures 6.4 and 6.5 show the same scene, from the same BEV point cloud as before and the perspective of the south 1 infrastructure camera, which proves that this is not the case.

From the BEV point cloud and south 1 camera perspectives, it is clear that the extra car the vehicle-only model detected is a false positive caused by the overfitting combined with a lack of data due to occlusion. While falsely detecting the car, it actually missed a bicycle near the cars and trucks behind the white trailer. Meanwhile, the seemingly correctly detected vehicles behind the white trailer to the left side are shown to be inaccurately detected by the vehicle-only model when observed from the BEV point cloud perspective. From this perspective, it is shown that the vehicle-only model placed the cars and the truck-trailer combination incorrectly compared to the cooperative model.

These results prove that the proposed cooperative method provides the ego-vehicle with the ability to see through occlusions with a much better accuracy than vehicle-only, proving the benefits of cooperative driving over traditional (vehicle-only) autonomous driving.

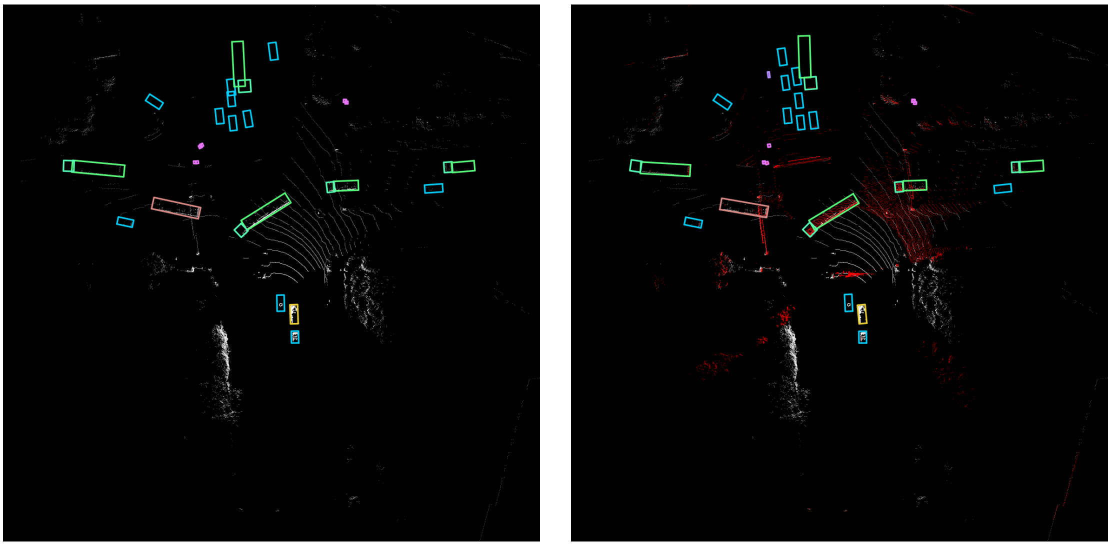


Figure 6.4: Occlusion set 2, BEV point cloud POV: vehicle-only vs. cooperative. The BEV perspective clearly shows that the vehicle-only model inaccurately estimates/guesses the locations of the objects behind the white trailer. Two cars are estimated as intersecting each other, while the truck intersects with its trailer. The vehicle-only model also missed the bicycle near the cars and trucks in the opposite direction of the ego-vehicle behind the white trailer. Meanwhile, the car behind the trailer in the ego vehicle's direction is a false positive, as shown in Figure 6.5.

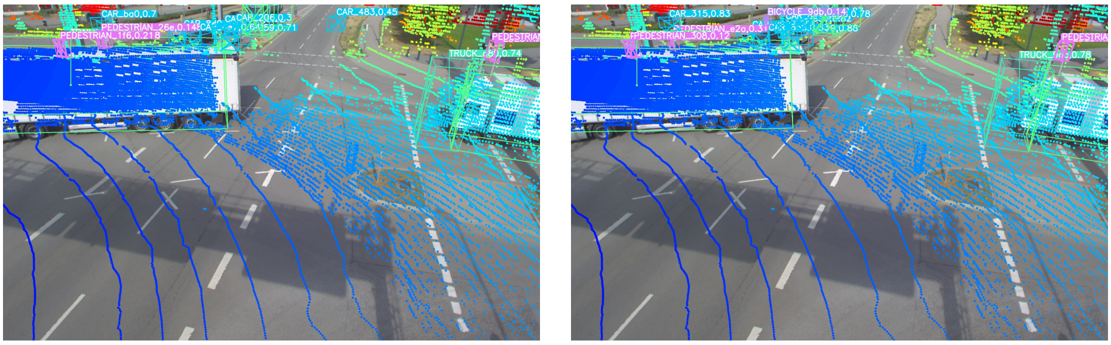


Figure 6.5: Occlusion set 2, south 1 camera POV: vehicle-only vs. cooperative. This south 1 camera perspective of the scenario shown in Figures 6.3 and 6.4 shows that the car the vehicle-only model detected and the cooperative model supposedly missed is actually a false positive. The extra information provided by cooperative driving allowed the cooperative model to avoid this false positive.

6.2.2 Far Objects

Another interesting aspect of the proposed method is its ability to detect objects far away. Based on the quantitative results in Table 5.14, it is shown that the proposed method in the camera-LiDAR fusion mode outperforms InfraDet3D [Zim+23a] by 10.46 mAP in south 1 FOV and 8.88 mAP in south 2 FOV in the hard category (50-64 m).

This can be visually shown using several visualizations made on the TUMTraf Intersection Dataset based on the detections from the retrained, infrastructure-only, PointPillars 512_2x + YOLOv8 s configuration of the proposed method. Two scenes from both the south 1 and south 2 FOVs are shown in Figures 6.6 and 6.7.

The top left corners of the south 1 visualizations in sets 1 and 2 show that the model is capable of detecting both the far away truck and trailer in set 1 and the far away cars in set 2. The south 2 visualizations show that it is able to detect the furthest away cares at the left side of the images and the static far away van in set 2. All of these objects are located in the

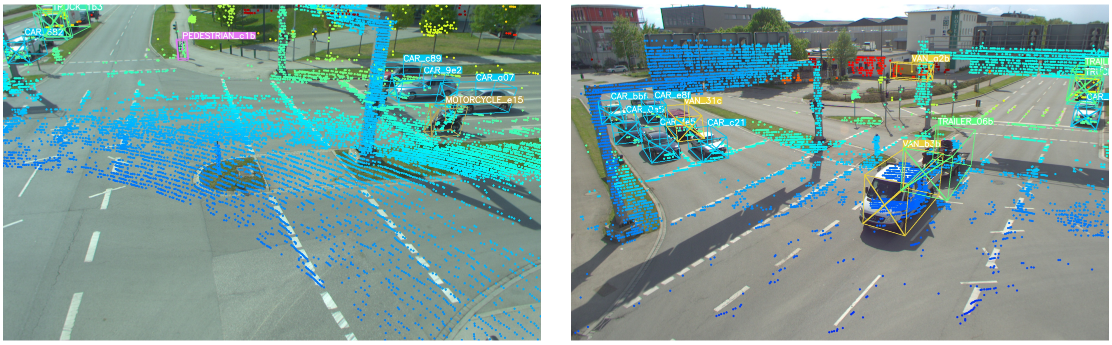


Figure 6.6: Far objects set 1, south 1 and south 2 FOV. The south 1 image shows that the model is able to detect the far-away truck in the top left corner. The south 2 image also shows that it is able to detect the far-away cars at the end of the lines of cars on the left side of the image.

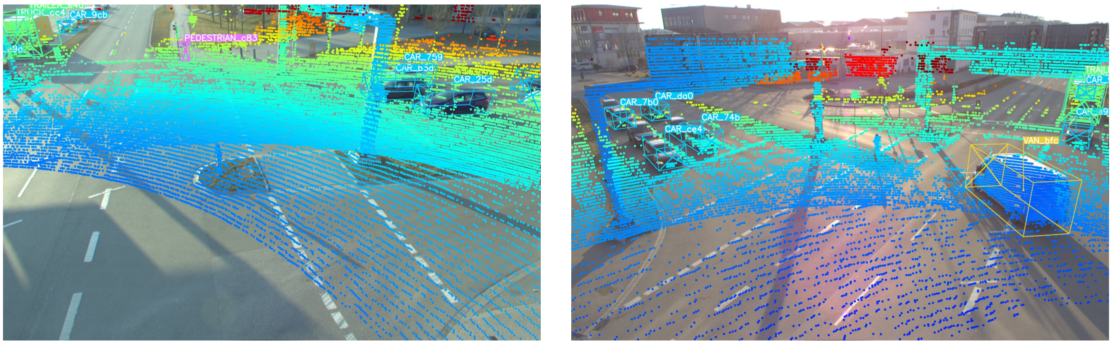


Figure 6.7: Far objects set 2, south 1 and south 2 FOV. The south 1 image shows that the model is able to detect the far-away cars hidden behind the truck. The south 2 image shows the same situation as Figure 6.6 where it is able to detect the far-away cars at the left side of the image.

hard category range and near the edges of the 75 m limit of the geo-fencing. These images show that the proposed method still performs well even with objects far away.

6.2.3 Effect of Camera-LiDAR Fusion

Another interesting observation from the visualizations is the ability of camera-LiDAR fusion to remove false positives compared to camera-only. This is proven in any visualization of camera-only models in the TUMTraf Cooperative dataset. The images shown in Figures 6.8 and 6.9 show the detections of the cooperative, camera-only YOLOv8 s model after retraining and the cooperative fusion configuration of PointPillars 512_2x + YOLOv8 s. Both use a score threshold of 0.1. This camera-only configuration is chosen to give it the best chance by providing the largest amount of data possible and the best-performing camera backbone.

As seen in both sets of images, camera-only produces a huge amount of false positives around the actual detection at a score threshold of 0.1. This is likely caused by the way camera features are projected into a 3D point cloud by the proposed method. The method works in a method proposed by the LSS [PF20]: projecting rays for every pixel in the 3D space creating a 3D frustum, placing discrete points at each ray (each with its own probability that the pixel exists at that location), then multiplying each pixel's feature with each point on its ray to generate the 3D feature point cloud, and finally using BEVPooling to turn it into a BEV feature. This means that the feature map received by the head is effectively a bunch of probabilities along each ray that the pixel is of a certain class. This is likely the cause of the false positives surrounding each true positive in roughly the rays' directions when using

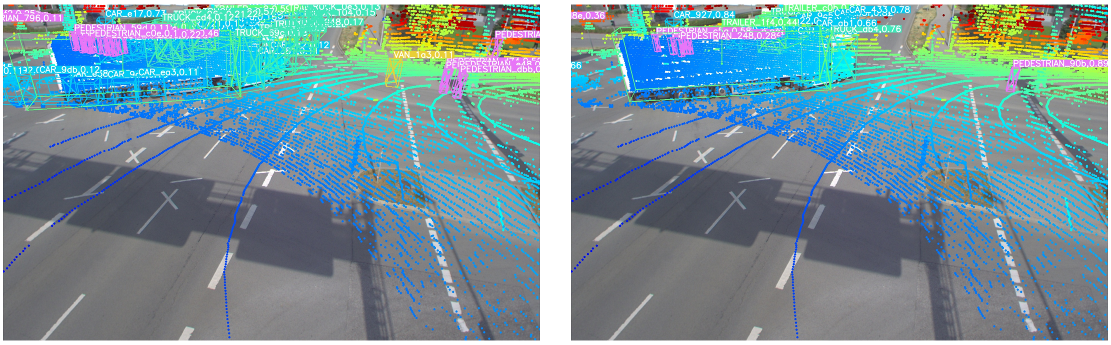


Figure 6.8: South 1 FOV: camera-only vs. camera-LiDAR fusion. The camera-only model image shows a lot of false positives around every true positive due to how the projection to BEV works in the proposed method and the very low score threshold of 0.1. The camera-LiDAR fusion image manages to remove the vast majority of the false positives even at the same score threshold.

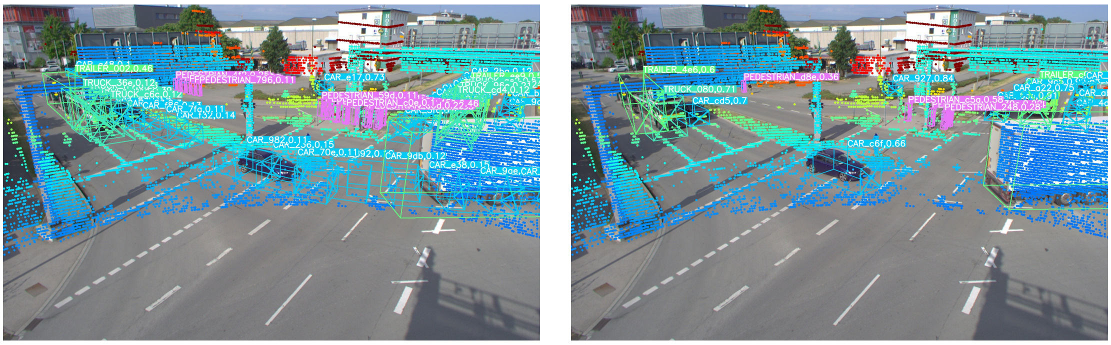


Figure 6.9: South 2 FOV: camera-only vs. camera-LiDAR fusion. The same scenario from Figure 6.8 is repeated. Camera-only results in lots of false positives around every true positive, while the camera-LiDAR fusion manages to remove the vast majority of the false positives, even at a very low score threshold.

camera-only mode.

Introducing LiDAR features to the final feature used for 3D object detection provides the proposed method with an extra, better feature map since the LiDAR features are extracted from each point in the input point cloud. This means the features exist in that 3D (or BEV) location and are not probabilistically projected. This allows the method to learn a better-fused feature, creating a more accurate feature map for the head, which eliminates false positives. This allows the usage of a low score threshold such as 0.1, reducing the amount of false negatives without having so many false positives.

6.2.4 Night Scenes

Unfortunately, the TUMTraf Cooperative Dataset doesn't have night scenes. However, since the proposed method was also evaluated on the TUMTraf Intersection Dataset, which contains night scenes, it is possible to evaluate the night performance of the infrastructure-only camera-LiDAR fusion configuration of the proposed method using that dataset. The visualizations of this scenario are shown in Figures 6.10 and 6.11.

These scenes were recorded in the rain. South 2 images show a lot of light reflections on the floor. However, the infrastructure-only camera-LiDAR fusion model correctly detected most, if not all, of the objects. It managed to detect the far-away van in the south 2 images, it managed to detect the far-away truck-trailer combination and bus in the south 1 images, and so on. This indicates that the proposed method would work well in night scenes.

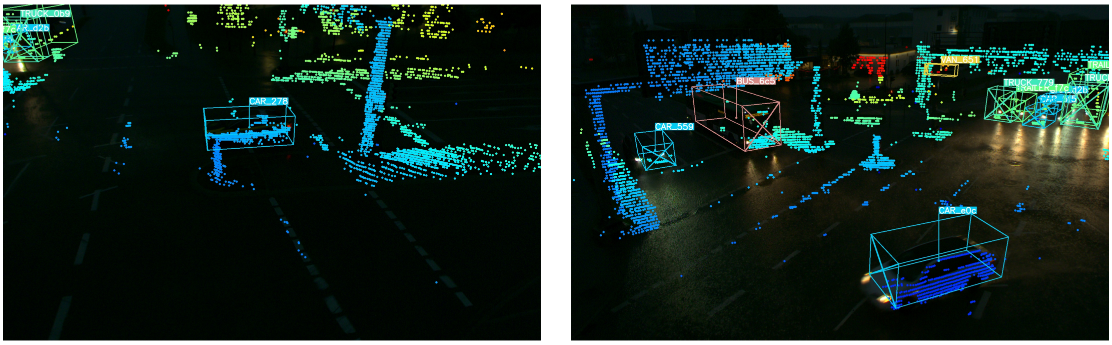


Figure 6.10: First night scene: south 1 and south 2. The scene shows a lot of reflections on the road surface due to the rainy weather. The proposed method manages to detect most, if not all, of the objects in the scene correctly. The biggest contributor in enabling this is likely the LiDAR point cloud data as it is not affected by lighting.

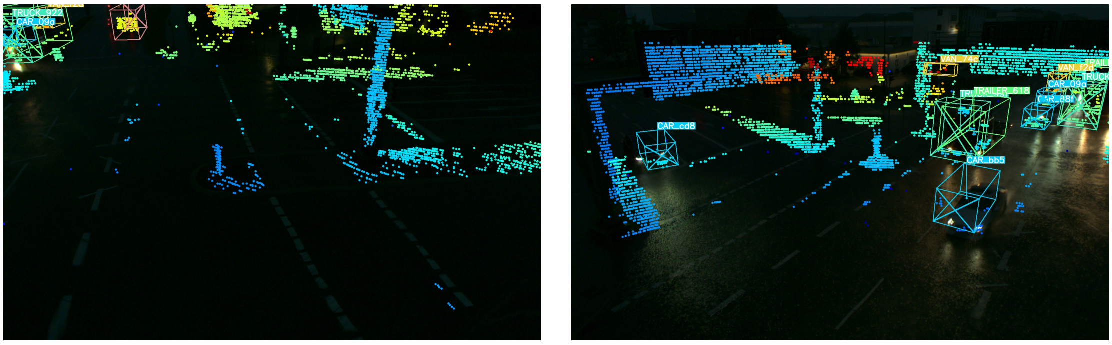


Figure 6.11: Second night scene: south 1 and south 2. A different scene in the same conditions shows that the proposed method is able to detect the objects in the scene properly at night.

6.2.5 Missing Labels in the TUMTraf Cooperative Dataset

Due to the very short amount of time available for recording, preparing, registering, and labeling by hand the frames of the TUMTraf Cooperative Dataset, mistakes were made in the form of the northern side in drive 22 not being labeled at all. Unfortunately, due to the overfitting, this was missed since the model generally always produces good mAP for its configuration. Examples of labeled vs. unlabeled frames with detection (red) and ground truth (green) from the north field of view of the camera are shown in Figure 6.12.

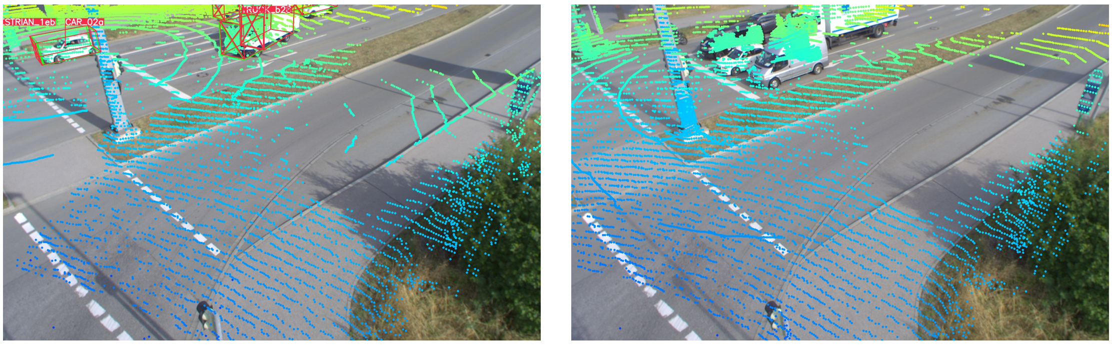


Figure 6.12: A frame that is properly labeled and one with labels missing in the north area. The model correctly detects the objects when labels are there and detects nothing when no labels are there. This is likely caused by the overfitting due to the small amount of data in the dataset.

As mentioned above, this mistake also provides another proof of overfitting. The model detects objects properly when the labels are there and detects nothing when there are no labels even if there are obviously objects there. Unfortunately, this was only discovered very close to the end of the thesis period and there was no more time to retrain all of the configurations with the fixed frames. This should not matter, however, since even with the fixed frames, the model will overfit anyway due to the very small dataset. And, as the proposed method works well in the TUMTraf Intersection Dataset, which contains 2400 frames of data split into 1920 frames for training and 240 frames each for validation and testing, the overfitting is almost certainly caused by the lack of data and no other factors.

Chapter 7

Future Work

This chapter contains recommendations for future work related to the methods proposed in this thesis. Several possible areas exist for improving the performance of the proposed method. This includes, but is not limited to, the dataset used, accelerating and optimizing the method, the camera-LiDAR fusion method, and the vehicle-infrastructure fusion method.

7.1 Dataset

As mentioned in Chapter 3, the dataset used for cooperative experiments was the TUMTraf Cooperative Dataset. Currently, this dataset is limited by its small size (500 data frames, each with 2 LiDAR point clouds, 4 images, and 1 label), which results in heavy overfitting in the experiments presented in this thesis. This could be improved in the future by adding more data frames to the TUMTraf Cooperative Dataset, including new scenarios, such as night and bad weather scenes, and instances of classes that are currently missing or underrepresented (MOTORCYCLE, BICYCLE, OTHER, and EMERGENCY_VEHICLE).

A good target would be around 2000 data frames at the minimum with night scenes and all of the object classes represented as the results and lack of overfitting from the TUMTraf Intersection Dataset experiments with 2400 frames and all object classes represented showed. An alternative would be training and testing the proposed method on larger cooperative datasets like the DAIR-V2X-C Dataset [Yu+22b], or even using it to pre-train the model before transfer learning to the TUMTraf Cooperative Dataset.

Another improvement that is possible on this front lies in the quality of the data contained in the TUMTraf Cooperative Dataset by fixing the missing labels mentioned in Chapter 6. This will improve the performance of the proposed method and/or prevent overfitting, as this prevents unlabeled foreground objects from being considered/learned as background objects.

7.2 Acceleration and Optimization

Although the proposed method currently manages to run in real-time using three different configurations as shown in Chapter 5, they are just barely above 10 FPS. A different GPU with less performance than the RTX 3090 used in the experiments would likely drop the speed of even these three configurations below 10 FPS.

One possible solution to this issue is to optimize the proposed method. Currently, the method is using hard voxelization. Unfortunately, even with non-deterministic hard voxelization, this part is still sub-optimal. A possible solution to this is to use dynamic voxelization.

Dynamic voxelization is considerably faster but there is a risk of it using more VRAM than hard voxelization. This is one improvement that is worth further investigation.

Another solution is to port the proposed method for usage with TensorRT. This would allow the model to instantly run faster than currently, but it will require a lot of extra work. The method is based on BEVFusion [Liu+23b] and as such utilizes a lot of custom GPU kernels and CUDA operations. These need to be ported manually to TensorRT. A possible example of this is CUDA-BEVFusion [NVI23].

Another aspect that could be monitored is the backend of sparse convolution. The baseline BEVFusion [Liu+23b] used SPConv v2 as its backend. This was later replaced with Torchsparse [Tan+22], improving VRAM usage and runtime performance. If a new and better backend for this part of the proposed were to be created in the future, it would be a good idea to try to integrate it into the proposed method.

7.3 Backbone Transfer Learning

Currently, the YOLOv8 camera backbone is pre-trained on the COCO dataset [Lin+14]. Unfortunately, a lot of the classes in the TUMTraf Dataset don't exist in the COCO dataset. The classes: VAN, TRAILER, EMERGENCY_VEHICLE, and OTHER are missing, while PEDESTRIAN is named PERSON, and TRUCK is defined differently from TUMTraf Dataset. A possible path forward is to pre-train the YOLOv8 backbone on all three releases of the TUMTraf dataset. Another alternative is to pre-train it on other datasets like nuScenes/nuImages [Cae+20] or DAIR-V2X-C [Yu+22a] and then fine-tuning on the three releases of the TUMTraf dataset.

7.4 Camera-LiDAR Deep Fusion Method

Another interesting possibility for improving the proposed method is to use a different baseline for camera-LiDAR fusion. Currently, the model uses the camera-LiDAR deep fusion method of BEVFusion [Liu+23b] as the baseline, which works in the bird's eye view (BEV) perspective and uses a convolution-based fuser to fuse the BEV camera and LiDAR features. It would be interesting to see alternative camera-LiDAR deep fusion methods, such as DeepFusion [Li+22] and Cross-Modal Transformer (CMT), which are instead based on the transformer architecture, or the current best method on the nuScenes 3D Object Detection leaderboards: Edge Aware, Lift Splat Shoot (EA-LSS) [Hu+23].

7.5 Vehicle-Infrastructure Deep Fusion Method

Other than the camera-LiDAR fusion method, the vehicle-infrastructure fusion method could also be improved as part of a hypothetical future work. Currently, the proposed method uses its own implementation of element-wise max fuser, a very rudimentary approach inspired by PillarGrid [Bai+22b]. Possible alternatives to this would be to use a convolution-based fuser, inspired by the camera-LiDAR fuser of BEVFusion [Liu+23b] or the transformer-based method proposed by V2X-ViT [Xu+22]. It would be interesting to see if technically heavier, transformer-based methods are still able to run in real-time.

7.6 Implementation on the Live System

The final goal of the proposed method is to enable it to be run and tested on a live system with a real car and at the real S110 intersection. The final suggestion for future work is related to this final goal. To make this possible, multiple issues need to be solved first. First, synchronization between vehicle and infrastructure need to be ensured. Hard synchronization between every node and sensor needs to be implemented. Latency issues need to be solved by finding a V2I communication method with low latency. Methods that are able to realign the features to minimize the effects of inevitable latency during transmission, such as FFNet [Yu+23] and CoBEVFlow [Wei+23] could be used as inspirations in helping to solve this issue.

Another issue that needs to be solved here is the registration between vehicle and infrastructure and calculating the transformation matrix from vehicle to infrastructure. One possible solution is to utilize GPS coordinates to calculate this transformation matrix. The important thing is, that no matter which method is chosen, it must be fast and light enough to allow real-time operation.

Finally, several more issues need to be solved to allow the proposed method to fully run live in the research group's toolchain. The first step would be to create a function that allows the model to work without using the .pkl dataset files and the dataset class plus the data loader from PyTorch/MMDetection3D. This would allow the method to load and run inference on unlabeled data. On top of this, some work needs to be done to allow the proposed method to work with the toolchain's shared memory system for loading images and point cloud data. ROS packages, subscribers to shared memory images and point clouds, and detection publishers need to be created. The received data need to also be properly formatted similar to the data from .pkl datasets after it goes through the data pipeline.

Chapter 8

Conclusion

Autonomous driving is an emerging field of technology and research that will only become more commonly used in the future. Currently, most of the research in this field is focused on onboard autonomous driving. However, this approach inherently is weak against occlusions. Infrastructure-based sensors are placed higher up and as such have a much better view of the situation in its surroundings. Utilizing this, the approach of cooperative driving, combining both onboard (vehicle) and infrastructure-based sensors is a promising approach for future research.

Another aspect of autonomous driving is the sensor modality used to gain information about the scene. Each sensor type has its own weaknesses. For example, cameras require good lighting and the produced images lack accurate 3D information. Meanwhile, LiDARs are expensive and weak to certain weather conditions such as rain and fog due to the reflection and refraction of the LiDAR beams from water particles. As such, the best approach is to combine both modalities, introduce redundancy, and allow one sensor to cover the weaknesses of the other sensor.

However, using both camera and LiDAR on both vehicle and infrastructure means that a large amount of data needs to be processed for each frame. From the considerations made in this thesis, it was found that deep fusion (fusing the features extracted from the raw data) provides the best trade-off between accuracy and speed as the features are lighter than raw data, but still provide more information to fuse than simple detection results.

Based on this, this thesis proposed a deep multi-modal cooperative approach of fusing camera and LiDAR sensors of the vehicle and infrastructure to perform the 3D object detection task. The approach is inspired by both BEVFusion [Liu+23b] and PillarGrid [Bai+22b], using BEVFusion as the baseline code and as such also working in the bird's eye view (BEV) perspective. It first extracts 3D features from the cameras and LiDAR of the vehicle or infrastructure, transforms both into a unified BEV perspective, and fuses the camera and LiDAR feature together using a convolution-based fuser. The separate and pre-aligned fused vehicle and infrastructure camera-LiDAR features are then fused together using an element-wise max operation to create the final fused feature. This is then used by the Transfusion [Bai+22a] head to perform the 3D object detection task.

This method is then trained and tested on a newly created TUMTraf Cooperative Dataset and TUMTraf Intersection Dataset. The TUMTraf Cooperative Dataset is small for the purpose of this task and as a result, the model is overfitting. However, the general trends from the results proved the initial proposals of this thesis: camera-LiDAR fusion is better than camera-only or LiDAR-only, cooperative is better than vehicle-only or infrastructure-only, it is possible to create a configuration of a cooperative, multimodal method that runs in real-time (10 Hz or more), trainable on a high-end mainstream GPU (RTX 3090), and runnable on middle-class mainstream GPUs with 8 GB of VRAM.

The experiments performed on the TUMTraf Cooperative Dataset also prove that this proposed method is also able to beat a state-of-the-art, infrastructure-only, camera-LiDAR late fusion model in the form of InfraDet3D [Zim+23a]. It outperformed InfraDet3D in both LiDAR-only and camera-LiDAR fusion modes in both the south 1 and south 2 camera fields of view in all 3D mAP metrics, except for hard detections (50-64 m) in LiDAR-only mode, while still being able to run in real-time.

Finally, a list of possible approaches to improve the proposed method is provided. Including, but not limited to improving the TUMTraf Cooperative Dataset with more data, more class variety, and better labeling; using other cooperative datasets like the DAIR V2X-C [Yu+22a] dataset to pre-train the model before fine-tuning on the TUMTraf Cooperative Dataset; optimizing and accelerating the model using approaches like dynamic voxelization, porting to TensorRT, and using a new backend; utilizing transfer learning in the camera backbone; using different camera-LiDAR and/or vehicle-infrastructure deep fusion methods; and finally implementing the new methods and updates needed to allow the proposed method to run in a live system.

List of Figures

List of Tables

Listings

Bibliography

[Bai+22a] Bai, X., Hu, Z., Zhu, X., Huang, Q., Chen, Y., Fu, H., and Tai, C.-L. “Transfusion: Robust lidar-camera fusion for 3d object detection with transformers”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 1090–1099.

[Bai+23] Bai, Z., Nayak, S. P., Zhao, X., Wu, G., Barth, M. J., Qi, X., Liu, Y., Sisbot, E. A., and Oguchi, K. “Cyber Mobility Mirror: A Deep Learning-based Real-World Object Perception Platform Using Roadside LiDAR”. In: *IEEE Transactions on Intelligent Transportation Systems* (2023).

[Bai+22b] Bai, Z., Wu, G., Barth, M. J., Liu, Y., Sisbot, E. A., and Oguchi, K. “Pillargrid: Deep learning-based cooperative perception for 3d object detection from onboard-roadside lidar”. In: *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2022, pp. 1743–1749.

[Bai+22c] Bai, Z., Wu, G., Barth, M. J., Liu, Y., Sisbot, E. A., Oguchi, K., and Huang, Z. “A Survey and Framework of Cooperative Perception: From Heterogeneous Singleton to Hierarchical Cooperation”. In: *arXiv preprint arXiv:2208.10590* (2022).

[Bai+22d] Bai, Z., Wu, G., Qi, X., Liu, Y., Oguchi, K., and Barth, M. J. “Infrastructure-based object detection and tracking for cooperative driving automation: A survey”. In: *2022 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2022, pp. 1366–1373.

[BD19] Bezdan, T. and Džakula, N. B. “Convolutional neural network layers and architectures”. In: *International Scientific Conference on Information Technology and Data Related Research*. Singidunum University Belgrade, Serbia. 2019, pp. 445–451.

[Cae+20] Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. “nuscenes: A multimodal dataset for autonomous driving”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11621–11631.

[Che+19a] Chen, Q., Ma, X., Tang, S., Guo, J., Yang, Q., and Fu, S. “F-cooper: Feature based cooperative perception for autonomous vehicle edge computing system using 3D point clouds”. In: *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. 2019, pp. 88–100.

[Che+19b] Chen, Q., Tang, S., Yang, Q., and Fu, S. “Cooper: Cooperative perception for connected autonomous vehicles based on 3d point clouds”. In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2019, pp. 514–524.

[Com23] Company, M.
bibinitperiod. *Autonomous driving’s future: Convenient and connected*. 2023. URL: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected (visited on 10/10/2023).

[Con22] Contributors, M. *MMYOLO: OpenMMLab YOLO series toolbox and benchmark*. https://github.com/open-mmlab/mmyolo. 2022.

[DT05] Dalal, N. and Triggs, B. "Histograms of oriented gradients for human detection". In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. Ieee. 2005, pp. 886–893.

[Dos+20] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[Gir15] Girshick, R. "Fast r-cnn". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1440–1448.

[Gir+14] Girshick, R., Donahue, J., Darrell, T., and Malik, J. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 580–587.

[Glo18] Global, S. *Autonomous vehicle sales to surpass 33 million annually in 2040 enabling new autonomous mobility in more than 26% of new car sales*. 2018. URL: https://www.spglobal.com/mobility/en/research-analysis/autonomous-vehicle-sales-to-surpass-33-million-annually-in-2040-enabling-new-autonomous-mobility-in-more-than-26-percent-of-new-car-sales.html (visited on 10/10/2023).

[He+16] He, K., Zhang, X., Ren, S., and Sun, J. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[HF17] Henderson, P. and Ferrari, V. "End-to-end training of object class detectors for mean average precision". In: *Computer Vision–ACCV 2016: 13th Asian Conference on Computer Vision, Taipei, Taiwan, November 20-24, 2016, Revised Selected Papers, Part V 13*. Springer. 2017, pp. 198–213.

[Hu+23] Hu, H., Wang, F., Su, J., Wang, Y., Hu, L., Fang, W., Xu, J., and Zhang, Z. "EA-LSS: Edge-aware Lift-splat-shot Framework for 3D BEV Object Detection". In: *arXiv preprint arXiv:2303.17895* (2023).

[Hu+22] Hu, Y., Fang, S., Lei, Z., Zhong, Y., and Chen, S. "Where2comm: Communication-efficient collaborative perception via spatial confidence maps". In: *Advances in neural information processing systems* 35 (2022), pp. 4874–4886.

[Inc19a] Inc., M. A. *nuScenes CPVR 2019 Object Detection Settings*. 2019. URL: https://github.com/nutonomy/nuscenes-devkit/blob/master/python-sdk/nuscenes/eval/detection/configs/detection_cvpr_2019.json (visited on 09/29/2023).

[Inc19b] Inc., M. A. *nuScenes Dataset Devkit*. 2019. URL: https://github.com/nutonomy/nuscenes-devkit/ (visited on 09/29/2023).

[Inc23] Inc., M. A. *nuScenes 3D Object Detection Leaderboards*. 2023. URL: https://www.nuscenes.org/object-detection?externalData=all&mapData=all&modalities=Any (visited on 09/23/2023).

[KSH17] Krizhevsky, A., Sutskever, I., and Hinton, G. E. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[Lan+19] Lang, A. H., Vora, S., Caesar, H., Zhou, L., Yang, J., and Beijbom, O. "Pointpillars: Fast encoders for object detection from point clouds". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 12697–12705.

[Li+20] Li, X., Wang, W., Wu, L., Chen, S., Hu, X., Li, J., Tang, J., and Yang, J. “Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 21002–21012.

[Li+21] Li, Y., Ren, S., Wu, P., Chen, S., Feng, C., and Zhang, W. “Learning distilled collaboration graph for multi-agent perception”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 29541–29552.

[Li+22] Li, Y., Yu, A. W., Meng, T., Caine, B., Ngiam, J., Peng, D., Shen, J., Lu, Y., Zhou, D., Le, Q. V., et al. “Deepfusion: Lidar-camera deep fusion for multi-modal 3d object detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 17182–17191.

[Lin+17] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. “Focal loss for dense object detection”. In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2980–2988.

[Lin+14] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. “Microsoft coco: Common objects in context”. In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. Springer. 2014, pp. 740–755.

[Liu+22] Liu, S., Yu, B., Tang, J., Zhu, Y., and Liu, X. “Communication challenges in infrastructure-vehicle cooperative autonomous driving: A field deployment perspective”. In: *IEEE Wireless Communications* 29.4 (2022), pp. 126–131.

[Liu+16] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. “Ssd: Single shot multibox detector”. In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*. Springer. 2016, pp. 21–37.

[Liu+21] Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. “Swin transformer: Hierarchical vision transformer using shifted windows”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 10012–10022.

[Liu+23a] Liu, Z., Tang, H., Amini, A., Yang, X., Mao, H., Rus, D., and Han, S. *BEVFusion Original ObjectPaste Sampling parameters*. 2023. URL: https://github.com/mit-han-lab/bevfusion/blob/601961a903c46d14ece4606b8acfe86e604499df/configs/nuscenes/default.yaml#L101 (visited on 09/29/2023).

[Liu+23b] Liu, Z., Tang, H., Amini, A., Yang, X., Mao, H., Rus, D. L., and Han, S. “Bevfusion: Multi-task multi-sensor fusion with unified bird’s-eye view representation”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 2774–2781.

[MGJ21] Misra, I., Girdhar, R., and Joulin, A. “An end-to-end transformer model for 3d object detection”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 2906–2917.

[NVI23] NVIDIA-AI-IOT. *CUDA-BEVFusion*. 2023. URL: https://github.com/NVIDIA-AI-IOT/Lidar_AI_Solution/tree/master/CUDA-BEVFusion (visited on 10/08/2023).

[OD08] Olson, D. L. and Delen, D. *Advanced Data Mining Techniques*. 1st edition (February 1, 2008). Springer, 2008, p. 138. ISBN: 3-540-76916-1.

[Ope23] OpenMMLab. *MMYOLO YOLOv8 COCO Pre-trained Weights Table*. 2023. URL: https://github.com/open-mmlab/mmyolo/tree/main/configs/yolov8#coco (visited on 10/04/2023).

[PMR20] Pang, S., Morris, D., and Radha, H. “CLOCs: Camera-LiDAR object candidates fusion for 3D object detection”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 10386–10393.

[PF20] Philion, J. and Fidler, S. “Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d”. In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*. Springer. 2020, pp. 194–210.

[Qi+17a] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. “Pointnet: Deep learning on point sets for 3d classification and segmentation”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[Qi+17b] Qi, C. R., Yi, L., Su, H., and Guibas, L. J. “Pointnet++: Deep hierarchical feature learning on point sets in a metric space”. In: *Advances in neural information processing systems* 30 (2017).

[Qia+22] Qian, G., Li, Y., Peng, H., Mai, J., Hammoud, H. A. A. K., Elhoseiny, M., and Ghanem, B. “Pointnext: Revisiting pointnet++ with improved training and scaling strategies”. In: *arXiv preprint arXiv:2206.04670* (2022).

[Red+16] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[Ren+15] Ren, S., He, K., Girshick, R., and Sun, J. “Faster r-cnn: Towards real-time object detection with region proposal networks”. In: *Advances in neural information processing systems* 28 (2015).

[SZ14] Simonyan, K. and Zisserman, A. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[Sta22] Statista. *Number of autonomous vehicles globally in 2022, with a forecast through 2030*. 2022. URL: https://www.statista.com/statistics/1230664/projected-number-autonomous-cars-worldwide (visited on 10/10/2023).

[Sun+20] Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., et al. “Scalability in perception for autonomous driving: Waymo open dataset”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2446–2454.

[Sun+22] Sun, P., Sun, C., Wang, R., and Zhao, X. “Object detection based on roadside LiDAR for cooperative driving automation: a review”. In: *Sensors* 22.23 (2022), p. 9316.

[Tan+22] Tang, H., Liu, Z., Li, X., Lin, Y., and Han, S. “TorchSparse: Efficient Point Cloud Inference Engine”. In: *Conference on Machine Learning and Systems (MLSys)*. 2022.

[Tea23] Team, T. T. D. *TUMTraf Dataset Devkit*. 2023. URL: https://github.com/tum-traffic-dataset/tum-traffic-dataset-dev-kity (visited on 10/01/2023).

[TC23] Terven, J. and Cordova-Esparza, D. “A comprehensive review of YOLO: From YOLOv1 to YOLOv8 and beyond”. In: *arXiv preprint arXiv:2304.00501* (2023).

[Ult23] Ultralytics. *Ultralytics YOLOv8*. 2023. URL: https://github.com/ultralytics/ultralytics (visited on 07/08/2023).

[Vas+17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[VJ01] Viola, P. and Jones, M. "Rapid object detection using a boosted cascade of simple features". In: *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*. Vol. 1. Ieee. 2001, pp. I–I.

[Wan+20] Wang, T.-H., Manivasagam, S., Liang, M., Yang, B., Zeng, W., and Urtasun, R. "V2vnet: Vehicle-to-vehicle communication for joint perception and prediction". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*. Springer. 2020, pp. 605–621.

[Wei+23] Wei, S., Wei, Y., Hu, Y., Lu, Y., Zhong, Y., Chen, S., and Zhang, Y. "Robust Asynchronous Collaborative 3D Detection via Bird's Eye View Flow". In: *arXiv preprint arXiv:2309.16940* (2023).

[Wik23] Wikipedia contributors. *Stratified sampling — Wikipedia, The Free Encyclopedia*. [Online; accessed 26-September-2023]. 2023. URL: https://en.wikipedia.org/wiki/Stratified_sampling.

[Xu+22] Xu, R., Xiang, H., Tu, Z., Xia, X., Yang, M.-H., and Ma, J. "V2X-ViT: Vehicle-to-everything cooperative perception with vision transformer". In: *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXIX*. Springer. 2022, pp. 107–124.

[YML18] Yan, Y., Mao, Y., and Li, B. "Second: Sparsely embedded convolutional detection". In: *Sensors* 18.10 (2018), p. 3337.

[YZK21] Yin, T., Zhou, X., and Krahenbuhl, P. "Center-based 3d object detection and tracking". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 11784–11793.

[Yu+22a] Yu, H., Luo, Y., Shu, M., Huo, Y., Yang, Z., Shi, Y., Guo, Z., Li, H., Hu, X., Yuan, J., et al. "Dair-v2x: A large-scale dataset for vehicle-infrastructure cooperative 3d object detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 21361–21370.

[Yu+22b] Yu, H., Luo, Y., Shu, M., Huo, Y., Yang, Z., Shi, Y., Guo, Z., Li, H., Hu, X., Yuan, J., and Nie, Z. "DAIR-V2X: A Large-Scale Dataset for Vehicle-Infrastructure Cooperative 3D Object Detection". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2022, pp. 21361–21370.

[Yu+23] Yu, H., Tang, Y., Xie, E., Mao, J., Yuan, J., Luo, P., and Nie, Z. "Vehicle-infrastructure cooperative 3d object detection via feature flow prediction". In: *arXiv preprint arXiv:2303.10552* (2023).

[Zha+18] Zhang, S., Chen, J., Lyu, F., Cheng, N., Shi, W., and Shen, X. "Vehicular communication networks in the automated driving era". In: *IEEE Communications Magazine* 56.9 (2018), pp. 26–32.

[Zhe+20] Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., and Ren, D. "Distance-IoU loss: Faster and better learning for bounding box regression". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 07. 2020, pp. 12993–13000.

[ZWK19] Zhou, X., Wang, D., and Krähenbühl, P. "Objects as points". In: *arXiv preprint arXiv:1904.07850* (2019).

[ZT18] Zhou, Y. and Tuzel, O. "Voxelnet: End-to-end learning for point cloud based 3d object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 4490–4499.

[Zim+23a] Zimmer, W., Birkner, J., Brucker, M., Nguyen, H. T., Petrovski, S., Wang, B., and Knoll, A. C. "Infradet3d: Multi-modal 3d object detection based on roadside infrastructure camera and lidar sensors". In: *arXiv preprint arXiv:2305.00314* (2023).

[Zim+23b] Zimmer, W., Creß, C., Nguyen, H. T., and Knoll, A. C. "A9 Intersection Dataset: All You Need for Urban 3D Camera-LiDAR Roadside Perception". In: *arXiv preprint arXiv:2306.09266* (2023).

[ZRT19] Zimmer, W., Rangesh, A., and Trivedi, M. "3d bat: A semi-automatic, web-based 3d annotation toolbox for full-surround, multi-modal data streams". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 1816–1821.