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

Master's Thesis in Automotive Engineering

Mamba-based 3D Object Detection with Object Relation

Mamba-basierte 3D-Objekterkennung mit Objektbeziehung

Supervisor Prof. Dr.-Ing. habil. Alois C. Knoll

Advisor Mingyu Liu, M.Sc.; Bare Luka Žagar, M.Sc.

Author Jun Meng

Date June 15, 2024 in Munich

Disclaimer

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

Munich, June 15, 2024

(Jun Meng)

Abstract

3D object detection with high accuracy and real-time capability is critical for ensuring the driving safety of autonomous vehicles. While the state-of-the-art two-stage 3D object detectors have exhibited promising performance, they refine proposals individually, ignoring the rich contextual information with the neighbor proposals. Besides, traditional CNN-based encoders cause high computational complexity. Recently Mamba has gained great attention by applying the state space model (SSM), which is of linear complexity, in natural language processing (NLP) and has been transplanted to computer vision (CV) domain agilely. In this work, we introduce a light-weighted inter-object relation based 3D object detection framework, highlighted with our proposed Mamba-based 2D backbone with visual state space block (VSS-block) and an object relation module based on message-passing graph neural network (GNN). Our approach improves upon the baseline model PV-RCNN on the KITTI validation set for the car class across easy, moderate, and hard difficulty levels by 0.54%, 0.74%, and 0.52%, respectively. The Mamba-based 2D backbone reduces FLOPs by 38% and the amount of parameters by 20% for PV-RCNN. Additionally we also conduct experiments with PV-RCNN as baseline model on Waymo Open Dataset and PartA2 as baseline model on KITTI dataset.

Zusammenfassung

3D-Objekterkennung mit hoher Genauigkeit und Echtzeitfähigkeit ist entscheidend für die Fahrsicherheit der autonomen Fahrzeuge. Während die modernsten zweistufigen 3D-Objektdetektoren vielversprechende Leistungen gezeigt haben, verfeinern sie Proposals einzeln und ignorieren dabei die reichhaltigen Kontextinformationen der benachbarten Proposals. Außerdem verursachen traditionelle auf CNN basierende Encoder eine hohe Rechenkomplexität. Kürzlich hat Mamba große Aufmerksamkeit erlangt, indem es das Zustandsraummodell (SSM), das eine lineare Komplexität aufweist, in der Natural Language Processing (NLP) anwendet und agil in den Bereich der Computer Vision (CV) übertragen wurde. In dieser Arbeit stellen wir ein leichtgewichtiges, auf Inter-Objekt-Beziehungen basierendes 3D-Objekterkennungs-Framework vor, implementiert mit unserem vorgeschlagenen Mamba-basiertes 2D-Backbone mit Visual State Space block (VSS-Block) und ein Objektbeziehungsmodul, das auf einem Graph-Neural-Network (GNN) mit Message-Passing basieret. Unser Ansatz verbessert das Basismodell PV-RCNN auf dem KITTI-Validierungsdatensatz für die Fahrzeugklasse in den Schwierigkeitsstufen leicht, mittel und schwer um 0,54%, 0,74% und 0,52%. Das Mamba-basierte 2D-Backbone reduziert die FLOPs um 38% und die Anzahl der Parameter um 20% für PV-RCNN. Zusätzlich führen wir auch Experimente mit PV-RCNN als Basismodell auf dem Waymo Open Dataset und PartA2 als Basismodell auf dem KITTI Datensatz durch.

Contents

Chapter 1

Introduction

Autonomous driving, as well as advanced driver assistance system (ADAS), aims to reduce accidents and improve transportation efficiency by making the vehicles precisely perceive and intelligently interact with other agents in the surrounding environment [Zho+23]. From the early DARPA Challenge, autonomous driving has moved from experimental stages to real-world testing and limited commercial deployment in the recent years. Autonomous vehicles rely on a deep stack of software that encompasses everything from low-level hardware interactions to high-level decision-making algorithms. Typically, the entire software pipeline consists of modules like perception, localization, planing and control. Hence, an effective and robust perception system is pivotal for the safety of autonomous vehicles.

Object detection is one of the most important tasks for ensuring a reliable perception system. The point cloud-based 3D object detection approaches show their advantages over the image-based 2D object detection in the perspective of scene representation due to the accurate depth information. Taking point clouds from LiDAR as input, 3D object detection predicts objects' locations, orientations, bounding box sizes and categories. In most of the open datasets and benchmarks, categories of vehicles, pedestrians and cyclists are considered.

The objective of this work is to improve autonomous vehicles' capabilities in 3D object detection. The upcoming sections of this chapter will dive into the importance of advancing autonomous vehicle technology and will outline the contributions and structure of this work.

1.1 Motivation

Due to sparsity and irregularity natures of point cloud, distant objects incline to be represented with only a few points on their surfaces. Besides, occlusions also lead to incomplete representations of objects. All these problems state challenges for robustness of 3D object detection. The current efficient LiDAR-based detection frameworks are lacking in exploiting object relations. Most of the works only implicitly explore object relations through the hidden features of the neural networks. However, these ignored geometric and location-related features can provide rich contextual and structural information for scene understanding. We believe that exploiting object relation explicitly can improve object recognition performance [Wu+22].

To build object relations within a single 3D point cloud scene, we construct a sparse undirected graph upon the object proposals, where each object proposal is viewed as a node. The corresponding edges between nodes can be generated upon spatial distances physically, or upon nearest neighbors topologically. Such a sparse graph allows us to iteratively refine the object relations and features through efficient message passing with a Graph Neural Network (GNN). The graph-construction and message-passing functions are capsuled as object-

relation module in the following of this work.

In autonomous driving, real-time capability is required under limited computational resources, algorithms of low complexity and high efficiency is desired. To reduce the parameter amount and improve the calculation efficiency, we try to use the recently proposed state space model (SSM) to replace the 2D convolution neural networks in some specific modules.

1.2 Contributions

This work aims to propose an object relation-based lightweight 3D object detection network. Specifically, we replace the 2D convolution layers in the baseline models' 2D BEV backbone with the Visual State Space block (VSS block) to reduce model's complexity. Furthermore, we explore inter-object relationships by integrating a GNN-based object relation module to the baseline model to improve the detection performance.

We take PVRCNN [Shi+21] and PartA2 [Shi+20] as our baseline model. We introduce PV-RCNN_Rel as the implementation with the GNN-based Object-Relation module, the PV-RCNN_Mamba as the implementation with VSS-Block and the PV-RCNN_Rel_Mamba as the implementation with the both modifications. Overall, the contributions of this work can be summarized as below:

- We propose a light-weighted inter-object relation-based 3D object detection network.
- We introduce a plug and play GNN-based object relation module to form a context-dependent end-to-end trainable 3D object detection pipeline. The proposal feature is refined by the explicitly learned object relation features from inter-object relationship graphs.
- We replace 2D convolution layers in the 2D BEV backbone with VSS-block to reduce model's complexity without losing detection performance.
- We conduct extensive experiments on the real-world datasets KITTI and Waymo Open Dataset. We verify the superiority of implementation of VSS-block over both baseline detector and detector with object relation module integrated respectively. The detection performance is estimated in various object classes and difficulty levels. The computational efficiency is evaluated with FLOPs and the amount of parameters of the entire network. We use PVRCNN and PartA2 as baseline models.

1.3 Structure

This work is organized in six chapters. After a brief introduction of the work, some theoretical background would be explained in chapter 2 to familiarize the readers with the generic autonomous driving software pipeline, the sensors for perception, and the modern object detection approaches. The basic deep learning methodologies like convolution neural networks (CNNs), graph neural networks (GNNs) and the recently proposed visual state space models (VSSM) would also be discussed. Chapter 3 presents related works about various approaches of 3D object detection and literature reviews on exploring object relation and applying state space models on the recognition of 2D images and 3D point clouds. Chapter 4 articulates the detailed methodologies of our implementations. Chapter 5 presents the experiment results in both qualitative and quantitative insights, as well as the analysis on ablation studies. Finally, this work ends in chapter 6 by giving the conclusion and limitation of this work and discussion on future directions.

Chapter 2

Theoretical Background

3D object detection is one of the active research fields of autonomous driving in both software and hardware domains. This chapter is dedicated to explain the necessary theoretical background, including the generic autonomous driving software pipeline and the conceptions of perception module, the related sensors, the basic deep learning methodology and common object detection approaches.

2.1 Autonomous Driving System Pipeline

The traditional modular system architecture [Yur+20; Lev+11] of autonomous driving software (ADS) stack are structured as a pipeline of separate components linking sensory inputs to actuator outputs (Fig. 2.1). First the raw data would be processed by the perception module, then followed by localization, planning and prediction modules to get an optimal and safe trajectory. Finally, the control module outputs control signals to the respective actuators to drive the vehicle with the expected maneuvers.

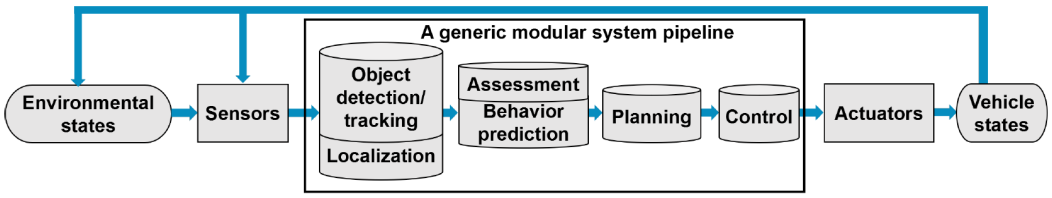


Figure 2.1: A generic modular system architecture of ADS [Yur+20]

Developing individual modules separately divides the complex automated driving task into a set of easier problems. These sub-tasks share corresponding literature in robotics, computer vision and vehicle dynamics, making the accumulated know-how and expertise directly transferable. In addition, the functions and algorithms can be integrated or built upon each other in a modular design. For example, additional safety constraints can be implemented as a supervision on top of a well-trained planning module to force some emergency rules without changing the inner workings of the planner. This enables designing redundant but reliable architectures. Yet the modular architecture has its shortcomings like error propagation and over-complexity. Some mistakes from upstream modules, like missclassification from perception module, can be propagated through the whole pipeline, and finally lead to fatal failure.

As the very upstream module, the perception module acts as a mediator between the low-level sensor input and the high-level decision module. It aims at establishing an understanding of the scene, where object detection is one of its primary tasks. The perception

module receives and processes various raw sensor inputs, which are then utilized by the localization and mapping module (Fig. 2.2).

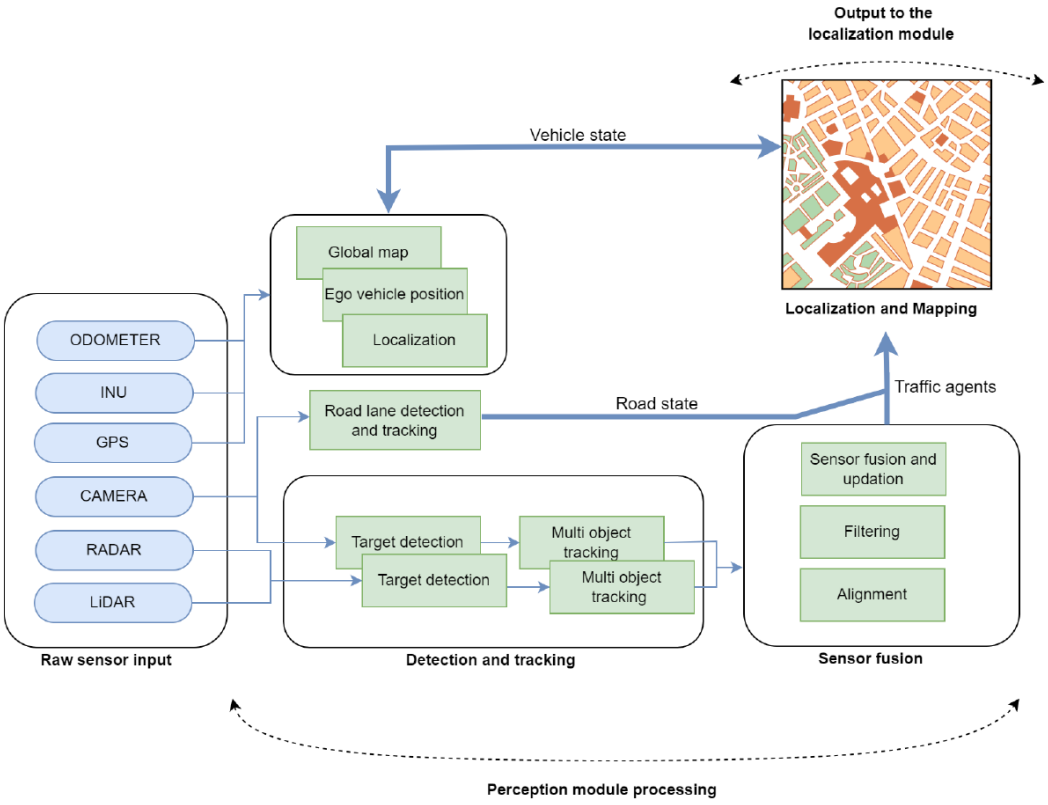


Figure 2.2: The perception module [CS23]

2.2 Sensorics for Perception

The recent mature ADSs utilizes a wide selection of onboard sensors. High sensor redundancy is needed in most of the tasks for robustness and reliability. Hardware units can be categorized into five: exteroceptive sensors for perception, proprioceptive sensors for internal vehicle state monitoring tasks, communication arrays, actuators, and computational units. In this section we focus on the sensors for perception, including camera, Radar and LiDAR.

Cameras can sense color, and as passive sensor they do not emit any signal for measurements. With the respect of perception for ADS, camera has a various of categories: Stereo camera can get a disparity map from left- and right-eye camera for a reliable depth estimation; Omnidirectional camera can provide a 360° 2D vision and can be used as an alternative to camera arrays. The panoramic view is particularly desirable for applications such as navigation, localization and mapping; Event cameras record data asynchronously for individual pixels with respect to visual stimulus. The output is therefore an irregular sequence of data points, or events triggered by changes in brightness. Sensing color is extremely important for tasks such as recognition of traffic lights and traffic signs. Furthermore, 2D computer vision is an established field with remarkable state-of-the-art algorithms. However, cameras have certain shortcomings. Illumination conditions affect their performance drastically, and depth information is difficult to obtain from a single camera. To deal with these issues, there are promising studies to improve camera based depth estimation, and studies about image

restoration against bad illumination and weather conditions. Yet the real-time capability is very essential for dynamic driving tasks.

Radar, LiDAR and ultrasonic sensors are very useful in covering the shortcomings of cameras. Depth information, i.e. distance to objects, can be measured effectively to retrieve 3D information with these sensors, and they are not affected by illumination conditions. However, emissions from active sensors can interfere with other systems. Radars emit radio waves that bounce back from objects and measure the time of each bounce. Radar is a well-established technology that is both lightweight and cost-effective. Radars are cheaper and can detect objects at longer distances than LiDARs, but the latter are more accurate.

LiDAR (Light Detection and Ranging) operates with a similar principle that of radar. But instead of radio waves, LiDAR emits pulses of infrared beams or laser light, which reflect off target objects. The result is a 3D representation of the surroundings in the form of point clouds. It has much higher resolution than radar within a range of 200 meters. Weather conditions such as fog or snow have a negative impact on the performance of LiDAR. Another aspect is the sensor size: smaller sensors are preferred on the vehicle because of limited space and aerodynamic restraints. LiDAR are generally larger than radars, which leads to a shortcoming in the perspective of packaging. A comparison of sensor properties is summarized as Fig. 2.3.

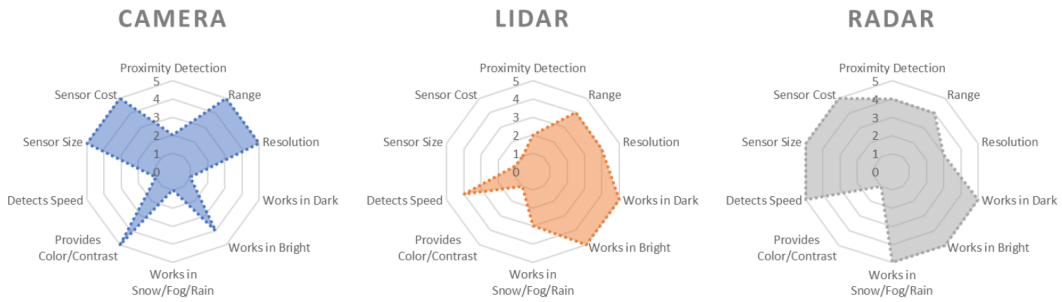


Figure 2.3: Comparison of sensor properties.

2.3 Deep Learning

Deep Learning (DL) is one subfield of Machine Learning (ML). Unlike the rule-based systems, where the calculation mechanism is programmed explicitly, machine learning system learns to infer pattern through a large amount of data implicitly. The model is initialized with learnable internal parameters. These unfixed parameters, usually known as weights, can be updated iteratively to minimize a loss function quantifying prediction errors between model's current prediction and the ground truth labels. The gradients of loss function with respect to model's input leads the update of weights towards a convergence, which is known as gradient descent. This entire weights-updating process is called backpropagation, and acts as the engine of ML.

The basic building block of an ML system, a learnable single-layer perceptron, is theoretically capable of approximating any continuous function to any desired degree, as proven by the Universal Approximation Theorem [Cyb88]. However, in practice stacking perception layers has empirically shown better performance than just using a single-layer. A model with many stacked layers is called a Multi-layer Perceptron (MLP). MLPs can be further organized to build more complicated deep neural networks, i.e. the core of deep learning models. Moreover, the functional blocks in deep learning models are not restricted to MLP. For specific tasks

there are various forms of blocks like Convolutional Neural Networks (CNNs) for computer vision, Transformers [Vas+23] for tasks with series input like natural language processing (NLP) or video understanding, Graph Neural Networks (GNNs) for social network analysis and for molecular biology and chemistry, and the recent State Space Models (SSMs) also for series input but with a linear complexity, etc. It's important to note that the corresponding tasks mentioned here only explained the initial motivation of the models' innovation, but not the limitation of their applications. Considering the relevance with this work, this section will cover three of them: CNN for the understanding of generic detectors, GNN for relation inference and SSM for model complexity reduction.

2.3.1 Convolutional Neural Network

Convolutional neural networks (CNNs) are powerful tools for understanding spatial hierarchies and local contexts in structured data such as images. In the early machine learning era the human-designed filters are applied to images to extract specific features, E.g. the Sobel operator for edge detection. These carefully-designed kernels are hard to generalize to various and more complicated tasks and inputs. In CNN, however, these fixed kernels are turned to be learnable by allowing the model to update their internal weights. This makes the model to be end-to-end trainable and thus increase the adaptability and performance on a wide range of applications.

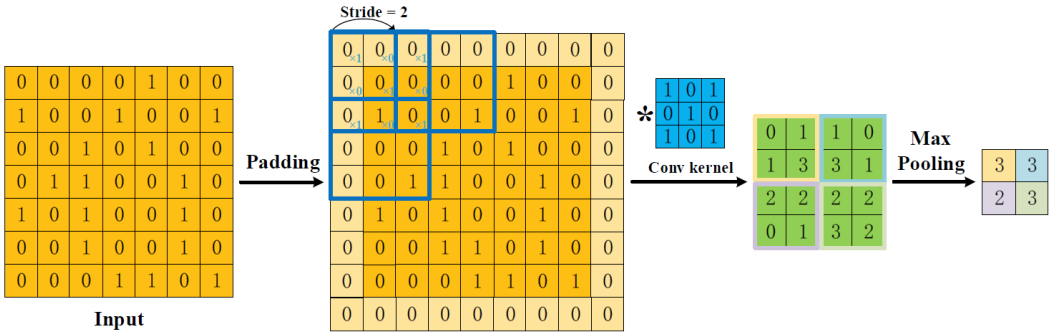


Figure 2.4: Procedure of a two-dimensional CNN. [Li+20]

As shown in Fig. 2.4, four components are typically needed. Convolution is a pivotal step for feature extraction. The outputs of convolution can be called feature maps. When setting a convolution kernel with a certain size, we will lose information in the border. Hence, padding is introduced to enlarge the input with zero value, which can adjust the size indirectly. Besides, for the sake of controlling the density of convolving, stride is employed. The larger the stride, the lower the density. After convolution, feature maps consist of a large number of features that is prone to causing overfitting problem. As a result, pooling (a.k.a. downsampling) is proposed to obviate redundancy, including max pooling and average pooling.

The concept of the receptive field is crucial for understanding the behavior of convolutional layers in a CNN. The receptive field refers to the spatial extent of the input that influences a particular output neuron (Fig. 2.5). As mentioned before, for a single convolutional layer, the size of the receptive field is equal to the size of the convolutional kernel. However, as we stack multiple convolutional layers, the receptive field for neurons in deeper layers grows with the stride of the convolutions, encompassing a larger portion of the input image. In Eq. 2.1 the receptive field in the previous layer $L-1$ could be calculated by the network's parameters kernel size K and stride S based on the receptive field size in current layer L:

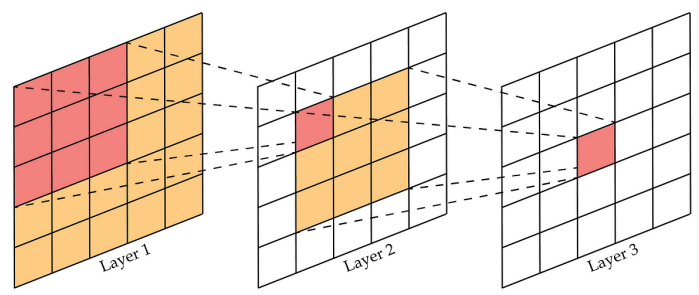
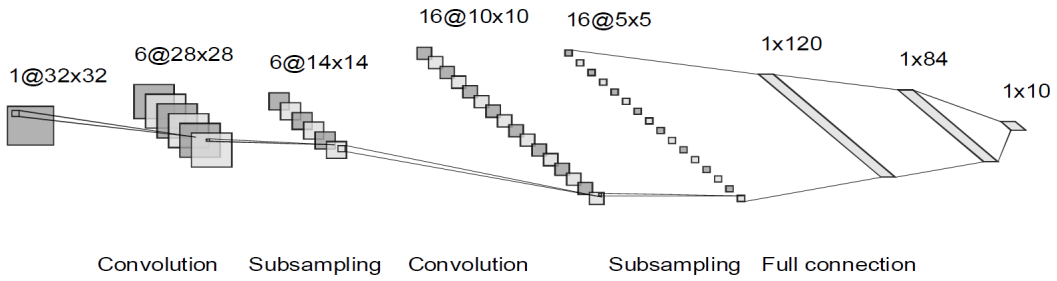


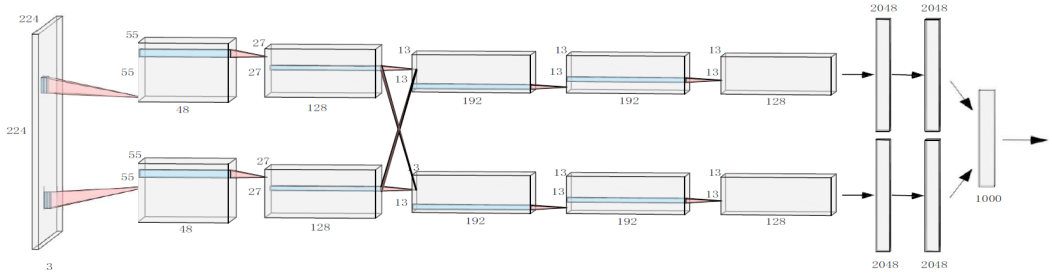
Figure 2.5: Receptive fields through CNN layers.

$$RF^{(L-1)} = RF^{L} + (K-1) \times S_L \tag{2.1}$$

In a typical modern CNN architecture, several layers of convolutions are stacked and connected with non-linear activation functions. The first layers learn low-level features like edges and corners, while the deeper layers learn more complex features like shapes and objects. The final layers are fully connected layers that use the learned features to compute the final output. LeNet [Lec+98] (Fig. 2.6a) was exactly built in this pattern and showed its utility for recognizing handwriting characters and reading bank checks. In the year 2012 the AlexNet [KSH12] (Fig. 2.6b won the championship in the ImageNet 2012 competition, which was regarded as one of the breaking-through milestone in computer vision.



(a) Architecture of LeNet-5 [Lec+98].



(b) Architecture of AlexNet [KSH12].

Figure 2.6: Some early CNN-based architectures. [Li+20]

The early CNN-based models are designed for basic tasks, such as handwritten digits recognition and character categories classification. The distinctive characteristics of CNNs are encapsulated in the convolution kernels, which employ receptive fields to capture visual information of interest from images. With the aid of powerful computing devices (GPU) and large-scale datasets, increasingly deeper and efficient models have been proposed to enhance performance across a spectrum of visual tasks.

When 3D scenes like point clouds are turned into forms with regular spatial structure, like voxels, it can be processed with convolution as well. Considering a large number of empty

voxels are contained due to point clouds' sparsity nature, it's not proper to perform convolutions directly over the entire sparse voxel space in a naive way. 3D sparse convolutional networks significantly improve efficiency by performing computations only on non-empty voxels.

2.3.2 Transformers

Transformers [Vas+23] was innovated to tackle the issues of CNN/RNN/GNN only capturing local relations when dealing with the long-range feature representations. The core operation is the self-attention mechanism which transforms the input tokens into query, key, and value features, and outputs the long-range features by multiplying the similarity matrix (obtained via product between the query and key features) with the value features. Since it was proposed in the year 2017, Transformer architecture first swept the NLP community. Then, other communities are also boosted with such networks, for example, the ViT [Dos+21] and Swin-Transformer [Liu+21] released in computer vision. Many researchers also exploit the hybrid network architectures by combining Transformer and other networks, or adapting the Transformer for multi-modal research problems.

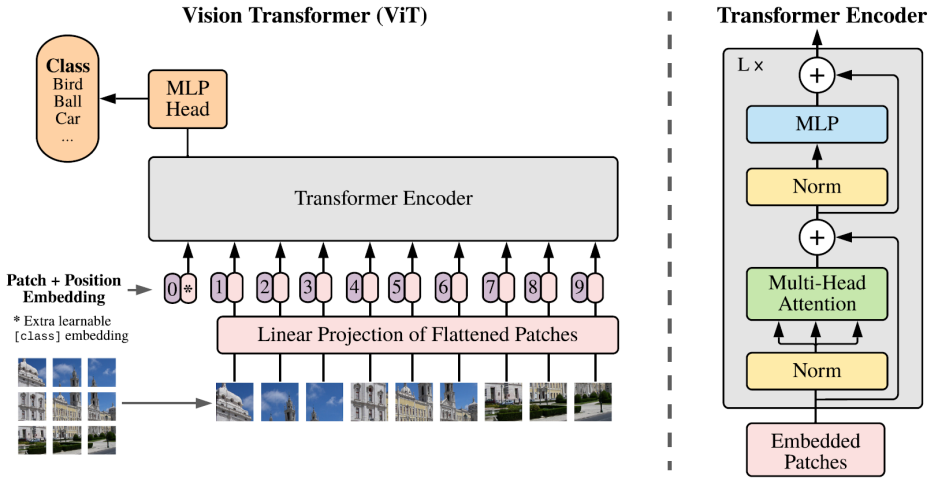


Figure 2.7: Architecture of ViT [Dos+21]: ViT splits an image into fixed-size patches, linearly embeds each of them, adds position embeddings, and feeds the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, ViT uses the standard approach of adding an extra learnable "classification token" to the sequence.

The Vision Transformer (ViT, Fig. 2.7) has spread Transformers' application to CV field and already exhibit superior performance over CNNs. This superiority can be attributed to global receptive fields and dynamic weights facilitated by the attention mechanism of Transformer. However, the attention mechanism requires quadratic complexity in terms of image sizes, resulting in expensive computational overhead when addressing downstream dense prediction tasks, such as object detection, semantic segmentation, etc. To tackle this issue, substantial effort has been dedicated to improving the efficiency of attention by constraining the size or stride of computing windows, albeit at the cost of imposing restrictions on the scale of receptive fields.

Attention-based transformers have revolutionized natural language processing and other sequence-to-sequence tasks. However, they encounter certain limitations, especially when dealing with long input sequences, especially when dependencies extend beyond the attention window size. This constraint is particularly crucial in applications such as high-resolution

imagery analysis and genomics. In summary, transformer-based vision methods face limitations *1) Computational Complexity*: Transformers exhibit high computational demands, particularly with large models. This complexity poses challenges for both training and deploying them on resource-constrained devices. *2) Large Memory Requirements*: Transformers necessitate significant memory resources for storing embeddings and intermediate activations. This can hinder scalability, especially for very long sequences, as they may surpass available memory capacity. *3) Fixed Sequence Length*: Transformers rely on fixed-size input sequences due to positional embeddings. Efficiently handling variable-length inputs presents a notable challenge in transformer-based architectures. *4) Attention Mechanism Scalability*: While attention is a powerful mechanism, it has a quadratic scaling with input sequence length. This makes it less efficient for very long sequences, and *5) Lack of Causality in Standard Attention*: The standard self-attention mechanism used in transformers doesn't inherently capture causality. It treats all positions equally, which can be problematic for tasks where causality matters.

2.3.3 From Transformers to State Space Models

State Space Models (SSMs) are recently proposed models that are introduced into deep learning as state space transforming. In this work we use SSMs to replace some 2D CNNs in the baseline models to reduce model complexity and improve calculation efficiency, and maintain the detection accuracy meanwhile. Inspired by continuous state space models in control systems, SSM aims at handling long range dependency problems, which is very similar to the targeting tasks of Transformers [Vas+23].

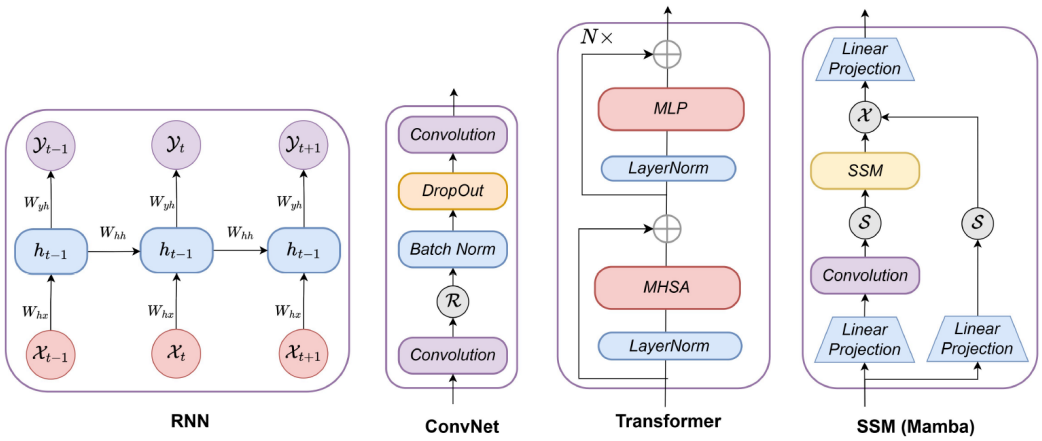


Figure 2.8: The evolutionary progression of sequential data modeling paradigms [PA24], from Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs) to Transformer models and State-Space Models (SSMs), highlighting advancements in capturing temporal dynamics, spatial hierarchies, and complex system interactions.

As explained in section 2.3.2, the attention mechanism requires quadratic complexity in terms of input sizes. To further decrease the computing cost, while capturing long-range dependency and maintaining high performance, many new sparse attention based models or new neural network paradigms are proposed. Among them, State Space Model (e.g., S4 [GGR22], S4nd [Ngu+22], Mamba(S6) [GD23]) becomes the center of attention, and was agilely adopted to computer vision area.

The State Space Model (SSM) is a framework initially proposed to model a dynamic system using state variables. When adapting this concept for deep learning, we usually refer to linear invariant (or stationary) systems. The original SSM is a continuous-dynamic system

that can be discretized for recurrent and convolutional views for the computer to handle. SSMs can be adopted for various data processing and feature learning, including image/video data, text data, structured graph data, event streams/point cloud data, multi-modal/multi-media data, audio and speech, time series data, tabular data, etc.

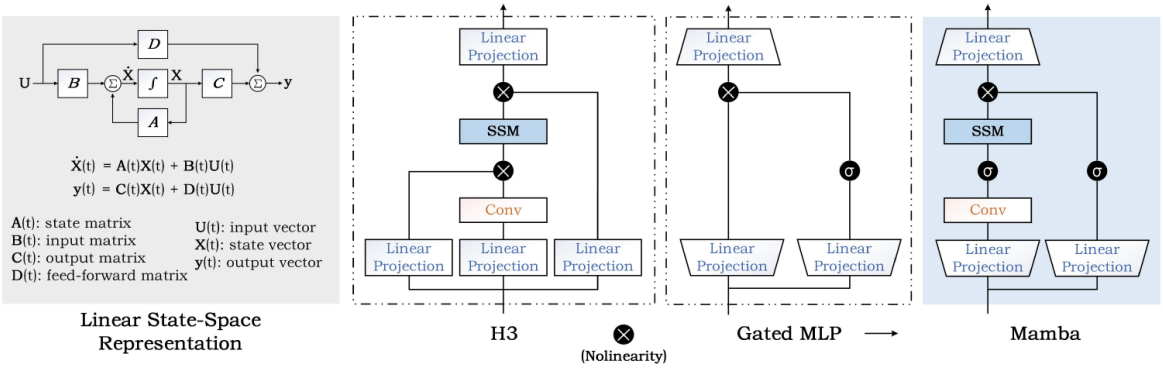


Figure 2.9: Block diagram representation of the linear state-space equations [Wan+24b]

State Space Model (SSM) originates from the classic Kalman filter, as illustrated in Fig.2.9, it takes the $1-D$ input signal (or, control signal) $x(t)$ and maps it into an $N-D$ latent state vector $\mathbf{h}(t)$, then, it projects into a $1-D$ output signal $y(t)$. The general mapping of $x(t) \in (R) \mapsto y(t) \in (R)$ through a latent state $\mathbf{h}(t) \in \mathbb{R}^N$ can be defined as Eq.2.2:

$$\begin{aligned} \dot{\mathbf{h}}(t) &= \mathbf{A} \cdot \mathbf{h}(t) + \mathbf{B} \cdot x(t) \\ y(t) &= \mathbf{C} \cdot \mathbf{h}(t) + d \cdot x(t) \end{aligned} \tag{2.2}$$

where $\dot{\mathbf{h}}(t) = \frac{}{dt}\mathbf{h}(t)$. $\mathbf{A} \in \mathbb{R}^{N \times N}$, $\mathbf{B} \in \mathbb{R}^{N \times 1}$, $\mathbf{C} \in \mathbb{R}^{1 \times N}$ and $d \in \mathbb{R}$ represents state matrix, input vector, output vector and feed-forward coefficient. As the raw system is continuous, we need to first discretize them before feeding the computer. Besides when there is no direct feedthrough (like skip connection) in the system model, d equals to zero. Thus, the discretized state space equation omitting d could be represented as:

$$\begin{aligned} \mathbf{h}_t &= \overline{\mathbf{A}} \cdot \mathbf{h}_{t-1} + \overline{\mathbf{B}} \cdot \mathbf{x}_t \\ \mathbf{y}_t &= \overline{\mathbf{C}} \cdot \mathbf{h}_t \end{aligned} \tag{2.3}$$

where the *discrete parameters* $(\overline{\mathbf{A}}, \overline{\mathbf{B}})$ are obtained from *continuous parameters* $(\Delta, \mathbf{A}, \mathbf{B})$, Δ denoting the step size, under the zero-order hold (ZOH) discretization rule:

$$\begin{aligned} \overline{\mathbf{A}} &= \exp(\Delta \mathbf{A}) \\ \overline{\mathbf{B}} &= (\Delta \mathbf{A})^{-1}(\exp(\Delta \mathbf{A}) - \mathbf{I}) \cdot \Delta \mathbf{B} \\ \overline{\mathbf{C}} &= \mathbf{C} \end{aligned} \tag{2.4}$$

After the parameters have been transformed $(\Delta, \mathbf{A}, \mathbf{B}, \mathbf{C}) \mapsto (\overline{\mathbf{A}}, \overline{\mathbf{B}}, \overline{\mathbf{C}})$, the model can be computed in two ways, either as a linear recurrence or a global convolution. Commonly, the model uses the convolutional mode for efficient parallelizable training, where the whole input sequence is seen ahead of time. By simply expanding the Eq.2.3 we can regard the multipliers of $\mathbf{x}^{(i)} = \{x0, x_1, ..., x_i\} \mapsto y_i$ as the convolutional kernel $\overline{\mathbf{K}} = \mathbf{C}\overline{\mathbf{B}} \cdot (\overline{\mathbf{A}}^0, \overline{\mathbf{A}}^1, \overline{\mathbf{A}}^2, ..., \overline{\mathbf{A}}^L)$, where L is the length of given input sequence. With this, we can get the convolutional formulation:

$$\begin{aligned} \overline{\mathbf{K}} &= (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}\mathbf{B}}, ..., \mathbf{C}\overline{\mathbf{A}}^k\overline{\mathbf{B}}...) \\ \mathbf{y} &= \mathbf{x} * \overline{\mathbf{K}} \end{aligned} \tag{2.5}$$

At this moment, we get the complete SSM model that can realize the parallelism of training and is suitable for the recurrent form of linear complexity of inference. State Space Models have emerged as compelling alternatives to transformers, particularly for processing long sequences. SSMs can be conceptualized as RNNs with fixed lengths, which do not grow with input length This brings significant efficiency benefits in terms of inference speed and computation/memory complexity compared to transformers. However, despite their efficiency advantages, SSMs often fall short of the performance gap with state-of-the-art transformers in certain data modalities, notably in vision tasks. The related research of Mamba's applications in CV would be discussed in details in section 3.3.

In the Transformer architecture, the context information is stored in the similarity matrix, however, the SSM doesn't have a similar module which makes it perform poorly in contextual learning. To address this issue, Gu et al. propose the Mamba (S6) [GD23] architecture, which involves *1). Selective Scan Operator* allowing the model to filter relevant information out and *2). Hardware-aware Algorithm* allowing efficient storage of (intermediate) results through parallel scanning, kernel fusion, and recalculation.

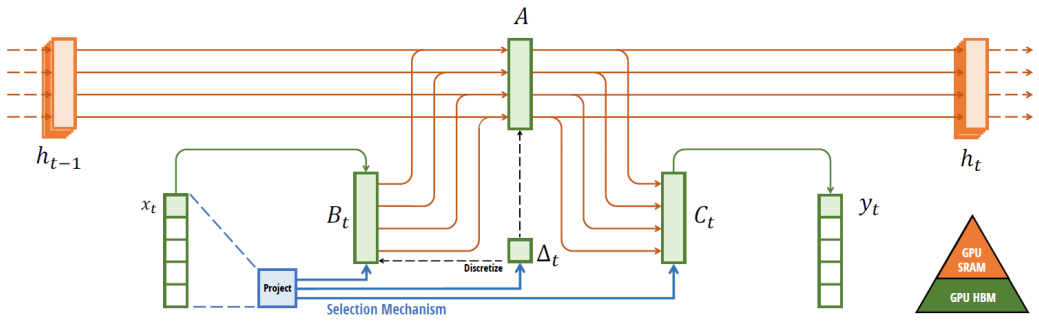


Figure 2.10: Overall architecture of Mamba (S6) [GD23] Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g.$N = 4$). Prior SSMs avoid materializing this large effiective state (DN, times batch size B and sequence length L) through clever alternate computation paths requiring time-invariance: the $(\Delta, \mathbf{A}, \mathbf{B}, \mathbf{C})$ parameters are constant across time. The selection mechanism adds back input-dependent dynamics, which also requires a careful hardware-aware algorithm to only materialize the expanded states in more efficient levels of the GPU memory hierarchy.

Mamba (S6): Selective State Space Model Mamba [GD23] highlights the quadratic computational and memory complexity of transformers, along with the perplexity gap that traditional state space models (SSMs) exhibit in comparison. Prior to Mamba, SSMs faced inefficiencies in addressing tasks such as selective copying and induction head. However, observations on Liquid-S4's input-dependent state-transition module suggest its potential for solving these tasks effectively. Mamba addresses these challenges by introducing a novel parametrization approach for SSMs based on input characteristics and incorporating a simple selection mechanism. Additionally, Mamba presents an efficient hardware-aware algorithm based on selective scan (Fig. 2.10). Similar to Gated State Spaces (GSS), Mamba employs a gated technique to reduce the dimensionality of global kernel operations. Moreover, Mamba combines a gated MLP with the SSM module to further enhance its capabilities. Mamba's linear-time complexity makes it more efficient than traditional Transformers. The core algorithms used in Mamba are illustrated in Alg. 1 and Alg. 2. The main difference is simply making several parameters Δ, $\mathbf{B}$, $\mathbf{C}$ functions of the input, along with the associated changes to tensor shapes throughout. This implies that S6 is aware of the contextual information embedded in the input, ensuring the dynamism of weights within this mechanism.

Algorithm 1 SSM (S4) [GGR22]

Require: $x : (\mathrm{B}, \mathrm{L}, \mathrm{D})$
Ensure: $y : (\mathrm{B}, \mathrm{L}, \mathrm{D})$
1: $\mathbf{A} : (\mathrm{D}, \mathrm{N}) \leftarrow$ Parameter
2: $\mathbf{B} : (\mathrm{D}, \mathrm{N}) \leftarrow$ Parameter
3: $\mathbf{C} : (\mathrm{D}, \mathrm{N}) \leftarrow$ Parameter
4: $\mathbf{\Delta} : (\mathrm{D}) \leftarrow \tau_\Delta(\text{Parameter})$
5: $\overline{\mathbf{A}}, \overline{\mathbf{B}} : (\mathrm{D}, \mathrm{N}) \leftarrow \text{discretize}(\Delta, \mathbf{A}, \mathbf{B})$
6: $y \leftarrow \text{SSM}(\overline{\mathbf{A}}, \overline{\mathbf{B}}, \mathbf{C})(x)$ ▷ Time-invariant
return y

Algorithm 2 SSM + Selection (S6) [GD23]

Require: $x : (\mathrm{B}, \mathrm{L}, \mathrm{D})$
Ensure: $y : (\mathrm{B}, \mathrm{L}, \mathrm{D})$
1: $\mathbf{A} : (\mathrm{D}, \mathrm{N}) \leftarrow$ Parameter
2: $\mathbf{B} : (\mathrm{B}, \mathrm{L}, \mathrm{N}) \leftarrow s_B(x)$
3: $\mathbf{C} : (\mathrm{B}, \mathrm{L}, \mathrm{N}) \leftarrow s_C(x)$
4: $\mathbf{\Delta} : (\mathrm{B}, \mathrm{L}, \mathrm{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
5: $\overline{\mathbf{A}}, \overline{\mathbf{B}} : (\mathrm{B}, \mathrm{L}, \mathrm{D}, \mathrm{N}) \leftarrow \text{discretize}(\Delta, \mathbf{A}, \mathbf{B})$
6: $y \leftarrow \text{SSM}(\overline{\mathbf{A}}, \overline{\mathbf{B}}, \mathbf{C})(x)$ ▷ Time-varying
return y

S4nd: Structured State Space Sequence for Images and Videos S4nd [Ngu+22] extends the applicability of SSMs beyond sequential data (such as Text data and time series) to continuous data domains such as images or videos. S4nd contributes a new deep learning layer that transforms a standard SSM, which typically represents a 1-dimensional Ordinary Differential Equation (ODE), into a multi-dimensional Partial Differential Equation (PDE). This transformation allows SSMs to capture spatial dependencies across multiple dimensions. Moreover, S4nd demonstrates that a multi-dimensional SSM can be equivalently represented as ND continuous convolution, wherein each dimension undergoes 1D SSM convolution. To evaluate its efficacy, S4nd was tested on the ImageNet dataset using ConvNeXt as the backbone model, resulting in improved performance over traditional methods. Notably, it improved the performance of Vision Transformers (ViTs). S4nd extends the capabilities of SSMs to handle continuous data, bridging the gap between sequential and spatial modeling.

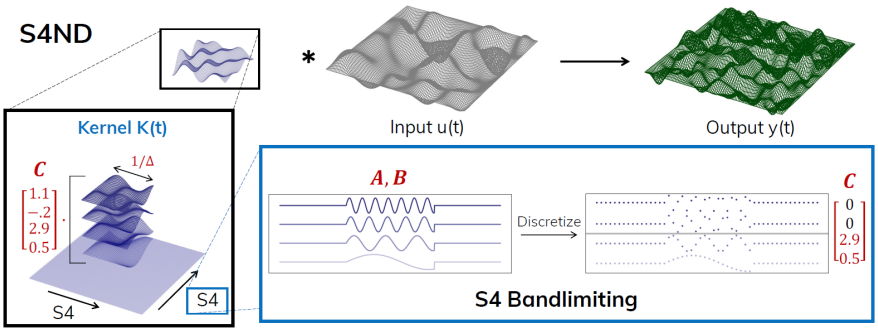


Figure 2.11: Overall architecture of S4nd [Ngu+22] *(Top)* S4ND can be viewed as a depthwise convolution that maps a multidimensional input *(black)* to output *(green)* through a continuous convolution kernel *(blue)*. *(Bottom Left)* The kernel can be interpreted as a linear combination (controlled by $\mathbf{C}$) of basis functions (controlled by $\mathbf{A}$, $\mathbf{B}$) with flexible width (controlled by step size Δ). For structured $\mathbf{C}$, the kernel can further factored as a low-rank tensor product of 1D kernels, and can be interpreted as independent S4 transformations on each dimension. *(Bottom Right)* Choosing $\mathbf{A}$, $\mathbf{B}$ appropriately yields Fourier basis functions with controllable frequencies. To avoid aliasing in the final discrete kernels, the coefficients of $\mathbf{C}$ corresponding to high frequencies can simply be masked out.

2.3.4 Graph Neural Network

Graph Neural Network (GNNs) have emerged as a powerful tool for learning from graph-structured data, extending traditional neural network architectures to handle irregular data formats. The complexity of graph data has imposed significant challenges on the existing machine learning algorithms. As graphs can be irregular, a graph may have a variable size of unordered nodes, and nodes from a graph may have a different number of neighbors,

resulting in some important operations (e.g., convolutions) being easy to compute in the image domain but difficult to apply to the graph domain. Furthermore, a core assumption of existing machine learning algorithms is that instances are independent of each other. This assumption no longer holds for graph data because each instance (node) is related to others by links of various types, such as citations, friendships, and interactions.

GNNs operate by iteratively aggregating information from a node's local neighborhood, allowing for the incorporation of both node and edge attributes in the learning process. This enables GNNs to perform well in tasks such as node classification, link prediction, and graph classification. GNNs have shown promise across various domains, including social network analysis, bioinformatics, and computer vision, demonstrating their utility in capturing complex relationships within data. In this work we utilize GNN to extract object relation features with *message passing* and *edge convolution*.

Formally, a GNN operates on a graph $G = (V, E)$ defined by its vertex set V and its edge set E [Wu+20]. An edge pointing from the node i to j can be illustrated as $(i, j) \in E$, where $j \in \mathcal{N}(i)$ is the index of a certain node in node i's neighboring set $\mathcal{N}(i)$.

Generally, a process of updating node features via a graph neural network can be demonstrated as Eq.2.6:

$$\mathbf{v}_i^{l+1} = f(\mathbf{e}_{ij}^l, \mathbf{v}_i^l) \tag{2.6}$$

where $l \in L$ is the lth layer of the GNN with L layers in total, $\mathbf{v}_i$ is the feature of the ith node, $\mathbf{e}_{ij}$ is the feature of edge (i, j), and $f(.)$ is the operation applying edge features to update the node features. Usually, edge feature $\mathbf{e}_{ij}^l$ can be defined as $\mathbf{e}_{ij}^l = h^l(\mathbf{v}_i^l, \mathbf{v}_j^l)$, where $h^l(.)$ is a function to capture the edge feature. This function can be implemented as any permutation invariant function, such as max or sum.

2.4 Object Detection

Deep learning have substantially boosted the research in computer vision (CV). Covering a wide range of applications, the major task in CV is to extract, analyse and understand the visual content from images and other representations of scenarios in a broad sense, like point clouds. In context of autonomous driving, these tasks can be specifically categorised into object detection, (multi-) object tracking, optical flow, and semantic segmentation, etc. As for object detection, the main categories are 2D object detection working on images from cameras and 3D object detection working on point clouds from RGB-D camera, Radar or LiDAR sensors. Tab.2.1 outlines the inputs and outputs for 2D and 3D object detection systems.

Table 2.1: Comparison of the inputs and outputs of 2D and 3D object detection.

	2D	3D
Device	Camera	LiDAR, Radar, RGB-D Camera
Inputs	Image $\in \mathbb{R}^{H \times W \times 3}$	Point clouds $\in \mathbb{R}^{N_{pts} \times 4}$
Outputs	$N_{objs} \times 4$	$N_{objs} \times 7$
(Bounding boxes)	$[x_c, y_c, h, w]$	$[x_c, y_c, z_c, h, w, l, \theta]$

2D detectors focus on a grid of pixels arranged in a two-dimensional plane, or simply, an image. Each pixel can have either a single value for its grayscale intensity or multiple values for its color components, e.g. RGB values. For depth information, radar, or LiDAR sensors come into play, adding a third dimension to the point cloud and possibly a value

for reflectivity. The output for both 2D and 3D systems consists of a set of detected objects, including bounding boxes and classification scores. Class assignment is treated as a classification problem, providing a confidence score for each of the C possible classes. Determining the bounding boxes is generally approached as a regression task where a 2D bounding box uses four parameters, while a 3D one requires seven [Zou+23].

As shown in Fig. 2.12, in the past two decades object detection has generally gone through two historical periods: *traditional object detection period* (before 2014) and *deep learning based detection period* (after 2014). In regard of this work we would focus on the deep learning-based period.

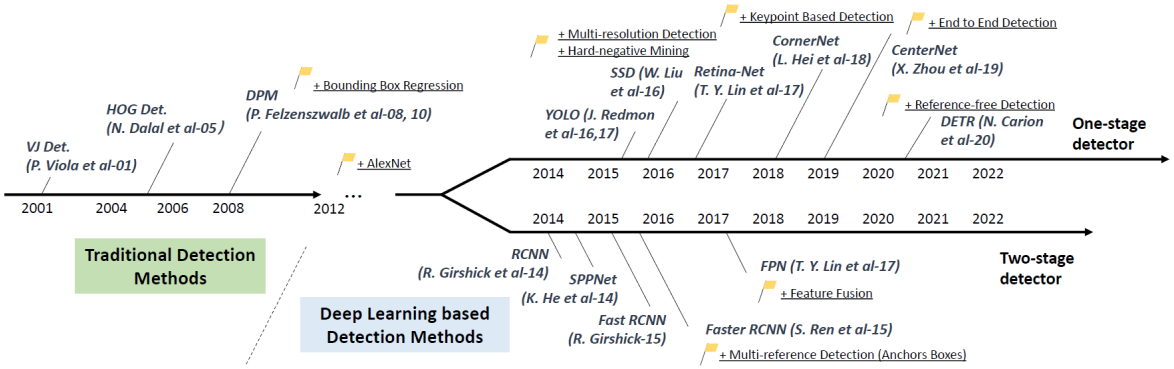


Figure 2.12: A road map of object detection [Zou+23]

Inspired by the application of a deep convolutional network is learning robust and high-level feature representations of an image, R. Girshick et al. took the lead to break the deadlocks in 2014 by proposing the Regions with CNN features (RCNN) [Gir+14]. Since then, object detection started to evolve at an unprecedented speed. There are two groups of detectors in the deep learning era: *two-stage detectors* and *one-stage detectors*, where the former frames the detection as a "coarse-to- fine" process, while the latter frames it as to "complete in one step".

2.4.1 Two-stage detector

RCNN [Gir+14] was one of the most significant two-stage detectors. As shown in Fig. 2.13, it starts with the extraction of a set of object proposals (object candidate boxes) by selective search [Uij+13]. Then each proposal is rescaled to a fixed size image and fed into a CNN model pretrained on ImageNet (say, AlexNet [KSH12]) to extract features. Finally, linear SVM classifiers are used to predict the presence of an object within each region and to recognize object categories.

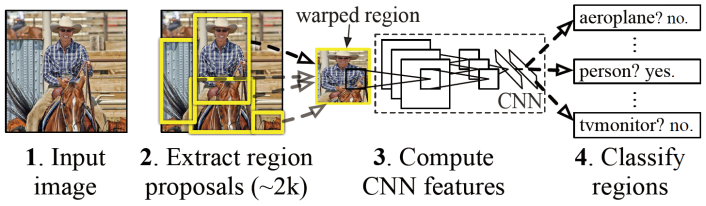


Figure 2.13: R-CNN: Regions with CNN features [Gir+14]

This was a significant move towards precise object detection, but R-CNN had a slow processing speed due to the fact that the CNN had to be applied to each proposed region sepa-

rately. Identifying the inefficiency in R-CNN, Fast R-CNN [Gir15] was designed to address this by introducing a technique known as region of interest (RoI) pooling. This allowed sharing the computation over the whole image, significantly reducing the time needed for processing an image. However, Fast R-CNN was still using the selective search, which continued to be a bottleneck in terms of speed. The Faster R-CNN [Ren+16] architecture was introduced to solve this issue. Here, a Region Proposal Network (RPN) was integrated to replace the selective search for generating region proposals. The RPN and the detection network share the convolutional features, accelerating the processing. This was a major step towards real-time object detection.

2.4.2 One-stage detector

One-stage detectors are aiming for real-time performance while still achieving competitive performance. YOLO [Red+16] (You-Only-Look-Once) was proposed by R. Joseph et al. in 2015. It was the first one-stage detector in the deep learning era. In contrast to two-stage detectors, YOLO applies a single neural network to the entire image in a single pass to realize both localize and classify for objects. As shown in Fig. 2.14, after dividing the image into grid, for each single grid cell the network predicts bounding boxes, corresponding confidence scores indicating the possibilities of containing an object, and the class probabilities simultaneously.

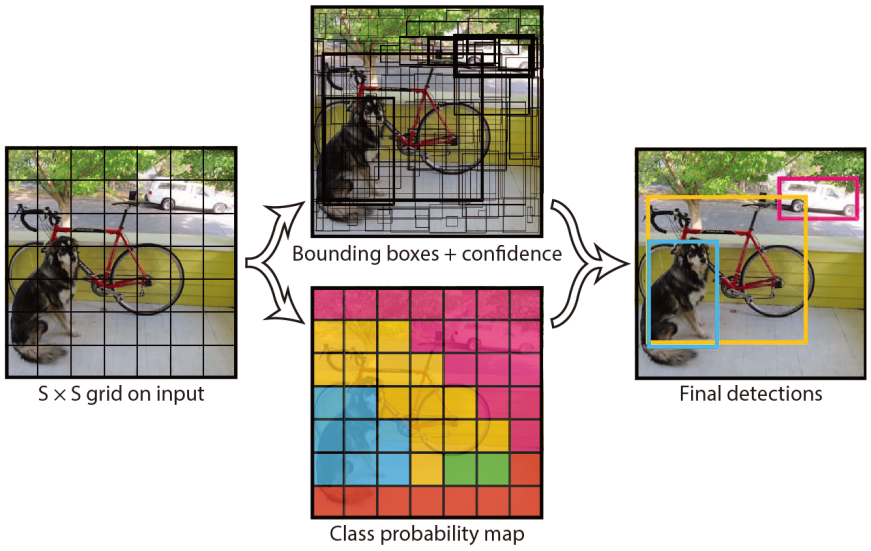


Figure 2.14: YOLO [Red+16] divides the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, corresponding confidence and C class probabilities. These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.

In spite of its great improvement of detection speed, YOLO suffers from a drop of localization accuracy compared with two-stage detectors, especially for some small objects. YOLO's subsequent versions [RF16; RF18; BWL20; WBL22] and the latter proposed SSD [Liu+16] has paid more attention to this problem. The newest YOLOv10 [Wan+24a] presents the consistent dual assignments for NMS-free training of YOLOs, which brings competitive performance and low inference latency simultaneously, and introduces the holistic efficiency-accuracy driven model design strategy for YOLOs.

2.5 3D Object Detection Approaches

2.5.1 Point-based

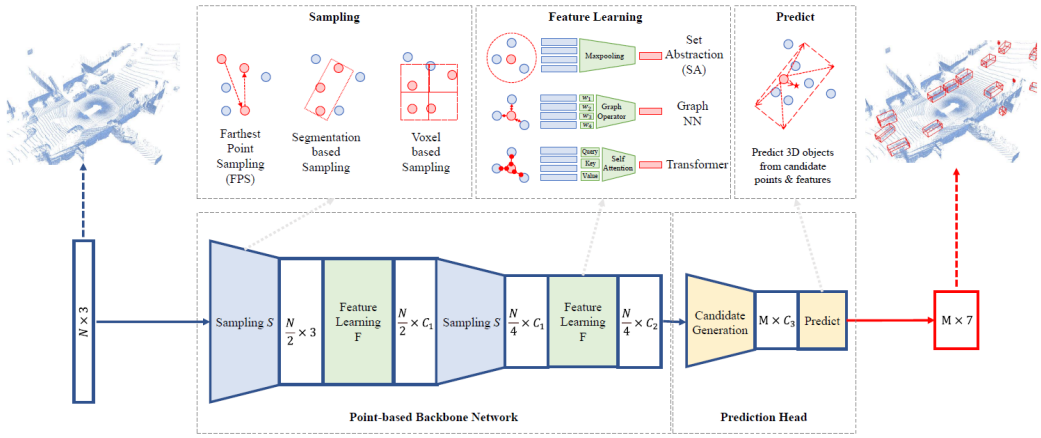


Figure 2.15: An illustration of point-based 3D object detection method [Mao+23]

Point-based approaches are adapted to deal with the point clouds of the natures unstructured and unordered. A first example of this kind of approach was PointNet [Qi+17a], which employs a shared multi-layer perceptron (MLP) network architecture to extract features from each input point individually. By applying the same set of weights to each point independently, PointNet learns point-wise features that capture local geometric information regardless of the point's position in the input point cloud. In addition to point-wise feature learning, PointNet incorporates a max-pooling layer to aggregate the point-wise features into a global feature vector representing the entire point cloud. This global feature vector captures holistic information about the entire shape or object, enabling PointNet to make predictions based on the overall structure of the point cloud.

Following, PointNet++ [Qi+17b] refined the concept, introducing a hierarchical neural network to capture fine-grained structure, alongside global context. These architectures work by extracting features from individual points and their neighborhood, progressively obtaining a global feature descriptor that encapsulates information about the entire point cloud (Fig. 2.15). The key to point-based methods is their ability to keep information about the initial point cloud, making them capable of obtaining precise detections. The shortcoming of point-based methods is the high computational intensity.

2.5.2 Voxel-based

Voxel-based approach could be categorized as one of the grid-based approach, which includes pillar-based and BEV-based approaches as well. Fig. 2.16 shows a general procedure of grid-based approachs.

To convert the unstructured point clouds into a structured format, the point cloud is rasterize into equally spaced 3D voxels. Points located inside a voxel are used by a *Voxel Feature Encoding (VFE)* layer to obtain a feature vector per voxel. The VFE layer aims to discretize the point cloud while keeping low-level information about it [ZT17]. By aggregating points within each voxel, these methods achieve a reduction in data complexity which is important for computational efficiency. Additionally, converting an unstructured point cloud into a structured voxel representation makes it possible to apply 3D convolutions to the data.

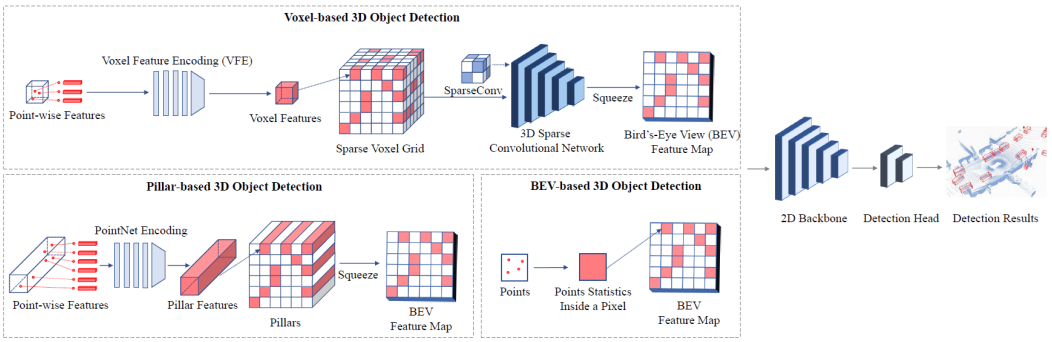


Figure 2.16: An illustration of grid-based 3D object detection methods [Mao+23]

Therefore the highly successful 2D convolutions can easily be extended to 3D perception. Section 2.3.1 explained the concept of sparse convolutions which were an adaptation to deal with the sparsity of 3D space. VoxelNet [ZT17] was an early architecture using several layers of 3D convolutions and achieving competitive results. While voxel-based methods lower the computational strain by structuring the point cloud data, they often neglect fine-grained information. The process of discretization can sometimes oversimplify the data, losing nuanced details crucial for precise detection [Den+21]. Fig. 2.17 shows the architecture of VoxelNet. The first step in this architecture is taking the unstructured data of the point cloud and converting it into discrete voxels. These voxels are subsequently processed using convolutions.

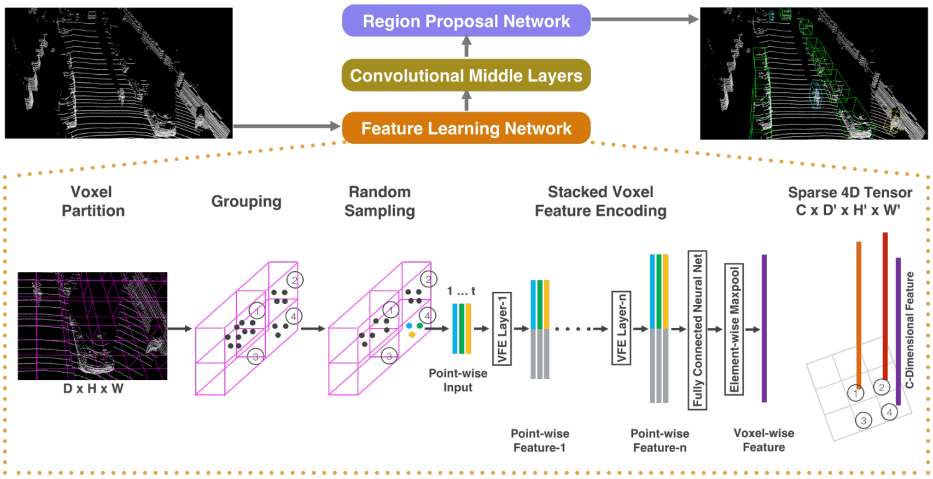


Figure 2.17: Architecture of Voxel-Net. [ZT17] The feature learning network takes a raw point cloud as input, partitions the space into voxels, and transforms points within each voxel to a vector representation characterizing the shape information. The space is represented as a sparse 4D tensor. The convolutional middle layers processes the 4D tensor to aggregate spatial context. Finally, a RPN generates the 3D detection.

2.5.3 Point-Voxel-based

Point-voxel-based approaches aim to combine the strengths of both point-based and voxelbased approaches for 3D object detection. The goal is to handle the sparsity and irregularity of point clouds while maintaining computational efficiency [NLH21]. Similarly to voxel-based methods, in this approach, the data is initially segmented into voxels. However, instead of simply aggregating the points in each voxel to obtain a feature vector per voxel, point-based methods like PointNet are applied to points within each voxel. Processing

the points in this way allows for more nuanced feature extraction, maintaining fine-grained information. The voxelized structure allows the use of 3D convolutions, just like in pure voxel-based approaches. The point-voxel based approaches could be further categorized into *one-stage* and *two-stage* methods, which will be discussed in section 3.1 Examples of architectures that use point-voxel-based approaches include PVF-Net [CZ20] and Hvpr [NLH21]. These architectures demonstrate improved performance over purely point-based or voxel-based methods, particularly in environments with varying point cloud density and complexity [NLH21]. Fig. 2.18 shows the architecture of HVPR. This model makes use of point-based methods as well as voxel-based methods to predict the bounding boxes of objects.

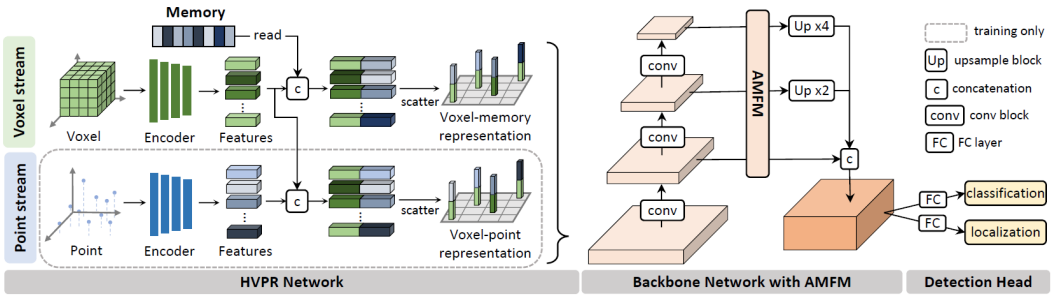


Figure 2.18: Architecture of HVPR. [NLH21] The HVPR network inputs point clouds and generates two types of hybrid 3D features via a two-stream encoder: Voxel-point and voxel-memory representations. The former representations are obtained by aggregating point-based features for individual voxel-based ones. For the later ones, we also perform the aggregation but with memory items, instead of using point-based features. That is, we augment the point-based features using a memory module, and exploit voxel-memory representations, i.e., hybrid 3D features, at test time for fast inference. The backbone network with AMFM inputs the voxel-memory representations to extract multiple scale-aware features, and the detection head predicts 3D bounding boxes and object classes.

PV-RCNN [Shi+21], one of the baseline models used in our work, is also a point-voxel based approach. PV-RCNN integrates both the voxel-based and the PointNet-based networks via a two-step strategy including the voxel-to-keypoint 3D scene encoding and the keypoint-to-grid RoI feature abstraction. The raw point clouds are first voxelized to feed into the 3D sparse convolution based encoder to learn multi-scale semantic features and generate 3D object proposals. Then the learned voxel-wise feature volumes at multiple neural layers are summarized into a small set of keypoints via the novel Voxel Set Abstraction (VSA) module. Finally the keypoint features are aggregated to the RoI-grid points to learn proposal specific features for fine-grained proposal refinement and confidence prediction. The overall framework of this Point-Voxel Feature Set Abstraction method is illustrated in Fig. 2.19.

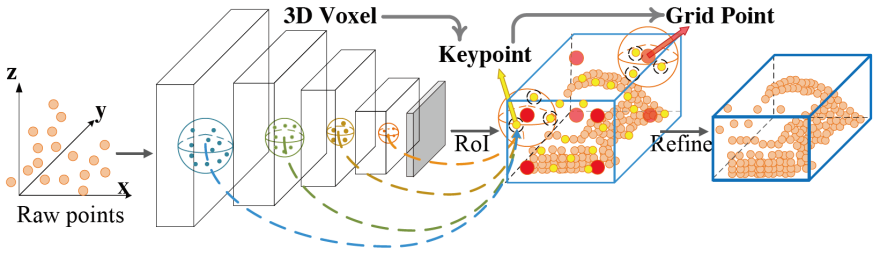


Figure 2.19: Architecture of PV-RCNN. [Shi+21] PV-RCNN deeply integrates both the voxel-based and the PointNet-based networks via a twostep strategy including the voxel-to-keypoint 3D scene encoding and the keypoint-to-grid RoI feature abstraction for improving the performance of 3D object detection.

Analysis: potentials and challenges of the point-voxel based methods. The point-voxel based methods can naturally benefit from both the fine-grained 3D shape and structure

information obtained from points and the computational efficiency brought by voxels. However, some challenges still exist in these methods. For the hybrid point-voxel backbones, the fusion of point and voxel features generally relies on the voxel-to-point and point-to-voxel transform mechanisms, which can bring non-negligible time costs. For the two-stage point-voxel detection frameworks, a critical challenge is how to efficiently aggregate point features for 3D proposals, as the existing modules and operators are generally time-consuming. In conclusion, compared to the pure voxel-based detection approaches, the point-voxel based detection methods can obtain a better detection accuracy while at the cost of increasing the inference time.

2.6 Evaluation Criterion for Object Detection

Evaluating object detectors typically involves assessing the quality of the generated bounding boxes and class labels. Detected objects are categorized as true positives (TP) if they match a ground-truth bounding box, or as false positives (FP) if they don't. Ground-truth bounding boxes with no corresponding predicted box are termed false negatives (FN). This matching is often done using the Intersection over Union (IoU) metric. Equation 2.7 shows how IoU quantifies the overlap between two bounding boxes. A match between a predicted and a ground-truth bounding box is considered a TP if it surpasses a certain threshold, usually 0.5 or 0.7 [Gei+13]. Using the counts of FP, TP, and FN, one can calculate precision and recall as per equations 2.8 and 2.9.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} \tag{2.7}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{All\ Ground\ Truths} \tag{2.8}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{All\ Detect\ Results} \tag{2.9}$$

In object detection, there is a trade-off between precision and recall. Precision measures how many of the predicted objects are actual objects, while recall evaluates how many actual objects the model predicted. Object detectors often output a confidence score to indicate their certainty that a proposed object is genuine. Setting a high threshold usually results in high precision but lower recall, whereas a lower threshold tends to produce lower precision and higher recall. Rather than assessing object detectors at a single, arbitrary threshold, the metric of mean average precision (mAP) considers a range of thresholds [Pow20]. Figure 2.20 shows the precision-recall curve for three different object detectors at 20 different thresholds. The area under the curve is the mAP score. The mAP score is the most common metric for evaluating object detectors. Detector 0 in figure 2.20 achieves an mAP score of 0.91, detector 1 of 0.77, and detector 2 of 0.95. The closer the mAP value is to 1 (ideal), the better the detector performs, meaning detector 2 has the best performance of the three.

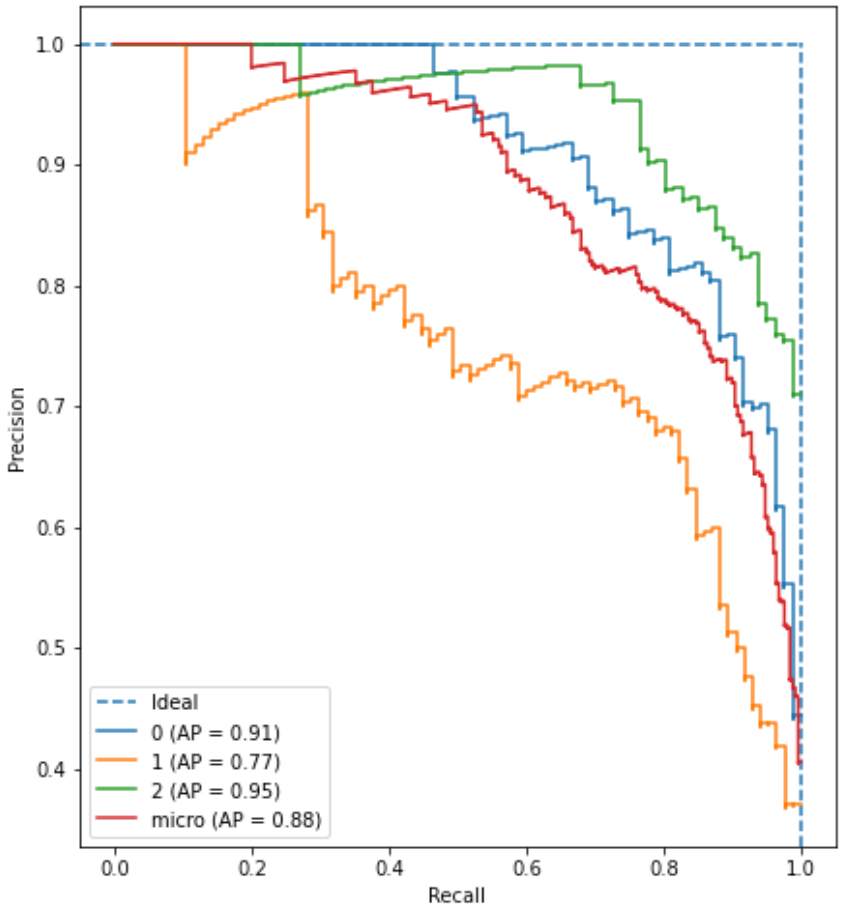


Figure 2.20: Examples of Recall-Precision curve for three detectors. Since TP+FN = All GT = const, *Recall* increases means TP increases, hence FN decreases. As TP has increased, only if FP decreases, will the *Precision* remain high i.e. the model will be doing less mistakes and hence will be good. Usually, Precision - Recall curve starts with high precision values, decreasing as recall increases.

Chapter 3

Related Works

This chapter gives detailed discuss about the related research that provides a foundation for this work. The aim is to give the reader an idea of how this work is embedded into a broader research field and how other work is applying similar ideas. First, some mainstream approaches of 3D object detection are introduced. Then, the recent research workings concerning with modeling object relation are presented. Finally, the newly proposed mamba-based approaches dealing with vision tasks are presented.

3.1 3D Object Detection

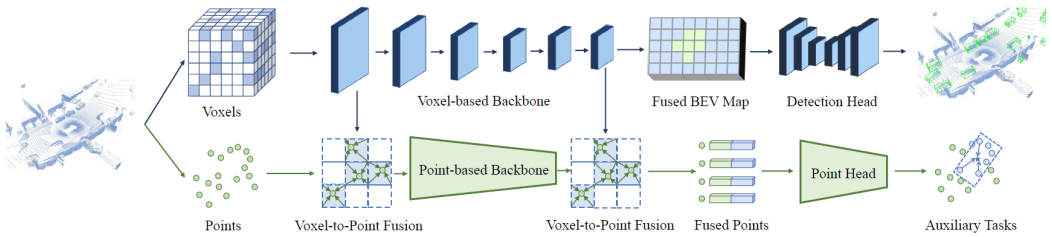
This section first gives an introduction to further categorization of point-voxel based 3D object detection methods: the one-stage detector and two-stage detector. Then there comes detailed reviews about the 3D object detection methods related to our work, including the architecture of our two baseline models PV-RCNN [Shi+21] and PartA2 [Shi+20].

3.1.1 One-stage and Two-stage 3D object detectors

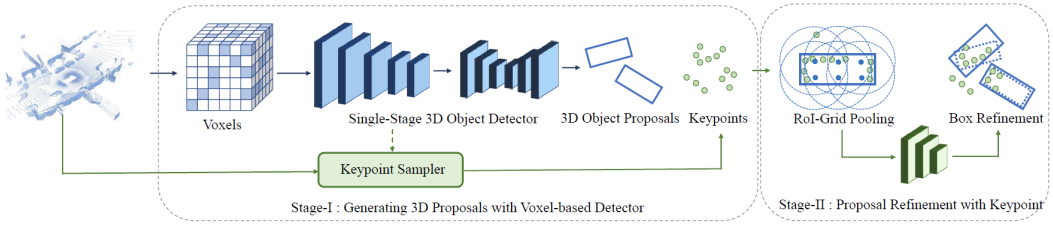
Point-voxel based approaches resort to a hybrid architecture that leverages both points and voxels for 3D object detection. These methods can be divided into two categories: the single-stage and two-stage detection frameworks. An illustration of the two categories is shown in Fig. 3.1.

One-stage point-voxel detection frameworks. Single-stage point-voxel based 3D object detectors try to bridge the features of points and voxels with the point-to-voxel and voxel-to-point transform in the backbone networks. Points contain fine-grained geometric information and voxels are efficient for computation, and combining them together in the feature extraction stage naturally benefits from both two representations. The idea that leverages point-voxel feature fusion in backbones has been explored by many works, with the contributions like point-voxel convolutions [Liu+19; Tan+20], auxiliary point-based networks [Li+21; He+20], and multi-scale feature fusion [Mia+21; NLH21].

Two-stage point-voxel detection frameworks. Two-stage point-voxel based 3D object detectors resort to different data representations for different detection stages. Specifically, at the first stage, they employ a voxel-based detection framework to generate a set of 3D object proposals. In the second stage, keypoints are first sampled from the input point cloud, and then the 3D proposals are further refined from the keypoints through novel point operators. PV-RCNN [Shi+21] is a seminal work that adopts SECOND [YML18] as the first-stage detector, and the RoI-grid pooling operator is proposed for the second-stage refinement. The



(a) One-Stage Point-Voxel Detection Framework



(b) Two-Stage Point-Voxel Detection Framework

Figure 3.1: An illustration of one-stage and two-stage point-voxel based 3D object detection methods. [Mao+23]

following works try to improve the second-stage head with novel modules and operators, e.g. RefinerNet [Che+19], VectorPool [Shi+22], point-wise attention [Wan+20], scale-aware pooling [LWW21], RoI-grid attention [MWL19], and point density-aware refinement module [HKW22].

3.1.2 PV-RCNN

Point-Voxel RCNN (PV-RCNN) [Shi+21] was identified in section 3.1 as a two-stage detector with a refinement stage pooling from different sources: raw points, multi-level voxel-features, and BEV-features. The main idea of this model is to combine the strengths of voxel-based and point-based approaches for the purpose of both the computationally efficiency and the detection accuracy.

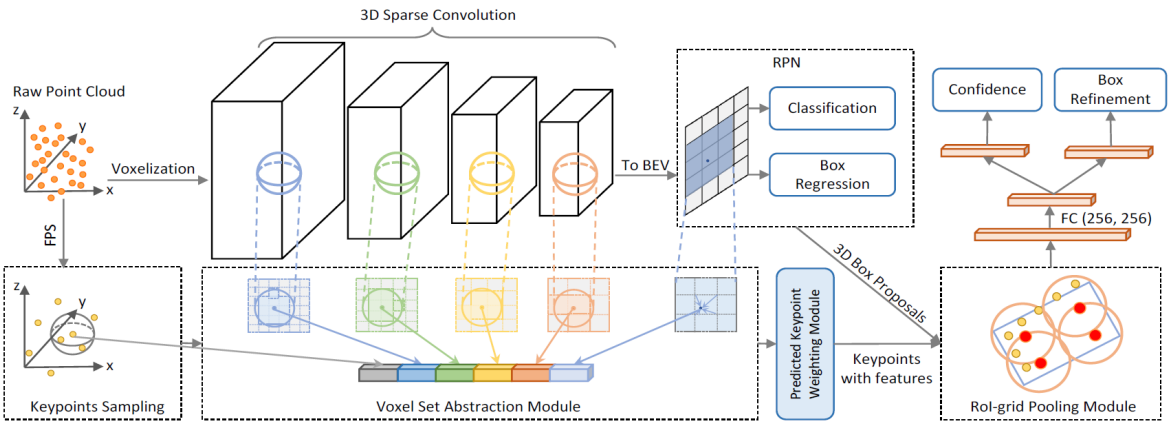


Figure 3.2: The overall architecture of PV-RCNN [Shi+21] The raw point clouds are first voxelized to feed into the 3D sparse convolution based encoder to learn multi-scale semantic features and generate 3D object proposals. Then the learned voxel-wise feature volumes at multiple neural layers are summarized into a small set of key points via the novel voxel set abstraction module. Finally the keypoint features are aggregated to the RoI-grid points to learn proposal specific features for fine-grained proposal refinement and confidence prediction.

In the voxel-wise approach, first the raw point clouds $\mathbf{P}$ are first voxelized into a voxel set with the spatial resolution of $L \times W \times H$. The spatial resolution, i.e. the grid size serves as hyperparameters and should be adapted with the dataset. Then this voxel set would be fed into a series of 3D sparse convolution based encoder to obtain multi-scaled feature volumes and also generate 3D object proposals. The final 3D feature volumes are stacked along the z-axis to obtain a 2D BEV features map. This format allows the network to make use of existing 2D detection heads, where we can apply VSS-Block. The SSD detector is used to obtain a set of high-quality 3D proposals for each pixel. Here, the pixel features are forwarded to a classification head to predict a class probability as well as to a regression head to predict the 3D bounding box values. The result is a set of proposals obtained from the first stage of the model that is subsequently forwarded to the refinement stage. Besides utilizing both voxel features and BEV features, the model also incorporates raw points during the refinement stage.

In another approach, the raw point cloud $\mathbf{P}$ is processed with Farthest Point Sampling (FPS) algorithm to be down-sampled into a subset of a much smaller number of points. This subset is called *Keypoints* $\mathcal{K}$, where each keypoint $p_i \in \mathcal{K}$ is encoded with raw point features f_i^{raw} to preserve the original point cloud information. The number of keypoints n serves as hyperparameter specific to dataset (eg. for KITTI dataset $n = 2048$; for Waymo open dataset $n = 4096$). The subsequent *Voxel Set Abstraction* (VSA) module aims to aggregate multi-scale semantic features from the 3D backbone onto these keypoints. For each individual keypoint $p_i \in \mathcal{K}$, the neighboring non-empty voxels at within a radius r_k at the k-th voxelization level are identified and the corresponding voxel features are retrieved by pooling. The VSA aggregates voxel features from four different levels $\{f_i^{pv_k}\}_{k=1}^4$ and concatenates raw point features f_i^{raw} and BEV features f_i^{BEV} at the keypoint location as well. The resulting feature vector $f_i^{(p)}$ of keypoint p_i is written as:

$$f_i^{(p)} = \text{Concat}\Big[\{f_i^{pv_k}\}_{k=1}^4, f_i^{raw}, f_i^{BEV}\Big]_{i=1\ldots n} \tag{3.1}$$

Intuitively, points corresponding to foreground objects should have a greater influence on proposal refinement than keypoints from the background. To achieve this, the model introduces a *Predicted Keypoint Weighting* module for re-weighting the keypoint features, denoting as $\tilde{\mathcal{F}} = \{\tilde{f}_j^{(p)}\}_{i=1}^n$. After processing through this module, the weighted keypoint features $\tilde{\mathcal{F}}$ not only include multi-scale semantic features from the 3D voxels but also have precise location data via their 3D keypoint coordinates. This provides the model with the ability to preserve the 3D structural information of the entire scene. Thus, the keypoints make use of a point-voxel approach. The *RoI-grid pooling* module infuses proposals with information from corresponding keypoints based on their positions (Fig. 3.3).

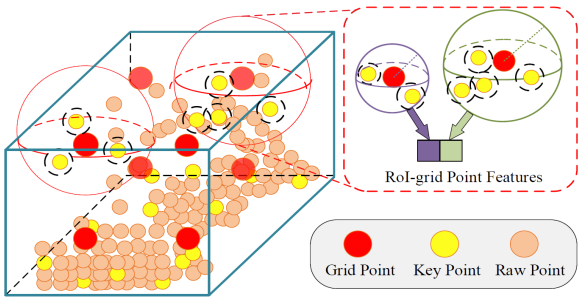


Figure 3.3: Illustration of RoI-grid pooling module [Shi+21]: Rich context information of each 3D RoI is aggregated by the set abstraction operation with multiple receptive fields.

For a given 3D proposal, the model uniformly samples $6 \times 6 \times 6$ grid points $\mathcal{G}$ within the proposal. To aggregate the features from keypoints to these grid points, the RoI-grid pooling

module first determines the neighboring keypoints for each grid point. Keypoints within a radius $r^{(g}$ would be considered. Notably, one single keypoint might contribute to multiple grid points because the neighboring spaces around grid points can overlap. Eq 3.2 illustrates the operation of the RoI-grid pooling module. For keypoints within $r^{(g)}$ of a certain grid point $g_i \in \mathcal{G}$, both the keypoint features and relative positions to the grid point are utilized.

$$\tilde{\Psi} = \left\{ \left[\tilde{f}_j^{(p)}; p_j - g_i \right]^\top \;\middle|\; \begin{array}{l} \|p_j - g_i\| < r^{(g)}, \\ \forall p_j \in \mathcal{K}, \forall g_i \in \mathcal{G}, \forall \tilde{f}_j^{(p)} \in \tilde{\mathcal{F}} \end{array} \right\} \tag{3.2}$$

Given the RoI-aligned features, two sibling subnetworks are used to predict confidence score and execute proposal refinement. The confidence score is used to filter out low-quality proposals. The proposal refinement is used to adjust the proposal location and size locally. The final output of the model is a set of refined proposals with their corresponding confidence scores.

3.1.3 PartA2

Part-aware and Part-aggregation Network (PartA2) [Shi+20] selects points from the point cloud, focusing on parts of objects that are most informative for detection tasks. In comparison to the FPS method in PV-RCNN, which selects keypoints based on spatial distribution or feature richness, the *Part-aware* in PartA2 sampling emphasizes the semantic significance of each point, leading to a more effective representation of the object's geometry. As shown in Fig.3.4, in a sub-step the architecture predicts to which part of the object each point belongs. This information is in the subsequent model used to make more precise predictions. The architecture of PartA2 is designed to be both computationally efficient and highly effective in detecting objects.

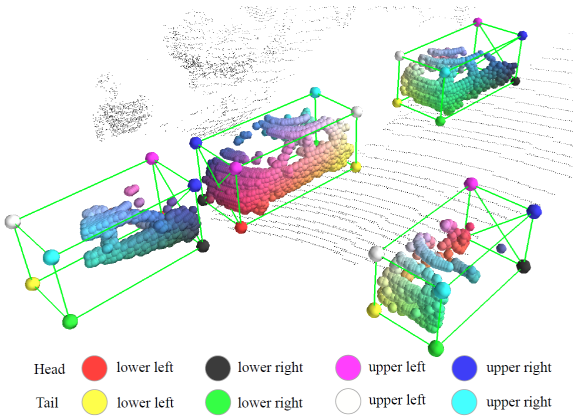


Figure 3.4: Part-aware sampling [Shi+20]: accurately predicts intra-object part locations even when objects are partially occluded. Such part locations can assist accurate 3D object detection. The predicted intra-object part locations are visualized by interpolated colors of eight corners.

The overall PartA2 architecture (Fig. 3.5) consists of two stages: *1) Part-aware 3D proposal generation* and *2) Part location aggregation*. In *1) Part-aware 3D proposal generation* stage, first the raw point clouds are voxelized and the center of each non-empty voxel is considered as a new point cloud with spatial regularity. Upon this, the point-wise feature are learned via sparse convolution. Then the subsequent step is the estimation of foreground points and intra-object part locations. The foreground points are segmented using focal loss, while their intra-object part locations are predicted using binary cross entropy loss. After the

part-aware sampling, 3D proposal boxes are generated. With the intra-object part location and 3D proposals, the RoI-aware point cloud feature pooling conducts box scoring and proposal refinement by aggregating the part information and learned point-wise features of all the points within the same proposal. In *2) Part location aggregation* stage, the predicted part locations and semantic part features are fused together and bounding boxes are scored with 3D IoU guiding.

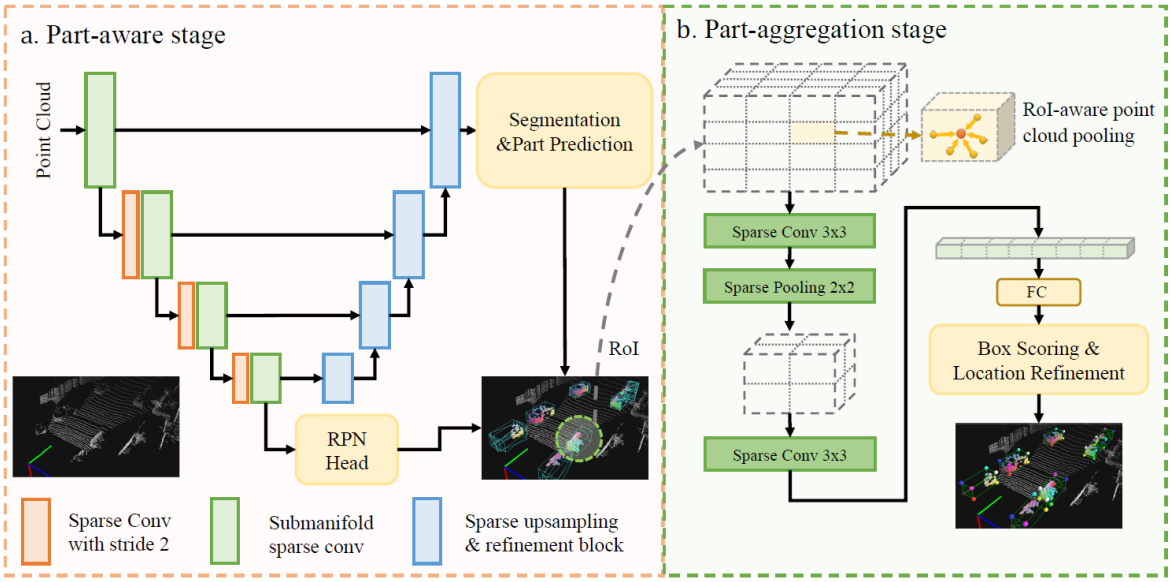


Figure 3.5: The overall architecture of PartA2 [Shi+20] It consists of two stages: (a) The first part-aware stage estimates intra-object part locations accurately and generates 3D proposals by feeding the raw point cloud to our newly designed backbone network. (b) The second part-aggregation stage conducts the proposed RoI-aware point cloud pooling operation to group the part information from each 3D proposal, then the part-aggregation network is utilized to score boxes and refine locations based on the part features and information.

3.1.4 PointNet

PointNet [Qi+17a] is a pioneering deep learning framework introduced by Charles R. Qi *et al.* in the year 2017. It is specifically designed to handle point cloud data. As a point-based method, the key features and innovations of PointNet can be summarized as: *1) Direct Processing of Point Clouds*: Unlike traditional methods that transform point clouds into structured formats (e.g., voxel grids or 2D images), PointNet operates directly on raw point sets. This allows it to retain the fine-grained details of the 3D data. *2) Permutation Invariance*: PointNet is designed to be invariant to the permutation of input points. This is achieved using a symmetric function (max pooling) that aggregates information from all points into a global feature vector, ensuring that the network's output remains consistent regardless of the order of input points. *3) Hierarchical Structure*: PointNet uses a hierarchical neural network architecture. It processes local features of points and aggregates them into global features, effectively capturing both local and global geometrical structures, and *4) Transformation Networks*: To address the issue of varying orientations and scales of objects in point clouds, PointNet introduces a mini-network called the *T-Net* (Fig. 3.6). This network predicts an affine transformation matrix, which aligns the input point cloud to a canonical space.

Upon these characteristics, PointNet has advantages in efficiency, simplicity and implementation flexibility. PointNet is computationally efficient due to its simple yet powerful architecture. It processes point clouds directly without needing complex pre-processing steps.

The architecture is relatively simple and easy to implement, making it accessible for further research and practical applications. And PointNet can be adapted and extended for various tasks involving point cloud data, including 3D shape reconstruction and point cloud upsampling. Yet, PointNet has its shortcomings in local structure awareness and scalability. While PointNet captures global features effectively, it may not always capture detailed local structures as efficiently as some of the later developed methods, like PointNet++ [Qi+17b]. In aspect of scalability, handling very large point cloud scenes or high-density data can be challenging for basic PointNet without any modifications.

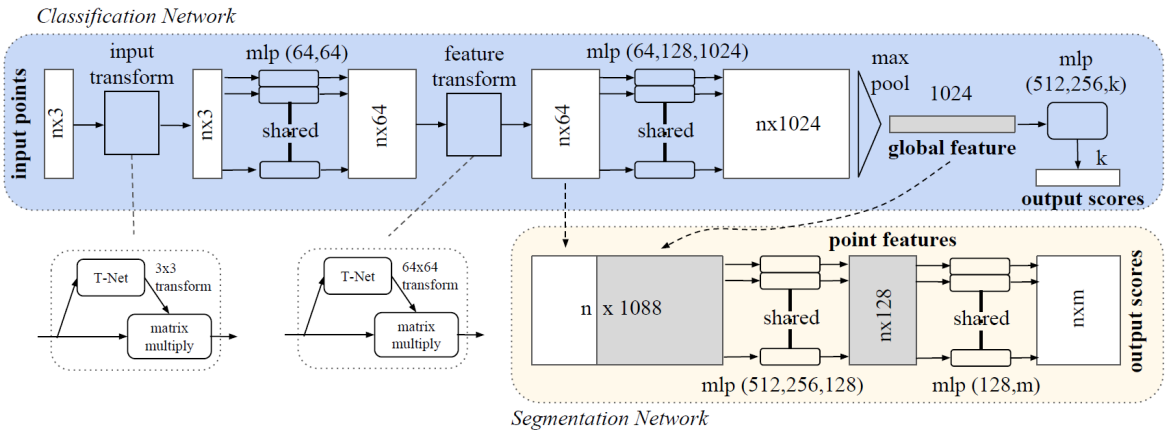


Figure 3.6: The overall architecture of PointNet [Qi+17a] The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. "mlp" stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net.

3.1.5 PointPillars

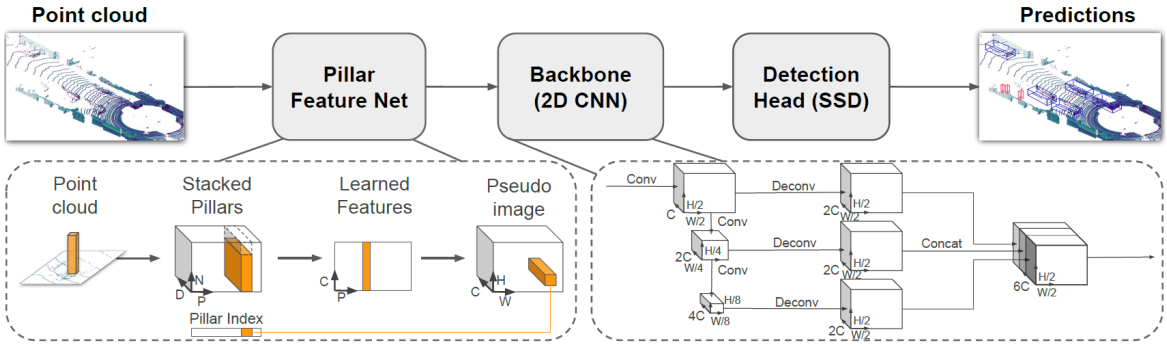


Figure 3.7: The overall architecture of PointPillars [Lan+19] The main components of the network are a Pillar Feature Network, Backbone, and SSD Detection Head. The raw point cloud is converted to a stacked pillar tensor and pillar index tensor. The encoder uses the stacked pillars to learn a set of features that can be scattered back to a 2D pseudo-image for a convolutional neural network. The features from the backbone are used by the detection head to predict 3D bounding boxes for objects. Note: here we show the backbone dimensions for the car network.

As a pillar-based approach, PointPillars [Lan+19] aims to find a middle ground between the unstructured nature of point clouds in point-based methods and the structured representations in voxel-based methods. This is done by dividing the point cloud into vertical columns,

or 'pillars', each containing points along the z-axis. PointPillars was a pioneering work using this approach. Here, each pillar is processed to obtain a feature vector, usually by using PointNet-like architectures. These features are arranged into a *pseudo-image* to which 2D convolutions can be applied. The pseudo-image format allows the model to capture both local features within each pillar and global contextual information. Pillar-based methods allow for efficient computation while still retaining fine-grained information [LLY23]. Specifically in applications where the vertical dimension does not contain much critical information, such as autonomous driving, pillar-based methods offer a good trade-off between efficiency and precision. Fig. 3.7 shows the architecture of PointPillars. We can see how the pseudo-image is formed by arranging the pillar features into a 2D grid. This grid is then processed by classical 2D convolutions.

3.2 Object Relation

There are already some works considering object relation features for 3D object detection in the indoor scenarios [20; Lan+22; Lan+21]. For outdoor scenes, specifically in the field of autonomous driving, the concept of relation features is relatively less explored. In this section we give a review about some methods exploiting object relation explicitly, which inspired our work.

3.2.1 Ret3D

Ret3D [Wu+22] is a two-stage detector that refines the detection results of one-stage detectors efficiently. It uses CenterPoint [YZK21] as the first stage to obtain object proposals. In the refinement stage, the basic features of each proposal are concatenated with map-view features pooled from the backbone at the respective locations. The feature vectors for each proposal are connected in a graph and related to each other via message passing. Then they are forwarded to the heads in which refined locations and labels are predicted.

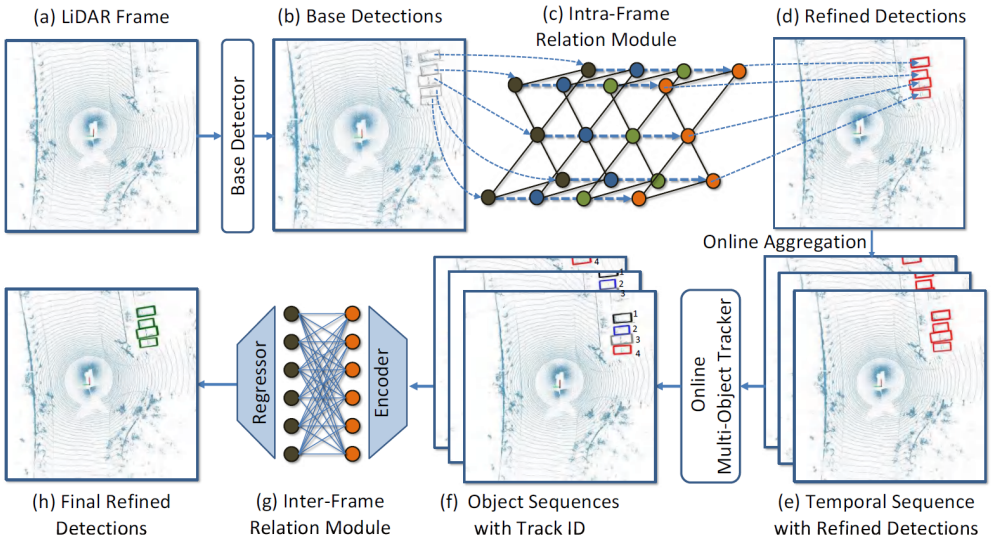
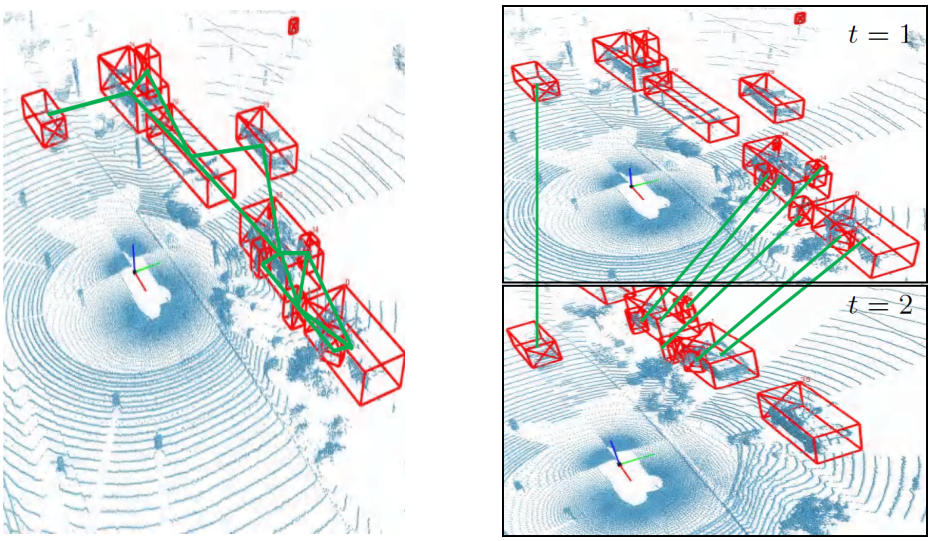


Figure 3.8: The pipeline of Ret3D [Wu+22]Ret3D is a two-stage detector that refines the detection results of one-stage detectors efficiently. Ret3D consists of two parts, IntraRM and InterRM, for refining detection results using intra-frame and inter-frame object relations, respectively.

The relation-module consists of two parts, IntraRM and InterRM, for refining detection results using intra frame and inter-frame object relations, respectively (Fig. 3.9). The intra-frame relation module (IntraRM), which constructs a sparse object graph to refine the feature of objects within the same LiDAR frame through message passing, while the inter-frame relation module (InterRM), which densely connects each object in its tracked sequences to further refine its representations through a lightweight transformer network.



(a) Intra-frame relation **(b)** Inter-frame relation

Figure 3.9: Intra-frame (a) and inter-frame (b) object relation [Wu+22]. Green lines indicate object relations. For simplicity, only a short sequence with a length of 2 is used to illustrate inter-frame object relations. Best viewed in color.

Ret3D represents a significant advancement in 3D object detection by rethinking the importance of object relations in driving scenes. Its innovative use of graph-based representations and attention mechanisms to model object interactions enhances both the accuracy and efficiency of 3D object detection models. This makes Ret3D particularly valuable for applications in autonomous driving and robotics, where understanding complex environmental contexts is crucial. The approach highlights the potential of integrating relational reasoning into deep learning models to achieve more robust and contextually aware object detection.

3.2.2 GACE

GACE [Sch+23], the geometry-aware confidence enhancement, is proposed to refine the confidence values for a given set of detections by exploiting the rich geometric information contained directly in the detections as well as in the underlying 3D points. In the traditional backbone-neck-head architecture of object detectors, the accuracy of the localization is hardly included in the confidence score. Confidence estimation is usually performed using only features within a small area around the object (depending on the receptive field), no explicit information about neighboring objects and their geometric properties or confidence values is used.

To re-evaluate the confidence score of a detection, as the orange 3D bounding box shown in Fig. 3.10, GACE [Sch+23] aggregates geometric properties of the detection itself and the points it contains into a feature vector (top). To capture the context of the detection, geometric relationships to neighboring detections are aggregated using a shared MLP and

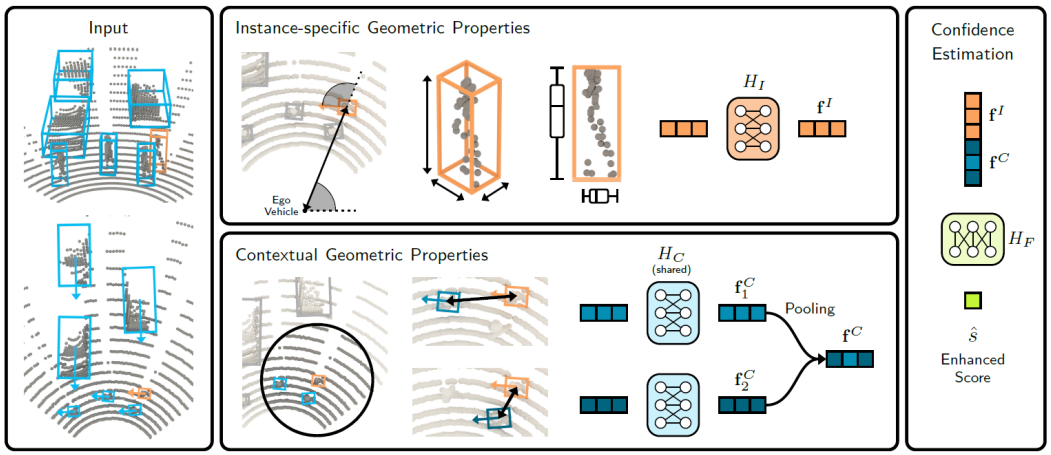


Figure 3.10: Instance-specific geometric properties and contextual information extraction in GACE [Sch+23] To re-evaluate the confidence score of a detection (orange), we aggregate geometric properties of the detection itself and the points it contains into a feature vector (top). To capture the context of the detection, geometric relationships to neighboring detections are aggregated using a shared MLP and subsequent pooling function (bottom). By merging both features (right), we obtain a new confidence score that takes into account the underlying geometric properties of the detection.

subsequent pooling function (bottom). And by merging both features (right), a new confidence score that takes into account the underlying geometric properties of the detection would be obtained.

$$\mathbf{f}^I = H_I\Big([\mathbf{b}, \alpha, \|\mathbf{c}\|, |\mathcal{X}_\mathbf{b}|, \mathcal{X}_\mathbf{b}^{\text{mean}}, \mathcal{X}_\mathbf{b}^{\text{std}}, \mathcal{X}_\mathbf{b}^{\text{min}}, \mathcal{X}_\mathbf{b}^{\text{max}}]^\top\Big) \tag{3.3}$$

The feature vector for instance-specific plausibility is defined as Eq. 3.3, where H_I stands for a MLP. The considered instance-level properties includes bounding box properties heading angle α, the distance to the observer $\|\mathbf{c}\|$, and the point distribution of the points within the bounding box$\mathcal{X}_\mathbf{b} \subset \mathcal{X}$.

$$\mathbf{f}_n^C = H_C\Big([\|\mathbf{c}-\mathbf{c}_n\|, \mathbf{c}-\mathbf{c}_n, \Theta-\Theta_n, y_n, \mathbf{f}_n^I]^\top\Big) \tag{3.4}$$

The contextual feature vector $\mathbf{f}_{n\ n=1\ldots n}^C$ between the object and its neighboring object n is defined as Eq. 3.4, where H_C stands for a MLP. The considered relation properties includes distance between the neighbor $\|\mathbf{c}-\mathbf{c}_n\|$, the direction vector $\mathbf{c}-\mathbf{c}_n$, the heading angle difference $\Theta-\Theta_n$, the neighbor's class label y_n and neighbor's instance-specific feature $\mathbf{f}_n^I$. To impose no constraint on the number of features, all these n contextual feature vectors w.r.t each individual neighbor are then pooled as one contextual feature vector $\mathbf{f}^C$ for the object.

3.2.3 DGCNN

DGCNN [Wan+19] by Wang *et al.* proposed the Dynamic Graph CNN for indoor 3D analysis. DGCNN addresses the challenges of capturing local geometric structures and relationships in point clouds, which are crucial for tasks like 3D object recognition and segmentation. DGCNN constructs graphs dynamically during the learning process. This involves defining a graph where each point is a node, and edges are formed based on the k-nearest neighbors (k-NN) in the feature space rather than the Euclidean space. This dynamic nature allows the network to adapt and capture complex local structures as the feature space evolves through the network layers.

The core innovation of DGCNN is the *Edge Convolution* (EdgeConv) operation (Fig. 3.11). Unlike traditional convolutions, EdgeConv operates on edges connecting neighboring points, allowing it to explicitly model the relationships and interactions between points. This operation captures local geometric features effectively and can aggregate both point-level and edge-level features. Besides, DGCNN employs a hierarchical approach to feature learning, where local features are progressively aggregated and refined through multiple layers of EdgeConv. This hierarchical structure helps in capturing both fine-grained local features and global context. After several EdgeConv layers, a global feature vector is obtained by aggregating features across all points using a symmetric function. This global feature vector is used for tasks like classification and segmentation.

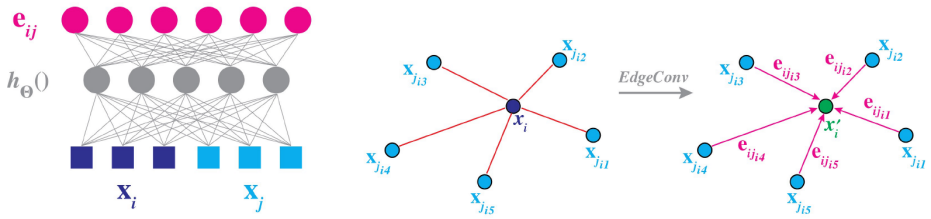


Figure 3.11: EdgeConv proposed by DGCNN [Wan+19] *Left*: Computing an edge feature, e_{ij}(top), from a point pair, x_i and x_j (bottom). In this example, $h_\Theta()$ is instantiated using a fully connected layer, and the learnable parameters are its associated weights. *Right*: The EdgeConv operation. The output of EdgeConv is calculated by aggregating the edge features associated with all the edges emanating from each connected vertex.

The network is designed to be invariant to the permutation of input points. By leveraging symmetric functions (e.g., max pooling) during the aggregation of features, DGCNN ensures that the output remains consistent regardless of the order of input points.

3.2.4 Object DGCNN

While DGCNN mainly deals with part-segmentation within point clouds of one single object, Object-DGCNN [WS21] generalizes dynamic graph regarding objects as nodes and integrates the dynamic graph learning capabilities into a 3D object detection framework. This involves using the learned point features to predict object bounding boxes and classes within the point cloud.

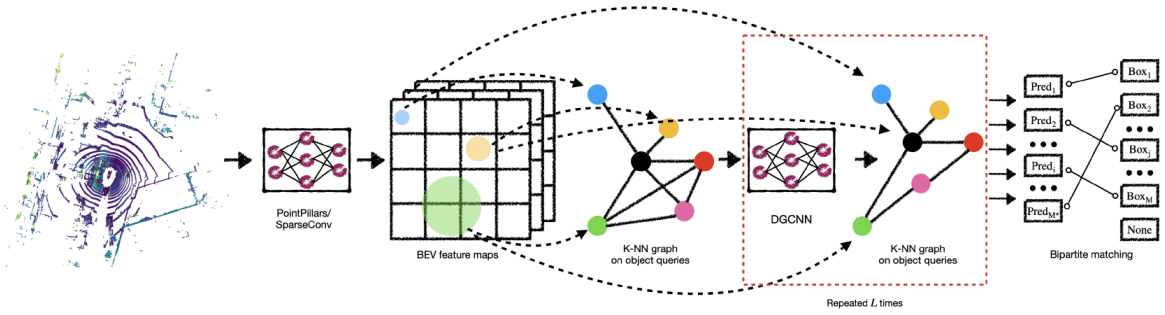


Figure 3.12: Architecture of Object-DGCNN [WS21] Point cloud features are learned in BEV, followed by L DGCNNs to model object relations. Each layer employs the following steps: *1)* predict a set of query points and attention weights; *2)* collect BEV features from keypoints determined by the queries; and *3)* model object-object interactions via DGCNN. Object-DGCNN predicts a set of bounding boxes and compute loss in a one-to-one manner.

Object-DGCNN uses the DGCNN-generated sparse set of object relations and then leverages query-based GNNs to further strengthen the sparse BEV features. This set-to-set distil-

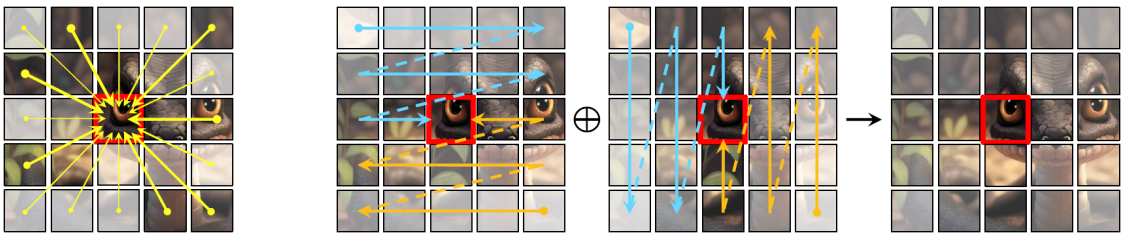
lation method aligns the outputs of the teacher and the student in a permutation invariant fashion. As shown in FIg. 3.12, the object-relation graph is updated dynamically based on the current feature representations of the points. This dynamic graph construction ensures that the model adapts to the changing feature space, capturing different levels of geometric detail. The stacked EdgeConv layers are used to learn point features at various levels of abstraction. Each EdgeConv layer updates the features by considering the relationships with neighboring points, effectively capturing local geometric structures. The final stage includes a detection head that utilizes the learned point features to predict 3D bounding boxes and object classes. The detection head processes the aggregated features to output the final detection results.

3.3 Mamba in Computer Vision

In section 2.3.3 we already had a basic understanding of the state space model (SSM) and the selective SSM (S6), i.e. Mamba [GD23]. In this section we cover some further Mamba-based approaches applied in the field of computer vision, specifically object detection for 2D images and 3D point clouds.

3.3.1 VMamba

VMamba [Liu+24c] introduced the *Visual State Space block* (VSS-Block) to extend the application of *State Space Model* (SSM) from the field dealing with 1-D sequential data, e.g. natural language processing (NLP), to 2D visual representation learning. To address the challenge that the vision data inherently lacks a sequential arrangement of visual components, the *2D Selective Scan* (SS2D) module, a four-way scanning mechanism tailored for spatial domain traversal is proposed in VMamba. In contrast to the self-attention mechanism (Fig. 3.13 (a)), SS2D ensures that each image patch gains contextual knowledge exclusively through a compressed hidden state computed along the corresponding scanning path (Fig. 3.13 (b)), thereby reducing the computational complexity from quadratic to linear.



(a) Self-Attention **(b)** 2D-Selective-Scan (SS2D)

Figure 3.13: Comparison of correlation establishment between image patches via (a) self-attention and (b) the proposed 2D-Selective-Scan (SS2D). red boxes indicate the query image patch, with patch opacity representing the degree of information loss.

While the sequential nature of the scanning operation in S6 aligns well with NLP tasks involving temporal data, it poses a significant challenge when applied to vision data, which is inherently non-sequential and encompasses spatial information (e.g., local texture and global structure). To address this issue, S4ND [Ngu+22] reformulates SSM with convolutional operations, directly extending the kernel from 1D to 2D through outer-product. However, such modification prevents the weights from being input-independent, resulting in a limited capacity for capturing contextual information. Therefore, VMamba adhere to the selective scan

approach [GD23] for input processing, and propose the 2D-Selective-Scan (SS2D) module to adapt S6 to vision data without compromising its advantages.

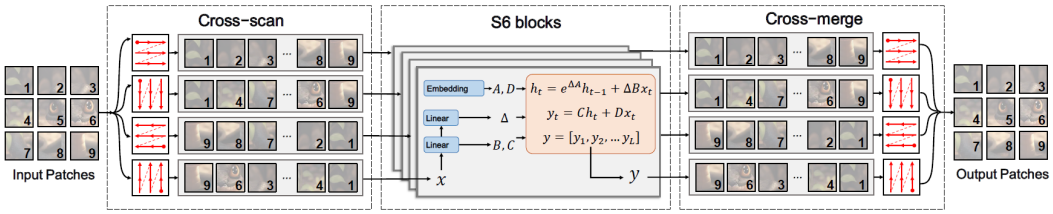


Figure 3.14: Mechanism of 2D-Selective-Scan (SS2D). Input patches are traversed along four different scanning paths *(Cross-Scan)*, and each sequence is independently processed by distinct S6 blocks. Subsequently, the results are merged to construct a 2D feature map as the final output *(Cross-Merge)*.

As illustrated in Fig. 3.14, data forwarding in SS2D involves three steps: cross-scan, selective scanning with S6 blocks, and cross-merge. Given the input data, SS2D first unfolds input patches into sequences along four distinct traversal paths (*i.e.*, Cross-Scan), processes each patch sequence using a separate S6 block in parallel, and subsequently reshapes and merges the resultant sequences to form the output map (*i.e.*, Cross-Merge). By adopting complementary 1D traversal paths, SS2D enables each pixel in the image to effectively integrate information from all other pixels in different directions, thereby facilitating the establishment of global receptive fields in the 2D space.

With the SS2D module as the core of VSS-block, VSS-block acts as the core of VMamba architecture. The input image $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$ s first partitioned into patches by a stem module, resulting in a 2D feature map with the spatial dimension of $\frac{H}{4} \times \frac{W}{4}$, Subsequently, multiple network stages are employed to create hierarchical representations with resolutions of $\frac{H}{8} \times \frac{W}{8}$, $\frac{H}{16} \times \frac{W}{16}$, and $\frac{H}{32} \times \frac{W}{32}$. Each stage comprises a down-sampling layer (except for the first stage) followed by a stack of VSS blocks. The overall architecture of VMamba is illustrated in Fig. 3.15.

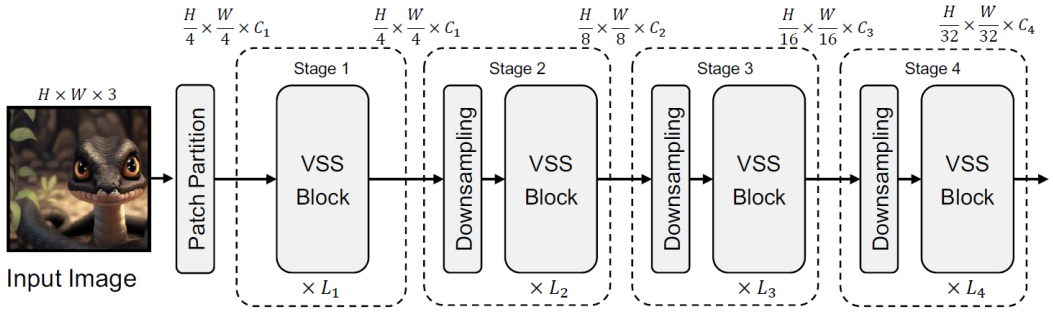


Figure 3.15: The overall architecture of a VMamba model.

3.3.2 Mamba in Mamba

Mamba-in-Mamba [Che+24] (MiM) is initially proposed to solve infrared small target detection (ISTD). Since the targets in ISTD are typically very small, directly transferring VMamba [Liu+24c] to ISTD is insufficient. ISTD necessitates a greater emphasis on local features compared to other vision tasks that predominantly involve standard-size targets. To achieve this, MiM divides the input image into several patches as "*visual sentences*" and then further divide them into sub-patches as "*visual words*". MiM uses an outer Mamba block to extract

features of visual sentences, and further assist it with inner Mamba blocks to excavate the local features of smaller visual words. In particular, features and relations between visual words in each visual sentence are calculated independently using a shared network to ensure the added amount of parameters and FLOPs is minimal. Then, the visual word features are consolidated back into their respective sentences.

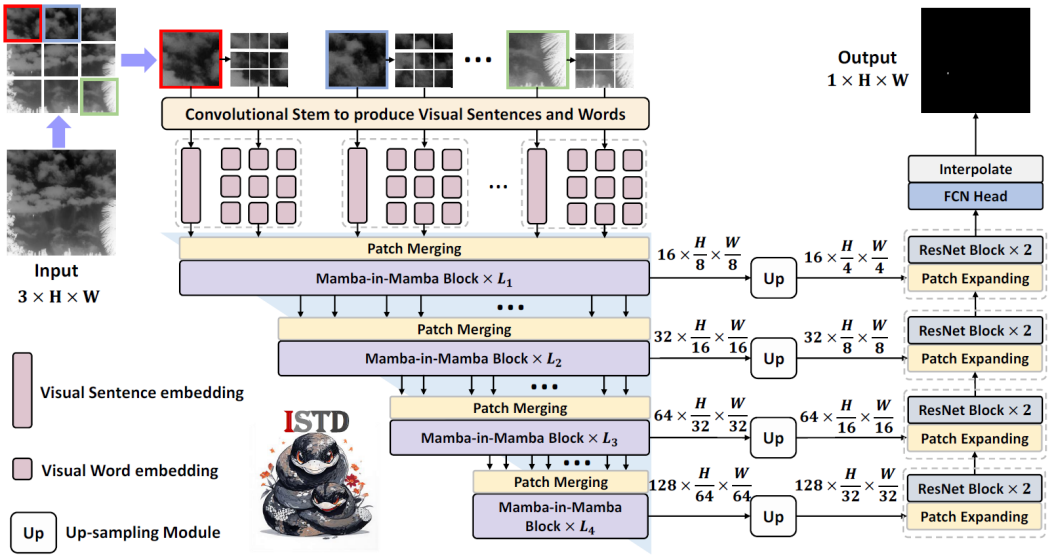


Figure 3.16: Overview of our MiM-ISTD [Che+24], which mainly includes a convolutional stem, a pure Mamba-based MiM hierarchical encoder, and a plain decoder. The inner Mamba block is shared in the same layer.

Given a 2D image $\mathcal{X} \in \mathbb{R}^{H\times W\times 3}$, it is divided evenly into n patches to form $\mathcal{X} = \{X^i\}_{i=1,2,...n}$, where each patch is in $\mathbb{R}^{n\times p\times p\times 3}$, with (p,p) denoting the resolution of each patch. In MiM, patches are viewed as *visual sentences* that represent the image. Further, each patch is segmented into m smaller sub-patches, making each visual sentence a sequential of *visual words*:

$$X^i = \left[x^{i,1}, x^{i,2}, ..., x^{i,m}\right] = \{x^{i,j}\}, j = 1, 2, ..., m \tag{3.5}$$

where $x^{i,j} \in \mathbb{R}^{s\times s\times 3}$ is the j-th visual word of the i-th visual sentence X^i, (s,s) is the spatial size of sub-patches. Since MiM adopts a hierarchical encoder structure, the spatial shapes of visual sentences and words are unfixed and will gradually decrease as the network layers deepen.

MiM constructs a convolutional stem, where a stack of 3×3 convolutions is utilized, to produce patches (visual sentences) $\in \mathbb{R}^{\frac{H}{8}\times\frac{W}{8}\times D}$ and sub-patches (visual words) $\in \mathbb{R}^{\frac{H}{4}\times\frac{W}{4}\times C}$ at the first stage, where D is the visual sentence dimension, C is the visual word dimension. Each visual word corresponds to a 2×2 pixel region in the original image, and each visual sentence is composed of 4×4 visual words at this stage. Unlike ViTs [Dos+21], the position embedding bias is not added to visual words and sentences due to the causal nature of visual mamba block.

The core part of MiM is its hierarchical encoder of four stages with different numbers of tokens, as shown in Fig. 3.16. MiM adopts the patch merging in Swin-unet [Cao+21] as the down-sampling operation. Each stage consists of multiple MiM blocks which process both word-level and sentence-level features. The visual words $x^{i,j}$ are mapped to a sequence of word embeddings $w^{i,j}$ via a linear projection, expressed as:

$$W^i = \left[w^{i,1}, w^{i,2}, ..., w^{i,m}\right], w^{i,j} = \mathrm{FC}(\mathrm{Vec}(x^{i,j})) \tag{3.6}$$

where $w^{i,j} \in \mathbb{R}^C$ is the j-th word embedding of the i-th visual sentence, c is the dimension of

word embedding, W^i is the collection of word embeddings of i-th visual sentence, and Vec$(\cdot)$ refers to the vectorization operation.

The MiM block handles two different data streams: one traverses through the visual sentences, while the other manages the visual words within each sentence. For the word embeddings, MiM block explores the relation between visual words:

$$W_l^i = W_{l-1}^i + \text{Mamba}(\text{LN}(W_{l-1}^i)), l = 1, 2, ..., L \tag{3.7}$$

where l is the index of the l-th block, and L is the total count of MiM blocks. All transformed word embeddings are denoted by $\mathcal{W}_l = \left[W_l^1, W_l^2, ..., W_l^n\right]$. This can be viewed as an inner Mamba block. In this process, the relationships among visual words are built by computing interactions among each two visual words. For example, in a patch containing a small target, a word denoting the target would have a stronger relation with other target-related words while interacting less with the background part.

At the sentence level, MiM generates sentence embedding memories as storage for the sequence of sentence-level representations $\mathcal{S}_l = \left[S_l^1, S_l^2, ..., S_l^n\right]$. In each layer, the sequence of word embeddings is mapped to the domain of sentence embedding by linear projection and subsequently integrated into the sentence embedding.

$$S_{l-1}^i = S_{l-1}^i + \text{FC}(\text{Vec}(W_l^i)), l = 1, 2, ..., L \tag{3.8}$$

By doing so, the sentence embedding can be augmented by the word-level features. Then, the outer Mamba block transforms the sentence embeddings:

$$\mathcal{S}_l = \mathcal{S}_{l-1} + \text{Mamba}(\text{LN}(\mathcal{S}_{l-1})), l = 1, 2, ..., L \tag{3.9}$$

The outer Mamba block can model the relationships among sentence embeddings. The inputs and outputs of the MiM block are visual word embeddings and sentence embeddings, so the MiM hierarchical encoder can be defined as:

$$\mathcal{W}_l, \mathcal{S}_l = \text{MiM}(\mathcal{W}_{l-1}, \mathcal{S}_{l-1}), l = 1, 2, ..., L \tag{3.10}$$

In MiM block, both the inner Mamba block and the outer Mamba block adopt the visual mamba block [Liu+24c]. The inner Mamba block models the relationship between visual words for local feature extraction, while the outer Mamba block captures the global feature by modelling the relationship between visual sentences.

3.3.3 PointMamba

Point Mamba [Liu+24a] is a SSM-based point cloud processing backbone with a causality-aware ordering mechanism. To construct the causal dependency relationship, an octree-based ordering strategy is deaigned on raw irregular points, globally sorting points in a z-order sequence and also retaining their spatial proximity. Point Mamba captures the global point-wise features with a hierarchical architecture, which can characterize the semantic affinities between points for various point cloud processing tasks.

The overall architecture of Point Mamba (Fig. 3.17) consists of an octree establishment layer, a feature embedding layer, and a sequence of Point Mamba blocks and downsampling layers. Point Mamba first normalizes the input point cloud and build the octree based on it. The point-wise features all are stored in the octree nodes, which are sorted by shuffled keys to form a z-order-based sequence. Then, the point cloud features are embedded into a higher dimension and apply a series of *Point Mamba Blocks* and downsampling layers to obtain the hierarchical point features. After getting the output features P_{out}, a lightweight Feature

Pyramid Network (FPN) [Kir+19] is applied for downstream classification or segmentation tasks.

Point Mamba block combines the bi-directional selective scanning mechanism [Liu+24c] to adjust the sequence-order dependence of Mamba. The reordered points from octree-based ordering stage are stacked into a 1-D sequence $P_{in} \in \mathbb{R}^{N \times C}$, where C is the embedded feature dimension and N denotes the number of points. Considering the linear complexity of Mamba, we can directly process the point cloud sequence with our designed Point Mamba Block without partitioning the window as previous transformers.

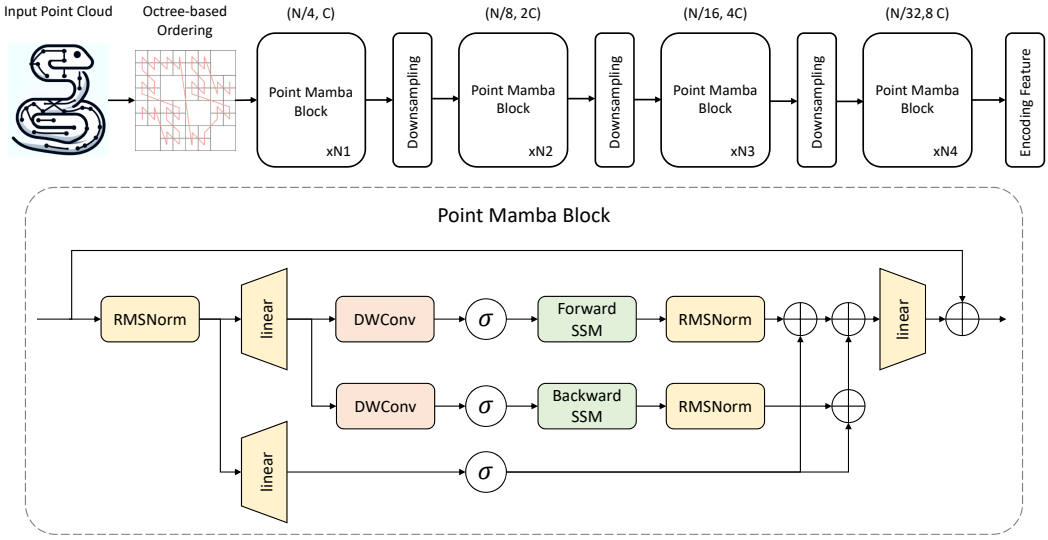


Figure 3.17: The overall architecture of Point Mamba [Liu+24a],which contains an octree establishment layer, a feature embedding layer, and a sequence of Point Mamba blocks and downsampling layers. N is the number of input points. C is the channel dimension of the point features. N_i denotes the number of Point Mamba Blocks in the i-th stage. The structure of one Point Mamba Block includes the core SSM module and the bidirectional selective scanning mechanism [Liu+24c].

Octree. Since the original design of the Mamba backbone is aimed at solving casual sequential language tasks, introducing Mamba into point cloud processing has a great challenge because of the intrinsic disorder and irregularity natures of point cloud. To construct the causal dependency relationship, inspired by OctFormer [Wan23], an octree-based ordering mechanism is designed to construct the causal dependency relationship based on a z-order curve for sorting the irregular point cloud data. Octree nodes are sorted by the shuffled keys, which can retain the original spatial proximity of raw point clouds. Then, we embed point features in different downsampling stages to get the hierarchical features for downstream tasks of the point cloud. Fig. 3.18 shows the mechanism of window partition of the octree, explained in figure caption.

The construction of an octree from a binary voxel grid starts by setting up the bounding cube as the root cell. Then, a cell is subdivided recursively whenever it contains both empty and full voxels, splitting it into 8 cubes, called the cells ”children”. This process is stopped when all cell’s contain only empty or full voxels (this happens at the latest, when the resolution of the original voxel grid is reached). Each cell in the tree can thus be either empty, full or mixed (only intermediate nodes). Next, this octree needs to be linearized into a sequence of tokens, from which the octree can be reconstructed. An example of quadtrees, the 2D variant of octrees is shown in Fig. 3.19 for easier visualization.

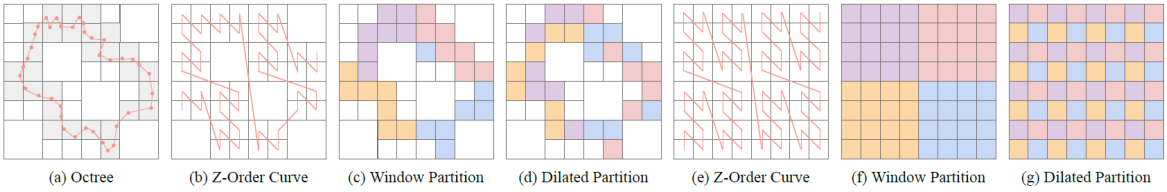
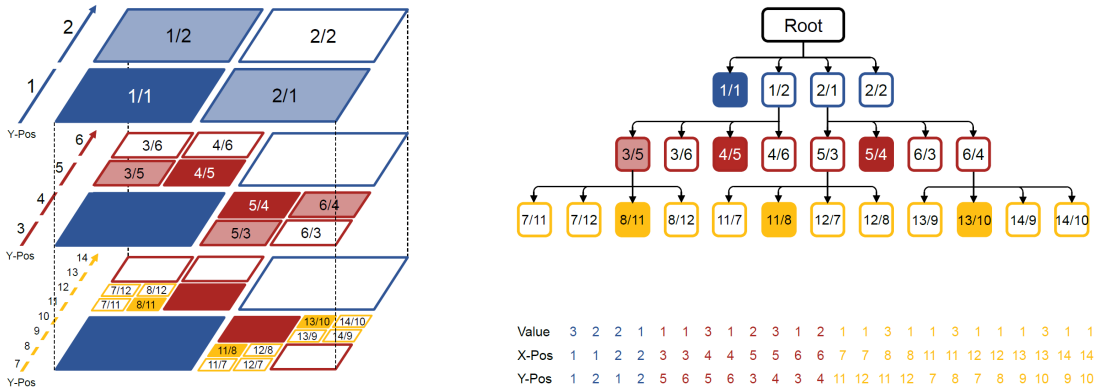


Figure 3.18: Window partition for the octree attention [Wan23] Here 2D images are shown for a better illustration. (a): An input point cloud sampled from a shape in red and the corresponding octree (quadtree). Non-empty octree nodes are highlighted in gray. (b): Z-order curve at depth 3 of the octree. (c): A window partition generated by tensor reshaping and transposing corresponding to (b), with a point number of 7. The features are stored in a tensor following the order of non-empty octree nodes on the z-order curve. (d): A dilated partition with a point number of 7 and a dilation of 2. (e): Z-order curve covering the whole space. (f): A window partition corresponding to (e) with a point number of 16. (g): A dilated partition corresponding to (e) with a point number of 16 and a dilation of 4.



(a) Shape **(b)** Tree and sequence

Figure 3.19: An example of octree encoding [IKK21] (for simplicity a 2D quadtree is depicted). Position values are represented as X/Y for the x- and y-coordinates. Colors are used to indicate the depth level. Filled cells represent full (value=3), white cells empty (value=1) and transparent (value=2) cells mixed voxels. Below are the resulting value and positional sequences.

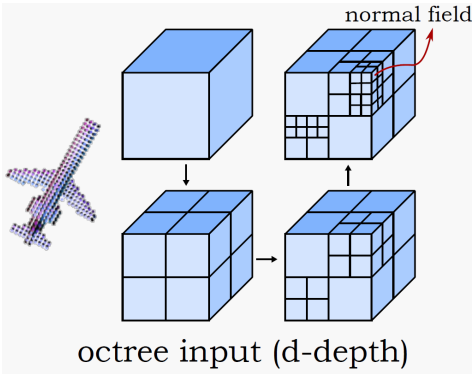


Figure 3.20: An example of 3D voxel space processed by octree [Wan+17] Each single cell is subdivided recursively whenever it contains both empty and full voxels, splitting it into 8 cubes, called the cells ”children”. This process is stopped when all cell’s contain only empty or full voxels (this happens at the latest, when the resolution of the original voxel grid is reached)

Chapter 4

Methodology

This work aims to *1) integrate the proposed Object-Relation module* into existing two-stage 3D object detection baseline models as well as define proper node features to get improved detection performances (Fig.4.2), and *2) substitute 2D convolution layers with Visual State Space block (VSS-Block)* in the baseline models' BEV backbone to reduce model's complexity.

4.1 Preliminaries

We focus on improving the detection performance and computational efficiency of two-stage 3D object detectors. To make the explanation of our work easy to understand, this section provides the fundamental definitions of the terms and their corresponding mathematical expressions as the preliminaries of this work. For a easy understanding to this work, this section provides the fundamental definitions of the terms their corresponding mathematical expressions.

4.1.1 Problem Statement

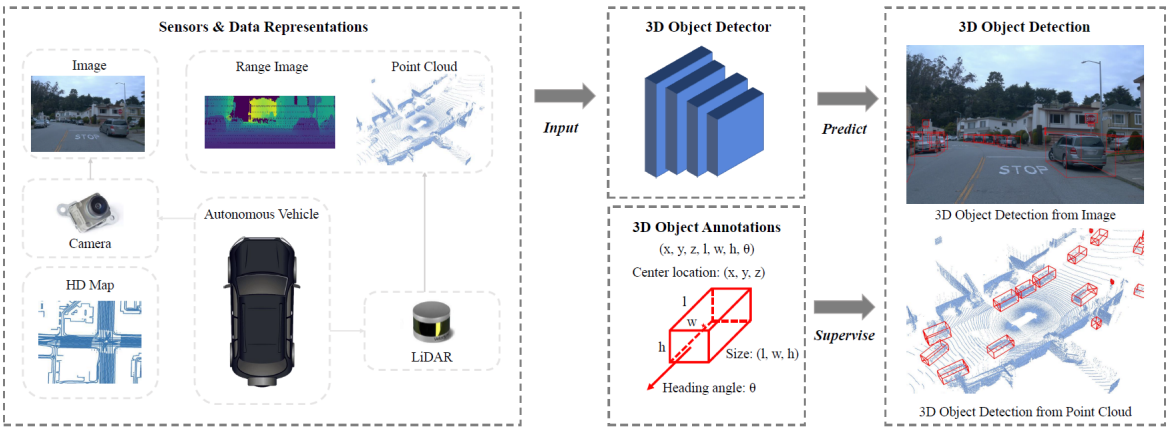


Figure 4.1: Problem statement of 3D object detection in autonomous driving scenarios [Mao+23]

3D object detection aims to identify the locations and classification of other entities within the driving environment. For each single LiDAR record frame, with set of raw point clouds $\mathcal{P} \in \mathbb{R}^{N\times 4}$ consisting of N points as input, the 3D detector returns a set of bounding boxes $\mathcal{B} \in \mathbb{R}^{k\times 8}$ consisting of k detected objects as output. Each single recorded point in point clouds

$\mathcal{P}$ is represented as a 4-D vector $[x, y, z, \text{refl}]$, which indicates the x-y-z position coordinates and the reflection rate $\in [0, 1]$. As the detection result, each single object in $\mathcal{B}$ is represented as a vector of 8 dimension, consisting of one bounding box $\mathbf{b} \in \mathbb{R}^7$ and a classification identity $c \in \mathbb{N}$. The bounding box $\mathbf{b} \in \mathbb{R}^7$ is represented as the combination of box size (h, w, l), center coordinate (x, y, z), and a heading angle θ.

4.1.2 Framework

We denote the set of 3D proposals in a frame generated by the Region Proposal Network (RPN) in the first stage of a 3D detector as $P = \{\mathbf{p}_i | i = 1, \ldots, n\}$, where n is the total number of predicted proposals in a frame. As the indicators of potential existing objects, proposals also consist of the classification $C \in \mathbb{R}^{n \times 1}$ and coarse predicted bounding boxes $B \in \mathbb{R}^{n \times 7}$. After proposals being generated, a Region of Interest (RoI) pooling module combined with a fully connected (FC) layer is used to create the fixed-size feature maps $F = \left\{\mathbf{f}_i \in \mathbb{R}^{1 \times d} | i = 1, \ldots, n\right\}$, where d is the dimension of features. In the end, the feature maps are input to a set of layers to obtain the final predicted bounding boxes $B^{res} = \left\{\mathbf{b}^{res}_{i'} | i' = 1, \ldots, m\right\}$ and their confidence scores $C_{score} \in \mathbb{R}^{1 \times m}$, where m is the number of predicted 3D bounding boxes.

In our approach, we fully use graph information while preserving local object features by incorporating our proposed inter-object relation module after the FC layer. This proposed module generates refined features $F' \in \mathbb{R}^{n \times d'}$ from the RoI feature maps F and the coarse bounding boxes B.

4.2 Object Relation Module

Traditionally, two-stage 3D detectors [Shi+21; Shi+20] process each proposal independently during the refinement stage, ignoring the rich context information included in the object relations. However, learning the inter-object relationship is beneficial for more effective and robust detection, especially for addressing occlusion and predicting object direction. To do so, we introduce an object relation module to enlarge the perception range and capture relation features by leveraging a GNN on the generated inter-object relationship graphs. The overview of our proposed method is illustrated in Fig. 4.2. The details about the Object Relation Module would be explained in this section.

4.2.1 Graph Constructor

We create an inter-object relationship graph based on the predicted proposals P in a point cloud. Creating a fully connected graph among all proposals requires the runtime complexity to $O(n^2)$, where n is the number of proposals. On the other hand, connecting all proposals can make a large receptive field while causing noise and introducing unrelated information from distant objects. Therefore, we consider establishing the graph based on box centroid (x, y, z) within a predefined range. Specifically, we utilize two strategies to construct a graph: k nearest neighbor (KNN), connecting a fixed number of proposals with each other, or a radius graph, connecting all proposals within a certain distance. After graph generation, each proposal connects to its neighbors, as shown in Fig. 4.3.

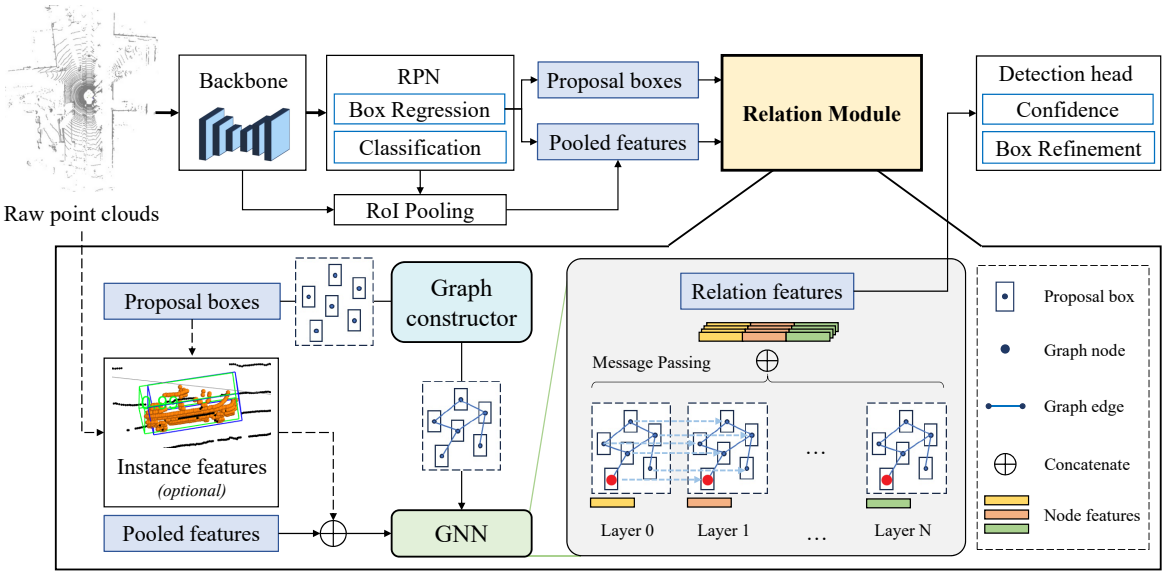


Figure 4.2: Implementation of Object Relation Module. With the two-stage baseline detector, we take the output of the first stage, the proposal boxes and pooled features, as the input of the proposed object relation module. We utilize the center coordinates of proposal boxes to build relation graphs using k nearest neighbor (kNN) method. With this graph, we assign pooled features and proposal box information as node features and feed to GNN to learn relation features through message passing. The learned relation feature from our proposed object relation module is then fed to detector's second stage for box refinement.

4.2.2 Message Passing through GNN

With the generated graph, the node features would be processed by GNN using the methods *message passing* and *edge convolution*. Generally, a process of updating node features via a graph neural network can be demonstrated as:

$$\mathbf{v}_i^{l+1} = f(\mathbf{e}_{ij}^l, \mathbf{v}_i^l) \tag{4.1}$$

where $l \in L$ is the lth layer of the GNN with L layers in total, $\mathbf{v}_i$ is the feature of the ith node, $\mathbf{e}_{ij}$ is the feature of edge (i, j), and $f(.)$ is the operation applying edge features to update the node features. Usually, edge feature $\mathbf{e}_{ij}^l$ can be defined as $\mathbf{e}_{ij}^l = h^l(\mathbf{v}_i^l, \mathbf{v}_j^l)$, where $h^l(.)$ is a function to capture the edge feature.

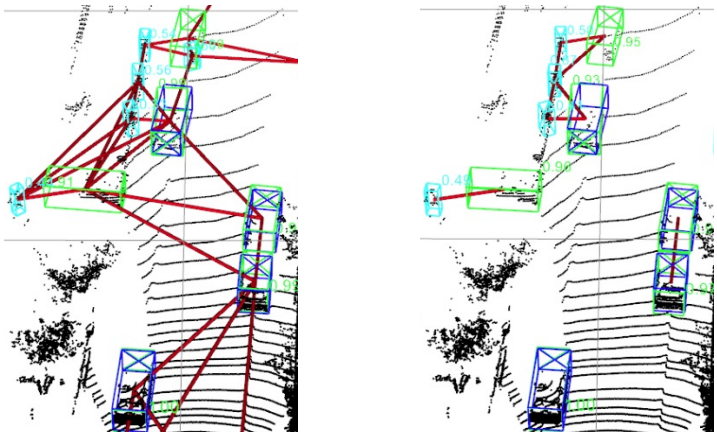
We demonstrate the GNN module we used in our graph relation module in Fig. 4.4. We initialize each node feature $\mathbf{v}_i^0$ by fusing the proposal feature obtained by the FC layer and the corresponding box information $\mathbf{b}_i$ with an MLP layer, as shown in Eq. 4.2. Therefore, each node feature compresses object and spatial features by fusing the proposal features with the proposal box information.

$$\mathbf{v}_i^0 = MLP(\mathbf{f}_i, \mathbf{b}_i) \tag{4.2}$$

We first leverage the differences of node features to obtain the edge features. Encompassing features from neighbor nodes enlarge the receptive field, alleviating occlusion issues and improving the capability to perceive the surrounding environment. Hence, Eq. 4.1 is re-written as:

$$\mathbf{v}_i^{l+1} = g^l(h^l(\mathbf{v}_j^l - \mathbf{v}_i^l, \mathbf{v}_i^l)) \tag{4.3}$$

Additionally, we add the proposal box difference between the neighbor nodes into edge features to utilize the inter-object relations more effectively. The proposal box differences directly reflect the spatial relationships between neighbor objects, allowing the GNN to capture relative positioning, which helps understand the movement patterns of objects and their interaction with each other.



(a) KNN (k=16) **(b)** Radius (r=6m)

Figure 4.3: Example of generated graphs on proposals from the first stage of PV-RCNN [Shi+21]. The left graph was generated based on KNN (K=16), leading every proposal to connect to its 16 nearest neighbors. The right diagram was generated via a radius graph with a threshold of six meters. Ground truth is shown in blue. Dark red lines illustrate the graph edges. Green and cyan represent the predicted cars and pedestrians.

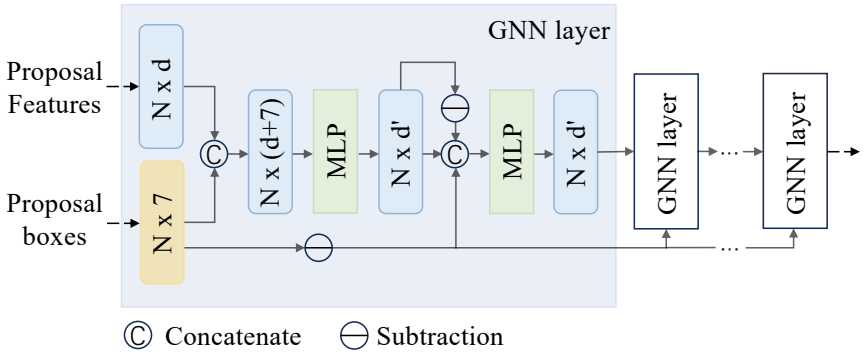


Figure 4.4: Illustration of Graph Neural Network architecture. We utilize the proposal features and boxes as input to initialize the node features of the inter-object relationship graph.

Specifically, we concatenate the proposal box differences and node feature differences to obtain the enhanced edge features. For Eq. 4.3, we conduct the function $h^l(.)$ as MLP to extract edge features, and $g^l(.)$ as max pooling to update the node feature $\mathbf{v}_i^l$ to $\mathbf{v}_i^{l+1}$ with the aggregating edge features. Hence, the final node feature updated by the $l+1$ layer can be represented as:

$$\mathbf{v}_i^{l+1} = \max_{\forall j \in \mathcal{N}(i)} (MLP(\mathbf{v}_j^l - \mathbf{v}_i^l, \mathbf{b}_j - \mathbf{b}_i, \mathbf{v}_i^l)) \tag{4.4}$$

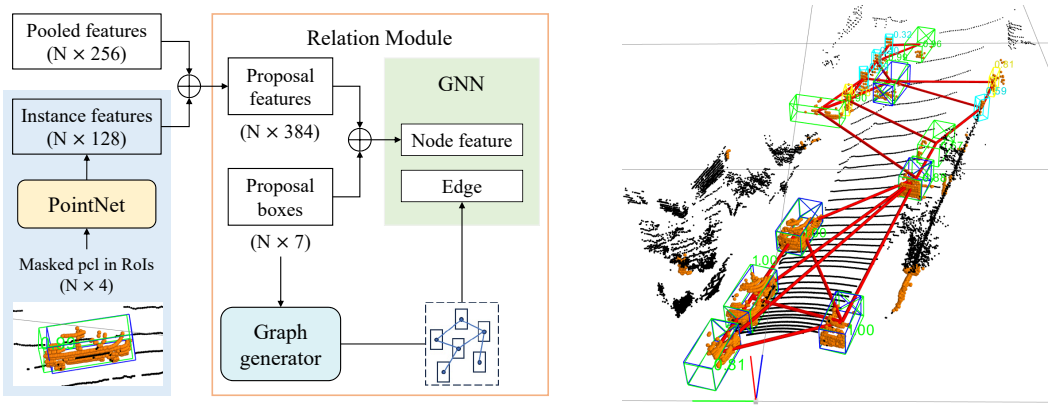
Since a node feature v_i is iteratively updated through multiple GNN layers, it encodes different hidden features after each layer. Hence, to keep all the hidden features learned by the GNN, we combine all the features of the same node from different layers:

$$\mathbf{f}_i' = (\mathbf{f}_i^0, \mathbf{f}_i^1, \ldots, \mathbf{f}_i^L) \tag{4.5}$$

Ultimately, we use the combined node features to input the following classification head to produce the refined object bounding boxes and predicted confidence score.

4.2.3 Instance-level feature extraction

To fully make use of the message passing mechanism of GNN, we consider to extend the node feature to contain as much information about object itself as possible. Inspired by GACE [Sch+23], here we mask the raw point cloud within each proposal box out and feed to a shared PointNet-like encoder to get a *instance feature* for the corresponding potential object. The instance feature would be concatenated to pooled feature from the first stage of baseline detector as the extended proposal feature. The pipeline of instance-level feature extraction and processing is demonstrated in Fig. 4.5.



(a) Pipeline **(b)** Scene visualization

Figure 4.5: Instance-level feature extraction. In (b) visualization, orange spheres are masked point clouds within proposal boxes; red lines are graph edges connecting neighboring objects; blue boxes are ground truth of all classes; the green, cyan and yellow boxes are the proposal boxes of vehicles, pedestrians and cyclists respectively.

4.3 Mamba-based 2D Backbone

Inspired by VMamba [Liu+24c], we decide to use VSS-block to replace the CNN-stack in the 2D-BEV backbone of baseline detectors to reduce computational complexity, meanwhile at least maintain or even improve the detection accuracy. The *Visual State Space block* (VSS-Block) in VMamba was initiated as a vision backbone integrating SSM-based blocks to facilitate efficient visual representation learning. As the core algorithm of Mamba (S6) [GD23], the parallelized *selective scan operation* was essentially designed for processing one-dimensional sequential data. To adapt this to 2D vision data, which inherently lacks a sequential arrangement of visual components, the *2D Selective Scan* (SS2D) was proposed for spatial domain traversal. The details about VSS-Block and SS2D as well as their implementation in 2D-BEV backbone in our 3D detector would be covered in this section.

4.3.1 Baseline 2D-BEV Backbone

The original 2D-BEV backbone of both PV-RCNN and PartA2 models consists of stacks of CNN layers as feature encoder. The input BEV feature map $\mathcal{F}^{\text{BEV}} \in \mathbb{R}^{C \times H \times W}$ is processed by two stages, each starts with a down-sampling convolution layer, which adjusts feature's spatial sizes and feature dimensions, and then followed by 5 subsequent CNN layers, where kernel size equals 3, padding is 1×1 and stride equals 1. This stack of CNN-layers only extract latent

information but would not change feature's dimensions. After this, the outputs of these two stages would be fed to up-sampling and be recovered into the original dimensions, i.e. the input BEV feature map's dimensions. These two hidden feature out of the U-Net like encoder would be concatenated as the resultant output of 2D-BEV backbone $\mathcal{F}_{out}^{\text{BEV}} \in \mathbb{R}^{2C \times H \times W}$.

4.3.2 Implementation of VSS-Block

With the newly proposed 2D-Selective-Scan (SS2D) module, the VSS block has successfully expand the application of selective state space method (S6), which stands as the core of Mamba in concurrently achieving global receptive fields, dynamic weights (*i.e.*, selectivity), and linear complexity, to the field of vision tasks.

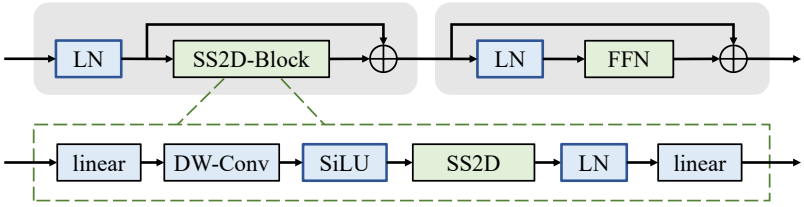


Figure 4.6: Visual State Space (VSS) block: *LN*: Layer normalization; *DW-Conv*: Depth-wise convolution; *FFN*: Feed-forward Network, in the implementation we use an MLP as FFN.

As shown in Fig. 4.6, VSS block starts with an initial linear embedding layer, and the output splits into two information flows. One flow passes through a 3×3 depth-wise convolution layer, followed by a Silu activation [Sha20] before entering the core SS2D module. The output of SS2D goes through a layer normalization layer and is then added to the output of the other information flow, which has undergone a Silu activation. This combination produces the final output of the VSS block. The design of VSS block diverges from the typical vision transformer structure, which employs the following sequence of operations: LayerNorm $\rightarrow$ Attention $\rightarrow$ LayerNorm $\rightarrow$ MLP in a block, and discards the MLP operation. This leads to a shallower architecture and allows to stack more blocks with a similar budget of total model depth.

Regardless of the patching partition stem at the beginning of the entire VMamba architecture, we directly take the 2D BEV feature map as embedded feature and feed to VSS block. For VSS block, only the feature dimension should be defined. The VSS block does not change the number of feature channels, which is of the same behavior as the 5-layer CNN-based encoder. Therefore, we decide to only use one single VSS block to substitute the CNN-based encoder in the baseline 2D backbone, and keep the original convolution layers and trans-convolution layers for down-sampling and up-sampling. The implementation is shown in Fig. 4.7.

In our implementation, we keep the overall architecture unchanged. With the input BEV feature map of shape $(C = 256, H, W)$, in *Stage I*, the down-sampling *Conv* is configured with input channel of 256, output channel 128, kernel size 3×3, stride 1×1, zero-padding 1×1, which only reduces feature channels from $C = 256$ to $C/2 = 128$ and keeps spatial size H, W unchanged. In *Stage II*, the down-sampling *Conv* is configured with input channel of 128, output channel 256, kernel size 3×3, stride 2×2, zero-padding 1×1, which doubles feature channels from $C/2 = 128$ to $C = 256$ and reduces spatial size H, W to $H/2, W/2$. Both *Stage I* and *Stage II* contain a corresponding *Trans-Conv* to up-sample these two middleware feature maps with shapes $(128, H, W)$ and $(256, H/2, W/2)$ back to origin shape $(256, H, W)$, and finally concatenate them to a $(512, H, W)$-shaped feature map as the output of 2D back-

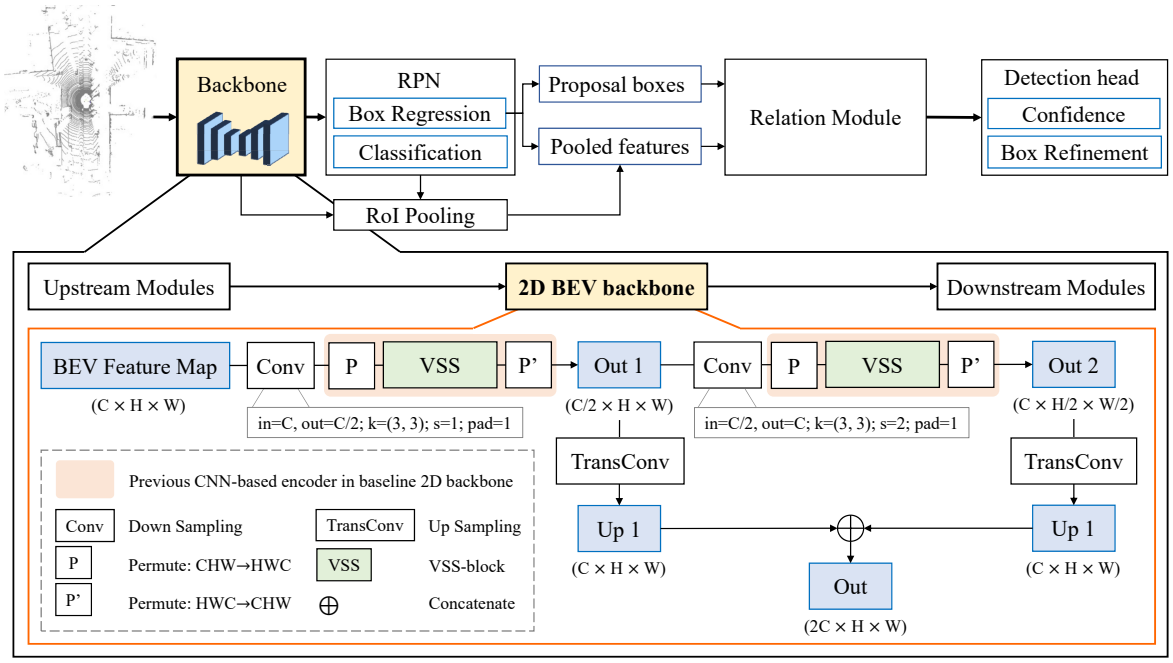


Figure 4.7: Implementation of VSS-block in 2D backbone. We use one VSS-block to substitute the 5-layer 3×3 convolution stack in the feature encoder of original 2D backbone, while the down-sample convolution layers and up-sample trans-convolution layers are retained.

bone. The complete block structures of *Mamba-based 2D backbone* (Lst. A.1) is provided in appendix.

$$F \in \mathbb{R}^{(C_h,H,W)} \xrightarrow{\text{Permute}} F_{in} \in \mathbb{R}^{(H,W,C_h)} \xrightarrow{\text{VSS-block}} F_{out} \in \mathbb{R}^{(H,W,C_h)} \xrightarrow{\text{Permute}} F \in \mathbb{R}^{(C_h,H,W)} \tag{4.6}$$

Since VSS-block takes channel-last shaped tensor, the incoming feature map should be permuted from (C,H,W) to (H,W,C) first, and the outcome of VSS-block should be permuted back to (C,H,W) to make it feasible for down-stream modules (Eq. 4.6). Within *Stage I* and *Stage II*, the VSS-blocks are configured with respective hidden dimensions $C_h^{l,l=1,2}$, i.e. $C_h^1 = 128$, $C_h^1 = 256$. Each VSS-block starts with a LayerNorm with the defined hidden dimension C_h, then comes to the SS2D-block. SS2D-block starts with an *out-norm* layer, executing layer normalization with dimension $2 \cdot C_h$. Then comes to *input-projection*, a linear layer converting the number of channels from C_h to $4 \cdot C_h$. Then it comes to *input-activation*, a SiLU activation and the following *SS2D*, a 2D convolution layer with input dimension $2 \cdot C_h$ and output dimension $2 \cdot C_h$, kernel size 3×3, stride 1×1 and padding 1×1. The following *output activation* is defined as Identity(), *output projection* as a linear layer converting the number of channels $2 \cdot C_h$ to C_h as the end of SS2D-block. After being concatenated with the feature map before SS2D-block, the outcoming feature map is then fed to subsequential *Layer-Norm* and an MLP-based Feed-forward net as shown in Fig. 4.6. The complete block structures of VSS-blocks for *Stage I* (Lst. A.2) and *Stage II* (Lst. A.3) are provided in appendix.

4.4 Training loss

Our proposed object relation module does not introduce external loss calculation. Therefore, in our experiments, we follow the default loss function of PV-RCNN and PartA2 to conduct experiments. Specifically, the loss function of PV-RCNN consists of region proposal loss L_{rpn}, keypoint segmentation loss L_{seg}, and the proposal refinement loss L_{rcnn}.

The L_{rpn} and L_{rcnn} are defined as Eq. 4.7 and Eq. 4.8, and L_{seg} is calculated by the focal loss [Lin+18].

$$L_{rpn} = L_{cls} + \beta \sum \mathcal{L}_{smooth-L1}(b - b^{gt}) \tag{4.7}$$

$$L_{rcnn} = L_{iou} + \sum \mathcal{L}_{smooth-L1}(b^{res} - b^{gt}) \tag{4.8}$$

where b, b^{res}, and b^{gt} represent the predicted proposals from the first stage, the final predicted bounding boxes results, and the ground truth boxes, respectively. The calculation for L_{cls} is also based on focal loss, and the L_{iou} is calculated by the binary cross entropy loss:

$$L_{iou} = -\frac{1}{K}\sum_{k=1}^{K}(\hat{y}_k \log(p(\hat{y}_k) + (1 - \hat{y}_k)\log(1 - \hat{y}_k)) \tag{4.9}$$

where k is the k-th RoI in terms of the corresponding ground truth, and $\hat{y}_k$ is the predicted classification score.

For PartA^2, the overall loss function is shown as:

$$L_{total} = L_{aware} + L_{aggregation} \tag{4.10}$$

where L_{aware} (Eq. 4.11) is the loss of part-aware stage-I, considering foreground point segmentation, regression of part locations, and 3D proposal generation. $L_{aggregation}$ (Eq. 4.12) is the part aggregation loss.

$$L_{aware} = L_{eg} + \frac{1}{N_{pos}} L_{part} + \lambda \frac{1}{M_{pos}} L_{box} \tag{4.11}$$

where λ is the loss weight, N_{pos} and M_p represent the total number of foreground points and the number of positive anchors for PartA^2-anchor model.

$$L_{aggregation} = L_{score} + \frac{1}{T_{pos}} L_{box\_refine} \tag{4.12}$$

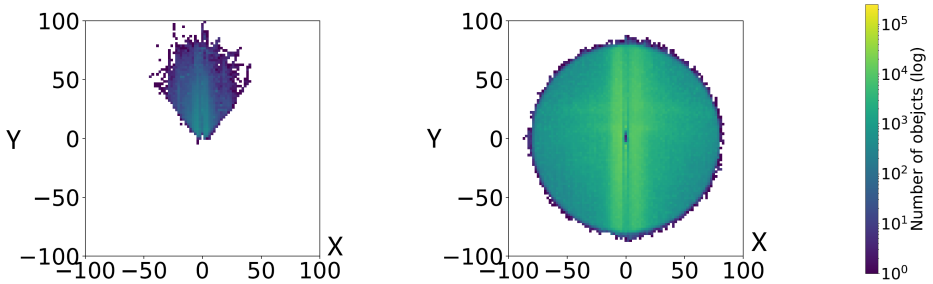
where L_{box_refine} is the residual-based box regression loss, L_{score} is the binary cross entropy loss, and T_{pos} is the number of positive proposals. Due to the page limitation, more details of the loss functions are explained in [Shi+20].

Chapter 5

Experiments and Results

This chapter gives the experiment setups and implementation results of object relation and Mamba-based backbone on the baseline models PV-RCNN and PartA2. We used KITTI dataset and Wymo open dataset for training. The detection performance is evaluated upon *KITTI metrics* and the computation efficiency is evaluated via *FLOPs* and *Params*.

5.1 Datasets



(a) KITTI **(b)** Waymo

Figure 5.1: Bird's-Eye View object distribution of datasets. Each heatmap represents a dataset and is plotted using X and Y coordinates. X is the latitudinal and Y is the longitudinal direction (i.e., driving direction) of the ego-vehicle. The unique annotation characters of each dataset are reflected in the distribution range, density, and number of bounding boxes. The figures are from [Liu+24b].

The rapid advancements in autonomous driving rely heavily on extensive datasets, which help autonomous driving systems be robust and reliable in complex driving environments. The two categories of autonomous driving datasets upon collection strategies are *synthetic datasets*, which are generated by simulators and *real-world datasets*, which are recorded from open roads and streets. Secondly, the datasets vary in composition, including but not limited to multiple sensor modalities like camera images and LiDAR point clouds, different annotation types for various tasks, and data distribution. The two datasets used in this work, KITTI and Waymo, are both real-world datasets. Fig. 5.1 depicts the comparison of the 3D object bounding box distribution of KITTI and Waymo datasets under a Bird's-Eye View (BEV). To evaluate datasets, data dimension could reflect a general data quality. This includes the perspectives of data size, temporal information, task number, and labeled categories. To make sure the recorded scenes to cover as much corner cases in reality as possible, environmental diversity should be considered, in the perspectives of 1) weather conditions (such as rain or snow), 2) time of day (e.g., morning or dusk), 3) types of driving scenarios (like urban or

rural), and 4) geometric scope referring to the number of countries or cities where the data is recorded. Tab. 5.1 shows the detailed comparison in respect of data dimension and sensor configuration, etc.

Table 5.1: Comparison of KITTI dataset and Waymo Open Dataset detailed comparison in respect of data dimension and sensor configuration, etc.

	KITTI	Waymo
Scenes	22	1150
Ann. LiDAR Fr.	15K	230K
Hours	1.5	6.4
3D Boxes	80K	12M
2D Boxes	80K	9.9M
LiDARs	1	5
Cameras	4	6
Avg. Points/Fr.	120K	177K
LiDAR Features	1	2
Maps	No	Yes
Released Year	2013	2020

5.1.1 KITTI Dataset

KITTI dataset [Gei+13] provides benchmarks for the tasks of stereo, optical flow, visual odometry-SLAM, and 3D object detection. The 3D object detection benchmark consists of 7481 training images and 7518 test images as well as the corresponding point clouds, comprising a total of 80.256 labeled objects. It was created with the aim to be highly realistic for autonomous driving and tried to close the gap between models performing well on earlier datasets but subsequently bad in real-world scenarios. KITTI is evaluated on the mAP of 40 thresholds (R40) dynamically chosen based on the model's detection. It comes with three classes (car, pedestrian, cyclist) and three difficulties (easy, moderate, hard). The sensor setup of KITTI is illustrated in Fig. 5.2.

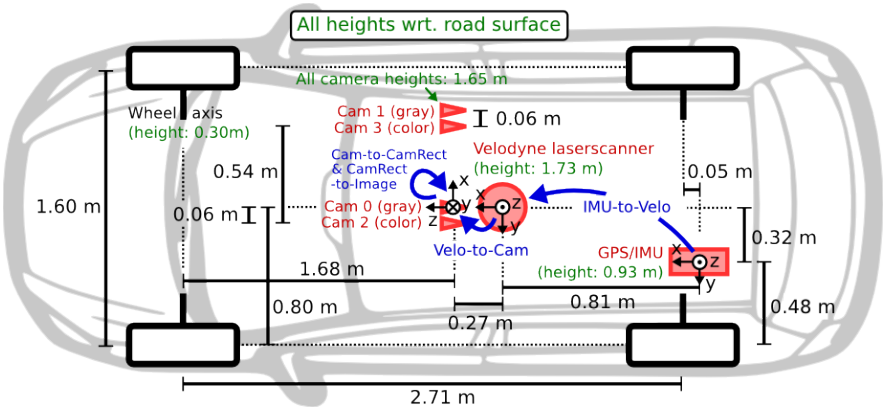


Figure 5.2: Layout of KITTI sensor setup. [Gei+13]

5.1.2 Waymo Open Dataset

The Waymo open dataset [Sun+20] is a much larger dataset that tries to provide diverse data from a variation of environments. It consists of 1150 scenes that each span 20 seconds, consisting of high-quality LiDAR and camera outputs captured across a range of urban and suburban geographies. This includes 158,361 samples in training set, 40,077 samples in validation set and 39,987 samples in test set. All the sensor data is recorded with an industrial-strength sensor suite consisting of multiple high-resolution cameras and multiple high-quality LiDAR sensors. Furthermore, Waymo offers synchronization between the camera and the LiDAR readings, which offers interesting opportunities for cross-domain learning and transfer. The LiDAR sensor readings are released in the form of range images. In addition to sensor features such as elongation, Waymo provides each range image pixel with an accurate vehicle pose. This is the first dataset with such low-level, synchronized information available, making it easier to conduct research on LiDAR input representations other than the popular 3D point set format. The sensor setup of Waymo is illustrated in Fig. 5.3.

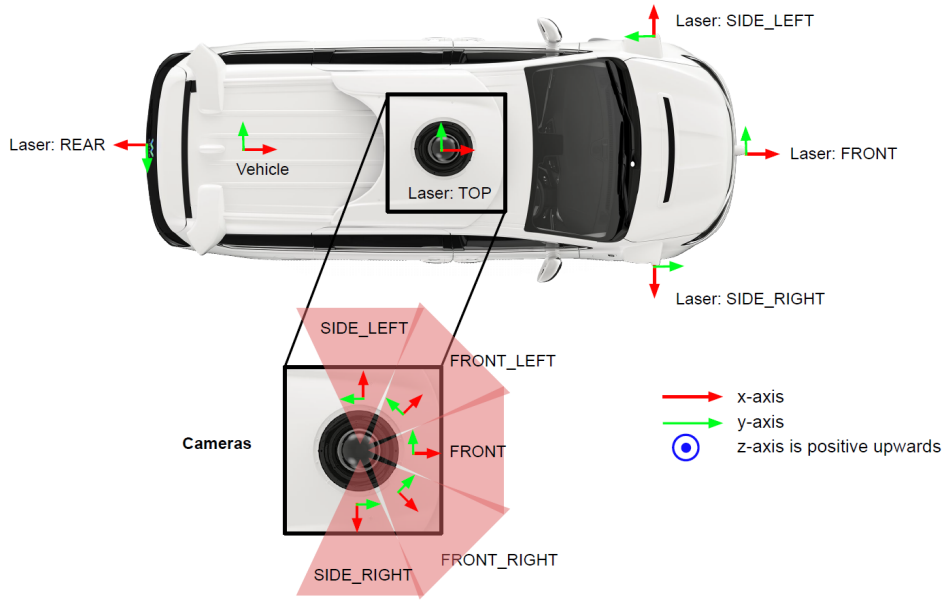


Figure 5.3: Layout of Waymo sensor setup. [Sun+20]

5.2 Evaluation Metrics

This section explains KITTI metrics and Waymo metrics for the evaluation of detection accuracy and the concepts of FLOPs and Params for the evaluation of computational cost.

5.2.1 KITTI Metrics

In KITTI benchmark, 3D object detection performance is evaluated using the PASCAL criteria, which is also used for 2D object detection. Far objects are thus filtered based on their bounding box height in the image plane. As only objects also appearing on the image plane are labeled, objects in don't care areas do not count as false positives.The evaluation does not take care of ignoring detections that are not visible on the image plane — these detections

might give rise to false positives. For *cars* a 3D bounding box with overlap of 70% is required, while for *pedestrians* and *cyclists* overlap of 50% required. Difficulties are defined as follows:

Table 5.2: Definition of KITTI difficulty levels.

	Min. BBox Height	Occlusion	Max. Truncation %
Easy	40 px	Fully visible	15%
Moderate	25 px	Partly occluted	30%
Hard	25 px	Difficult to see	50%

As already discussed in section 2.6, *Average Precision* (AP) is calculated using area under the curve (AUC) of the *Precision - Recall* curve. As AP curves are often zigzag curves, comparing different curves in the same plot usually is not an easy task. In practice AP is the precision averaged across all recall values between 0 and 1. KITTI benchmark provides AP of *bbox*, *bev*, *3D* and *aos* at recall of 11 (R11) and 40 (R40) as evaluation results. In this work, we report AP of *3D* (in text) and *BEV* (in appendix) at recall of 40 (R40) as the qualitative results.

5.2.2 Waymo Metrics

Waymo Open Dataset [Mei+22] provides three classes and each is evaluated with mAP, mAPH (*mean Average Precision High*) measuring the performance on larger objects closer to the observer and mAPL (*mean Average Precision Low*) measuring objects further away. The objects are categorized into two classes LEVEL_1, the easy detections, and LEVEL_2, the more difficult detections.

5.2.3 Computational Efficiency and Complexity

We use *FLOPs* (Floating Point Operations) [DMH23] to indicate computational efficiency and *Params*, the number of parameters to indicate the model's complexity. When dealing with computing effort and computing speed (hardware performance), terminology is usually confusing. For instance, the term 'compute' is used ambiguously, sometimes applied to the number of operations or the number of operations per second. However, it is important to clarify what kind of operations and the acronyms for them. In this regard, the acronym *FLOPS* would be used to measure hardware performance, by referring to the number of floating point operations *per second*, as standardised in the industry, while *FLOPs* will be applied to the amount of computation for a given task (e.g., a prediction or inference pass). To note that in consideration of the feasibility to the hardware end, PyTorch Op-Counter provides a built-in function to report the number of MACs (Multiply-Accumulate Operations) and Params for DNN models [NVI15]. More specifically, one fused multiply-add operation could be counted as 2 FLOPs. So we get FLOPs by multiplying MACs with 2, and get Params directly from Op-Counter.

5.3 Experiment Setups

We conduct all experiments using the OpenPCDet [1], an open source project for LiDAR-based 3D object detection based on PyTorch. OpenPCDet provides a standardized workflow

[1] https://github.com/open-mmlab/OpenPCDet

(Fig. 5.5) for various 3D object detection approaches and organizes these models in form of structured modules (Fig. 5.4) for efficient code re-use and convenient extension and modification.

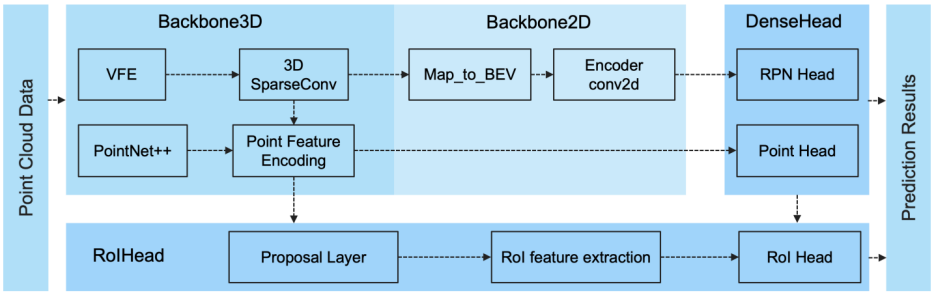


Figure 5.4: OpenPCDet Framework: Flexible and clear model structure to easily support various 3D detection models.

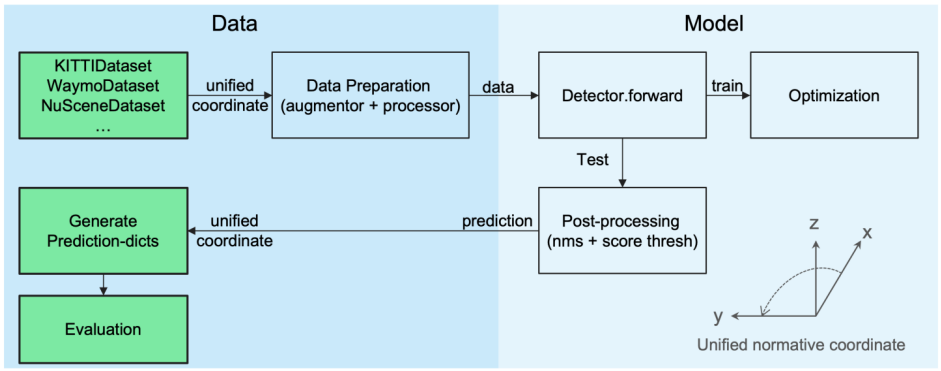


Figure 5.5: OpenPCDet Workflow: Data-Model separation with unified point cloud coordinate for easily extending to custom datasets.

Dataset Splits and GPU Resources The experiments on KITTI dataset are conducted on a NVIDIA GeForce 3090 (24GB) GPU, using the common train-val split with 3712 training and 3769 validation samples to train and evaluate. The experiments on Waymo open dataset are conducted on a NVIDIA GeForce 4080 (16GB) GPU, using a 20% subset of training set containing 31618 samples, and a full validation set containing 39987 samples to conduct the experiments.

Network Architecture We use PV-RCNN [Shi+21] and PartA2 [Shi+20] as the baseline models and implement our proposed object relation module in them. For both PV-RCNN and PartA2, we keep using the same model architectures reported in [Shi+21] and [Shi+20], respectively. For the proposed object relation module, we apply KNN (k = 16) as the graph generator for the main experiments and use a four-layer GNN to extract features. To keep the feature dimension alignment, we set the input and output of the GNN module to 256, the same as the dimensions of the pooled features and the input features of the detection head of the two-stage 3D detectors. For the Mmaba-based 2D backbone, we set the hidden dimensions of the two VSS-blocks as 128 and 256 respectively.

Hyperparameters For all experiments, we set the batch size to 2, using the Adam OneCycle optimizer with an initial learning rate of 0.01. All models on KITTI dataet are trained for 80

epochs, Waymo 30 epochs, and we report the best result of each experiment in the experiment tables.

Training Loss Our proposed object relation module does not introduce external loss calculation, making our module easily adapted to any other two-stage detectors. Therefore, in our experiments, we follow the default loss function of PV-RCNN and PartA2 as shown in section 4.4 to conduct experiments.

Data Augmentation We utilize four widely adopted data augmentation strategies in our experiments: 1) Ground truth sampling [YML18], randomly selecting several ground truths from other scenes and adding them into the current frame. 2) Random flipping along the x-axis. 3) Random rotation within the angle range $[-\frac{\pi}{4}, \frac{\pi}{4}]$ around the z axis. 4) Random scaling with a scaling factor within the range $[0.95, 1.05]$.

5.4 Experimental Results

We conduct several experiments to show the efficacy of 1) inter-object relation module in improving detection performance and 2) Mamba-based backbone in saving computational cost for two-stage 3D object detectors.

5.4.1 Experimental results on KITTI

All Classes

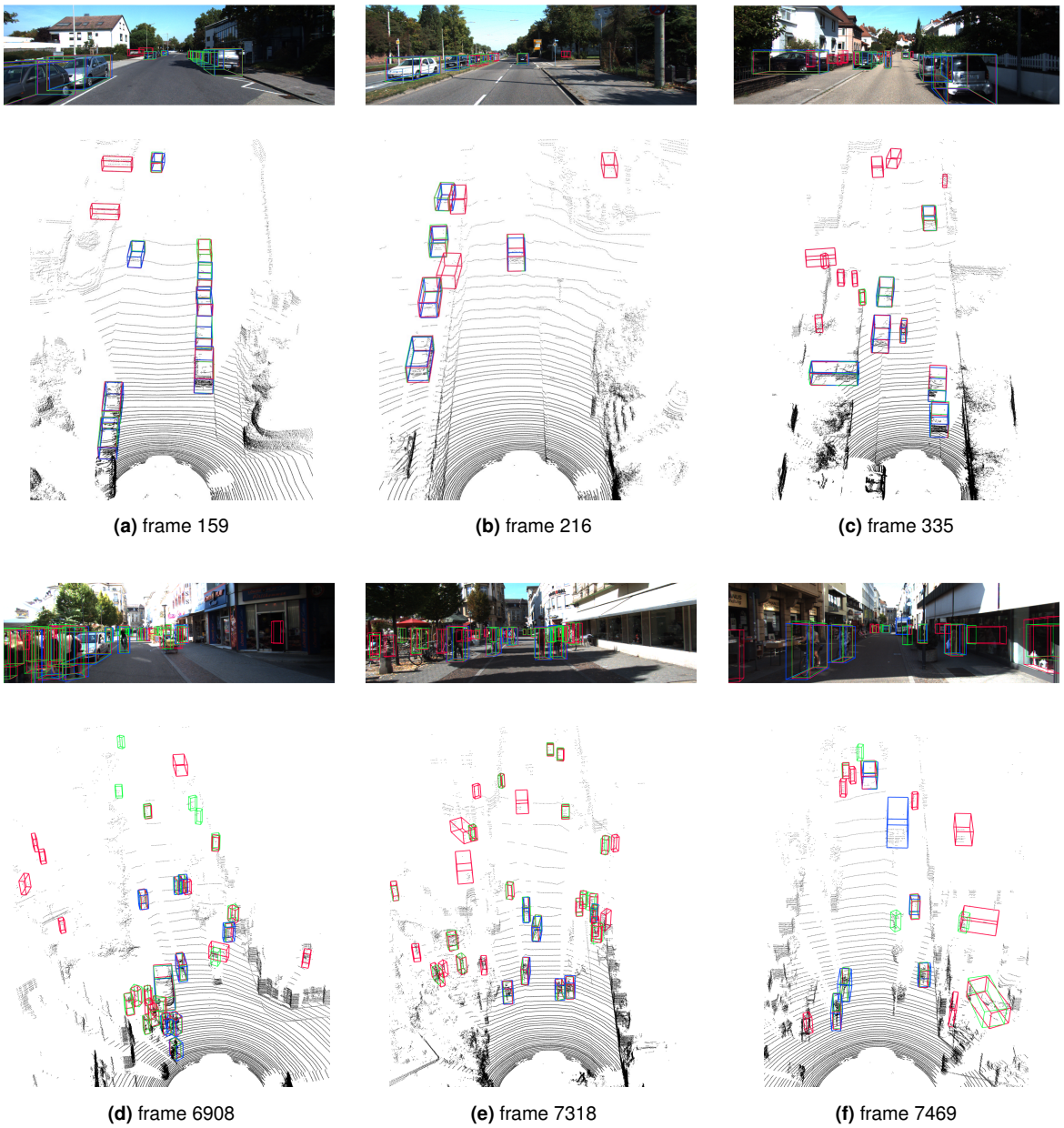
We first train PV-RCNN [Shi+21] baseline and PV-RCNN_Rel_Mamba, the model implemented with our proposed relation module and Mamba-based 2D backbone, on the KITTI training set and evaluate the prediction results on the KITTI validation set. The statistics of 3D AP_{R40} is reported in this section, and the BEV AP_{R40} is reported in appendix.

Table 5.3: Comparison of the baseline models (PV-RCNN [Shi+21] and PartA2 [Shi+20]) with our proposed object relation module and Mamba-based 2D backbone. Results reported on the KITTI [Gei+13] **validation set** in terms of 3D AP_{R40}. For the car, pedestrian and cyclist categories, the IoU thresholds are 0.7, 0.5 and 0.5, respectively.

Method	Car 3D AP 0.7 (%) ↑			Pedestrian 3D AP 0.5 (%) ↑			Cyclist 3D AP 0.5 (%) ↑		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
PV-RCNN	91.87	84.53	82.41	**65.38**	**57.99**	**53.17**	89.95	70.81	66.37
PV-RCNN_Rel_Mamba	**92.41**	**85.27**	**82.93**	62.86	56.17	50.95	**92.02**	**74.68**	**69.90**
Improvements	+0.54	+0.74	+0.52	-2.52	-1.82	-2.22	+2.07	+3.87	+3.53
PartA2	91.84	**82.53**	80.17	**64.13**	**58.04**	**53.08**	88.24	71.12	66.76
PartA2_Rel_Mamba	91.84	82.48	**80.35**	59.94	51.82	47.39	**88.36**	**71.54**	**68.22**
Improvements	+0.00	-0.05	+0.18	-4.19	-6.22	-5.69	+0.12	+0.42	+1.46

In Tab. 5.3, we demonstrate the 3D AP results of the baseline PV-RCNN and ours PV-RCNN_Rel_Mamba. For the car category, our PV-RCNN_Rel_Mamba network outperforms the baseline with accuracies of 92.41%, 85.27%, and 82.93% in terms of the 3D AP for car class under easy, moderate, and hard difficulties. For cyclist category we achieve more prominent improvements, getting accuracies and absolute improvements of 92.02% (+2.07%), 74.68% (+3.87%), and 69.90% (+3.53%) in terms of 3D AP under easy, moderate and hard difficulties. Yet for pedestrian category our implementation gets decrement in 3D AP than the baseline. Additionally, we also report the experiment results based on PartA2 [Shi+20]

in Tab. 5.3. In general, our implementation for PartA2 exhibits a similar capability as in PV-RCNN. Our proposed PartA2_Rel_Mamba shows improvement in car class under hard-difficulty, decrement in pedestrian class and significant improvement in cyclist class. The BEV AP result Tab. A.1 in appendix shows similar behavior, the implementation of relation module and Mamba-based 2D backbone brings improvement of $+0.8\% \sim +2.4\%$ for car class, a decrement of -4% $\sim$ -6.9% for pedestrian class and a significant improvement of $+3.5\% \sim +4.7\%$ for cyclist class.



(a) frame 159 **(b)** frame 216 **(c)** frame 335

(d) frame 6908 **(e)** frame 7318 **(f)** frame 7469

Figure 5.6: Qualitative results on KITTI **validation set**. We demonstrate detection results from six scenarios. **Ground truths**, the results of **baseline**, and **ours** are shown in blue, red, and green, respectively. By learning similar movement patterns and extracting relation information, our approach significantly reduces false positives and improves predicted rotation accuracy. Zoom in for more details.

For car class, the improvements are attributed to the contextual patterns learned by our graph relational module, which effectively mitigates occlusion issues, boosting detection

accuracy. Besides, Mamba-based encoder in 2D backbone also has a better ability on latent feature representation than CNN-based encoder. **For cyclist class**, the detection performance is promisingly optimized by the proposed object relation module and Mamba-based 2D backbone. In most cases, only a few cyclists are typically present in a scene (Fig. 5.6 (c)), which can be easily obscured by vehicles or pedestrians. By establishing object relationships with nearby proposals, a broader receptive field and object relation are created. It enables the network to effectively capture the distinctive movement patterns of cyclists, thereby enhancing detection accuracy. **For pedestrian class**, the performance decrement might due to the lack of pedestrian samples in the KITTI dataset. We observe that for baseline model the accuracy on pedestrian is already lower than other two categories, which might because the model is easy to be overfitted on car and cyclist when trained on KITTI dataset. And thus the involved relation module encourages this overfitting, leading to a much worse detection accuracy on pedestrian category.

In order to further analyse the individual contributions of object relation module and Mamba-based 2D backbone to the performance improvements, we conduct ablation study in section 5.4.4.

Car Only

Table 5.4: Comparison with other state-of-the-art methods, the baseline models (PV-RCNN [Shi+21] and PartA^2 [Shi+20]), and ours on the KITTI [Gei+13] **validation set** for car class using 3D AP_{R40} with IoU threshold 0.7.

Method	Car - 3D AP (%) ↑			Car - BEV AP (%) ↑		
	Easy	Mod.	Hard	Easy	Mod.	Hard
SECOND [YML18]	87.43	76.48	69.10	89.96	87.07	79.66
CIA-SSD [Zhe+21]	90.04	79.81	78.80	-	-	-
SSL Point-GNN [Erç+22]	91.43	82.85	80.12	93.55	89.79	87.23
PointRCNN [SWL19]	88.88	78.63	77.38	-	-	-
PartA2	92.28	82.70	80.41	93.33	**89.69**	88.40
PartA2_Rel (ours)	**92.53**	**83.15**	**80.88**	**95.89**	89.45	**89.19**
PartA2_Rel_Mamba (ours)	89.62	82.35	80.39	93.45	88.96	86.94
PV-RCNN	91.91	84.78	82.63	93.17	90.72	88.73
PV-RCNN_Rel (ours)	**92.73**	**85.52**	**83.21**	**95.48**	**91.25**	89.09
PV-RCNN_Rel_Mamba (ours)	92.12	83.12	82.81	93.43	91.04	**89.16**

Since the detection of car class is more concerned than other two classes for our work, to further validate the effectiveness of the proposed object relation module, we conduct training and evaluation on the KITTI data using only car class. We report the experiment results of 3D AP and BEV AP in Tab. 5.4, comparing the baseline model, the model with only object relation module , and the model with both object relation module and Mamba-based 2D backbone. The best results among these three are showed in bold text and the second best as underlined. Additionally, we also include the reported results of some other related works as a reference.

Compared to the baseline PV-RCNN, our PV-RCNN_Rel_Mamba increases the 3D AP by 0.21% and 0.18% on easy and hard difficulty levels, respectively. For the BEV AP, we achieve improvements of 0.26%, 0.32%, and 0.43%. For PartA2, our PartA2_Rel_Mamba shows decrement in detection accuracy, while the PartA2_Rel without Mamba-based 2D backbone brings improvements over baseline model in general. We also observe that the PV-RCNN_Rel

achieves better performance than PV-RCNN_Rel_Mamba. We infer that when dealing with big-sized objects distant from foreground, Mamba-based 2D backbone could not learn enough latent feature due to the objects' incomplete representation.

The improvements provided by object relation module are critical for the driving safety of autonomous vehicles. We exhibit the qualitative results in Fig. 5.6. The selected scenarios include cars parking on the roadside (Fig. 5.6 (a)) or driving parallel (Fig. 5.6 (b)), which are suitable for verifying our hypothesis of exploiting certain patterns to improve object detection. Our object relation module successfully reduces the two false positives shown in the top right of Fig. 5.6 (a), which parks horizontally, while the true positives park along the driving direction. Fig. 5.6 (b) further verifies the effectiveness of our method.

5.4.2 Experimental Results on Waymo

Since the PV-RCNN_Rel out comes with a drastic worse result on Waymo dataset in comparison to the baseline PV-RCNN (-5% in vehicle, -10% in pedestrian and -15% in cyclist class), we only investigate the effectiveness of Mamba-based 2D backbone by reporting PV-RCNN versus PV-RCNN_Mamba and PV-RCNN_Rel versus PV-RCNN_Rel_Mamba separately.

Table 5.5: Comparison of the baseline model PV-RCNN [Shi+21] and the PV-RCNN_Mamba with our proposed substitution of Mamba-based 2D backbone. Trained with a 20% split of Waymo [Mei+22] training set, results reported on the full Waymo **validation set** in terms of L1_mAP, L1_mAPH, L2_mAP, and L2_mAPH.

Methods	Vehicle (%) ↑			
	Level 1 mAP	Level 1 mAPH	Level 2 mAP	Level 2 mAPH
PV-RCNN	75.19	74.53	66.59	65.99
PV-RCNN_Mamba	**76.31**	**75.71**	**67.79**	**67.25**
Improvements	+1.12	+1.18	+1.20	+1.26
	Pedestrian (%) ↑			
	Level 1 mAP	Level 1 mAPH	Level 2 mAP	Level 2 mAPH
PV-RCNN	71.46	34.88	62.51	30.53
PV-RCNN_Mamba	**74.00**	**37.27**	**65.33**	**32.93**
Improvements	+2.54	+2.39	+2.82	+2.40
	Cyclist (%) ↑			
	Level 1 mAP	Level 1 mAPH	Level 2 mAP	Level 2 mAPH
PV-RCNN	50.54	**30.46**	48.61	**29.30**
PV-RCNN_Mamba	**50.92**	28.22	**48.98**	27.15
Improvements	+0.38	-2.24	+0.37	-2.15

As shown in in Tab. 5.5, except the mAPH in two difficulty levels for cyclist class, PV-RCNN_Mamba shows improvements in detection accuracy over the baseline PV-RCNN, i.e. for vehicle class around +1.20%, for pedestrian +2.40% ∼ +2.80% and for cyclist a slight improvement of beneath +0.5%. The comparison of PV-RCNN_Rel and PV-RCNN_Rel_Mamba illustrated in Tab. A.2 (see in appendix) also shows improvement brought by Mamba-based 2D backbone. For vehicle class the improvement counts around +4.5%, for pedestrian +9% for mAP and +3% for mAPH, and for cyclist a way more significant improvement of around +10%. Additionally, the point cloud scenes of PV-RCNN and PV-RCNN_Rel are provided in Fig. A.2 (side view) and Fig. A.1 (bird-eye-view) in appendix as a qualitative investigation for object relation module's effect on Waymo dataset.

5.4.3 FLOPs and Params

Tab. 5.6 shows FLOPs and Params of listed model configuration. We observe that substituting CNN-based encoder with Mamba-based VSS block in 2D backbone reduces FLOPs and Params significantly and brings save on computational cost.

Table 5.6: Comparison of FLOPs and Params of 4 model configurations for the two baseline models (PV-RCNN [Shi+21] and PartA2 [Shi+20]). Here we use avg. 3D AP across all classes and difficulties as a reference of model performance. We also report the modified models' absolute increments (green) and/or decrements (red) in average 3D AP in comparison to the baseline model, and the relative decrement (green) in FLOPs and Params. of models applied with Mamba-based 2D backbone relative to those with original CNN-based 2D backbone.

Method	avg. 3D AP@R40 (%) ↑		FLOPs (G) ↓		Params (M) ↓	
PV-RCNN (*Baseline*)	73.61		178.5		12.41	
PV-RCNN_Mamba	**74.86**	(+1.25)	110.1	(−38.3%)	9.87	(−20.5%)
PV-RCNN_Rel	73.99	(+0.38)	180.3		13.47	
PV-RCNN_Rel_Mamba	74.13	(+0.52)	111.9	(−37.9%)	10.94	(−18.8%)
PartA2 (*Baseline*)	72.88		165.1		61.60	
PartA2_Mamba	**74.63**	(+1.75)	96.7	(−41.4%)	59.11	(−4.0%)
PartA2_Rel	70.02	(−2.86)	168.7		62.71	
PartA2_Rel_Mamba	71.33	(−1.55)	100.2	(−40.6%)	60.22	(−4.0%)

5.4.4 Ablation Studies

In this section, we analyze the effectiveness of each mechanism in the inter-object relation architecture by extensive ablation experiments. We conduct all model training in ablation studies on the KITTI training set and evaluate them on the validation set.

Effect of Mamba-based 2D Backbone

For each baseline model, four configurations are taken for comparison: 1) baseline model without any modofications; 2) baseline + Mamba-based 2D backbone; 3) baseline + object relation module; 4) baseline + object relation module + Mamba-based 2D backbone. Tab. 5.7

Table 5.7: Comparison of the two baseline models (PV-RCNN [Shi+21] and PartA2 [Shi+20]), each with 4 model configurations. Results reported on the KITTI [Gei+13] **validation set** in terms of 3D AP_{R40}. For the car, pedestrian and cyclist categories, the IoU thresholds are 0.7, 0.5 and 0.5, respectively.

Method	Model Config		Car 3D AP (%) ↑			Pedestrian 3D AP (%) ↑			Cyclist 3D AP (%) ↑		
	Relation	Mamba	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
PV-RCNN	-	-	91.87	84.53	82.41	65.38	57.99	53.17	89.95	70.81	66.37
	-	✓	91.48	84.12	82.28	**66.28**	**59.03**	**53.70**	**93.04**	73.99	69.82
	✓	-	**92.45**	**85.36**	**83.00**	65.93	58.13	51.81	91.52	71.13	66.59
	✓	✓	92.41	85.27	82.93	62.86	56.17	50.95	92.02	**74.68**	**69.90**
PartA2	-	-	**91.84**	82.53	80.17	64.13	58.04	53.08	88.24	71.12	66.76
	-	✓	**91.84**	**84.00**	**82.10**	**67.72**	**60.72**	**54.64**	90.20	**72.02**	**68.39**
	✓	-	91.72	82.59	80.43	54.10	48.37	43.63	**90.78**	70.93	67.60
	✓	✓	**91.84**	82.48	80.35	59.94	51.82	47.39	88.36	71.54	68.22

We observe that for PV-RCNN, both Mamba-based sD backbone and object relation module brings improvements individually for all three classes in general. Except pedestrian class,

the combination of Mamba-based sD backbone and object relation module leads to a better performance for the rest two classes. The improvement for cyclist is pretty significant. For PartA2, object relation module causes decrease in the detection for car and pedestrian classes in general, while the Mamba-based 2D backbone brings pretty good improvement.

Effect of GNN Components

We compare the effects of different components in the GNN in Tab. 5.8. We report results of cars on the 3D AP_{R40} metric. We design four-group experiments, including 1) initializing the node features without box information, 2) combining the box information with the proposal features as the initial node features, and only using the feature differences to calculate the edge features and update the node features, 3) concatenating feature differences and proposal box differences to update the node features, and 4) based on the third experiment, concatenating node features of the same node from different layers to be the final output feature of the GNN module.

Table 5.8: Comparison between using different components of relation graph on the KITTI [Gei+13] **validation set**. We report car class (moderate difficulty) using 3D AP_{R40} with IoU threshold 0.7. PV-RCNN [Shi+21] is utilized as the basic network. "Init. box" represents using box information to initialize node features and "box diff." is the proposal box differences.

Method	init. box	box diff.	feature appended	3D AP (Mod.) (%) ↑
Exp. 1	-	-	-	82.91
Exp. 2	✓	-	-	84.07
Exp. 3	✓	✓	-	84.86
Exp. 4	✓	✓	✓	**85.36**

We notice that applying box information to initialize node features improves more than 1%. The reason is that the original box information provides the spatial state of each proposal, such as position, box size, and rotation angle, which helps the network to effectively capture contextual information and pass messages between the neighbor nodes. We conduct the second experiment with the box differences, which further improves 0.79% accuracy. This result verifies that learning the similar movement patterns of neighbor objects assists the network in overcoming partial occlusion and predicting more precise bounding box rotation. Additionally, we combine the node features from different GNN layers, making the output features encode various latent features and keeping semantic information. The combination further increases the detection accuracy to 85.36%.

Effect of Different Graph Generator

We explore the influence of different graph generation strategies on the detection performance. We compare two methods, including KNN with k equal to 16 and a radius graph with a range of 6 meters. Usually, fewer proposals are included in the range of 0 to 6 meters in the KITTI dataset. (as shown in Fig. 4.3).

In Tab. 5.9, we illustrate the experiment results of all three categories, such as car, pedestrian, and cyclist, on the moderate difficulty level. The model with the graph generated by KNN outperforms the radius method for cars with 1.43%. Conversely, the radius model outperforms the KNN model by achieving 1.97% and 1% higher accuracy rates for detecting pedestrians and cyclists, respectively. Due to the large distribution range of cars, connecting more neighbor nodes can provide general pattern information to improve performance. In contrast, pedestrians and cyclists tend to cluster densely in scenes; hence, incorporating

Table 5.9: Comparison between different graph generation approaches (KNN and Radius) on the KITTI [Gei+13] **validation set**. PV-RCNN [Shi+21] is utilzed as the basic network. We set k to 16 and the radius to 6 meters. The evaluation metric is 3D AP_{R40} moderate difficulty with IoU threshold 0.7 for cars and 0.5 for cyclists and pedestrians.

Method	3D AP (Mod.) (%) ↑		
	Car	Pedestrian	Cyclist
PV-RCNN Relation (Radius)	83.25	**59.28**	**74.29**
PV-RCNN Relation (KNN)	**84.68**	57.31	73.29

information from distant objects can introduce noise. However, leveraging proposal information from a relatively short range, which is more pertinent to these proposals, significantly improve the detection of these objects.

Effect of K Value of KNN-based Graph

Moreover, we explore the impact of k values on the KNN-based graph generator for the performance of 3D object detection. We compare the baseline model with our object relation approach using different k values (16 and 32). We show the experiment results based on the PV-RCNN [Shi+21] network in Tab. 5.10.

Table 5.10: Comparison between the baseline PV-RCNN [Shi+21] and our PV-RCNN relation network with various KNN settings on the KITTI [Gei+13] **validation set** for car class using 3D AP_{R40} w.r.t IoU threshold 0.7. Bold and underlined represent each metric's best and second-best performance.

Method	Car - 3D AP (%) ↑		
	Easy	Moderate	Hard
PV-RCNN (baseline)	91.95	84.60	82.49
PV-RCNN Relation (K=16)	<u>92.04</u>	**85.13**	**82.76**
PV-RCNN Relation (K=32)	**92.22**	<u>84.98</u>	<u>82.55</u>

When k is 16, the PV-RCNN relation model achieves the best performances for the moderate and hard-level cars. If we utilize a larger value (k=32), the object relation module performs the best on easy difficulty (92.22%), while the 3D AP of moderate and hard levels decreases. The reason is that a suitable k value helps the network extend the receptive field, assisting the detector in perceiving the surrounding environment. However, a larger k value leads the relation module to establish a connection with more objects, which can introduce noise into the object features, causing a decrease in the object detection performance.

Effect of Additional Instance Feature

As discussed in section 4.2.3, we attempt to extend node feature in the relation graph with additional instance-specific features by masking raw points within proposal boxes out and feeding them to a PointNet-like encoder. In Tab. 5.11 we report the detection results of 1) the baseline model PV-RCNN, 2) the model with original object relation module PV-RCNN_Rel only counting pooled feature as node feature, and 3) PV-RCNN_Rel + IF (Instance Feature), counting instance-specific feature into node feature.

We observe that for all the object categories and all the difficulty levels, the proposed feature extension leads to a decrement on model performance, although the idea is intuitively reasonable. For pedestrian the 3D AP even decreases by over -5%. We analysed this issue

Table 5.11: Comparison of the baseline model *PV-RCNN* [Shi+21] with *PV-RCNN_Rel*, only taking pooled feature as node feature, and *PV-RCNN_Rel + IF*, counting extracted instance-specific feature into node feature. Results reported on the KITTI [Gei+13] **validation set** in terms of 3D AP_{R40}. For the car, pedestrian and cyclist categories, the IoU thresholds are 0.7, 0.5 and 0.5, respectively. Concatenating instance-specific feature to extend node feature leads to decrement on model performance in general.

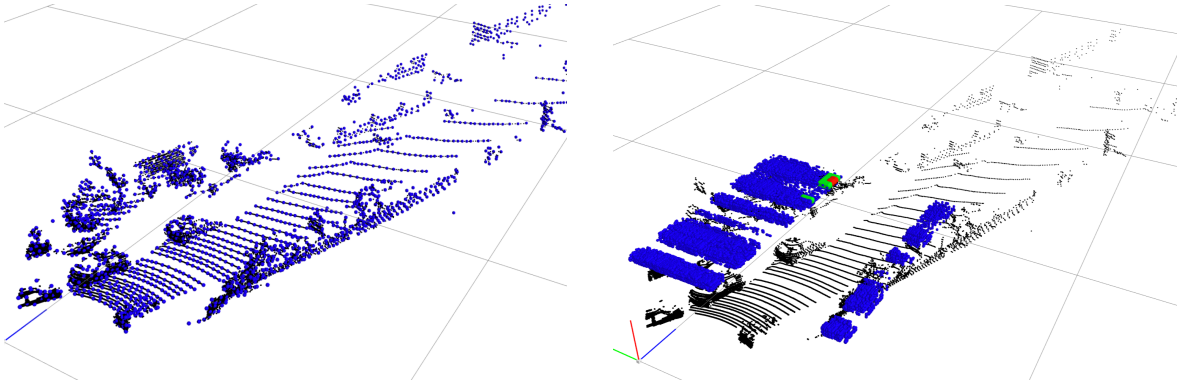
Method	Car 3D AP (%) ↑			Pedestrian 3D AP (%) ↑			Cyclist 3D AP (%) ↑		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
PV-RCNN	91.87	84.53	82.41	65.38	57.99	**53.17**	89.95	70.81	66.37
PV-RCNN_Rel	**92.45**	**85.36**	**83.00**	**65.93**	**58.13**	51.81	**91.52**	**71.13**	**66.59**
PV-RCNN_Rel+IF	92.28	83.50	82.74	60.41	52.70	48.77	89.88	70.96	66.58

by visualizing the distribution of proposal boxes before NMS (Fig. A.5 in appendix) and note that most of the proposal boxes are generated in non-object area. Features encoded from points within these non-object proposals actually introduce noise to node feature.

Pre-process Point Clouds with Octree

Inspired by PointMamba [Liu+24a], we also attempt to use octree to pre-process point cloud to construct the causal dependency relationship on original disordered point cloud data. So far, PointMamba has only been applied to process indoor scenes, where the point clouds are pretty dense. Applying PointMamba to do 3D object detection in autonomous driving is our first attempt to process outdoor point cloud scene, which has larger range scale and higher sparsity.

We first visualized the octants, i.e. the nodes of octree of some common indoor objects to get a intuitive understanding (Fig, A.3, see in appendix). Then we compare the outcomes of octree and the Farthest Point Sampling (FPS) by visualizing several downsampled point cloud scenes in KITTI dataset in Fig. 5.7. In this example we observe that when the street is oblique to ego car's longitudinal axis, octree could fail to reach distant points, causing an incomplete downsampled representation for the entire scene. Fig. A.4 in appendix displays more examples that octree fails to downsample outdoor scene completely. In consideration of this issue, we finally give up to use octree as downsampling method for outdoor point cloud scenes.



(a) Downsampled with FPS **(b)** Downsampled with octree

Figure 5.7: Comparison downsampling outcomes of FPS method and octree method for the same KITTI point cloud scene. For FPS method, the 2048 downsampled points are displayed in blue spheres. For octree method, octants in level 2, level 4, and level 8 are displayed in red, green, and blue spheres respectively.

Chapter 6

Conclusion and Future Works

6.1 Conclusion

In this work, we present an inter-object relation module to improve the detection performance and a Mamba-based light-weighted feature encoder to save computational cost for arbitrary two-stage 3D object detectors. The Mamba-based 2D backbone replaces the traditional 2D convolution stack with Visual State Space block as the feature encoder. The object relation module leverages a graph neural network to extract features from inter-object relationship graphs, refining proposal features to improve detection accuracy in occlusion and distant cases. We exploit specific patterns, such as parallel parking on narrow streets, to refine the predicted object direction.

The Mamba-based 2D backbone decreases the amount of parameters by 20% for PV-RCNN and 4% for PartA2, and reduces floating point operations (FLOPs) by around 40% for both baseline models, saving the computational cost effectively. Our evaluation on the KITTI dataset demonstrates improvements of 0.52% and 0.18% in detecting cars at the hard difficulty level over the PV-RCNN and PartA2 baseline models, respectively. The evaluation of PV-RCNN_Mamba on the Waymo dataset demonstrates improvements around 1.2% in vehicle class and 2.4% $\sim$ 2.8% in pedestrian class over PV-RCNN baseline model. We further conduct comprehensive ablation studies to verify the effectiveness brought by Mamba-based 2D backbone and object relation module separately, and the effectiveness of core components of our proposed inter-object relation approach.

6.2 Future Works

Substituting convolution layers of feature encoders in 2D backbone with Mamba-based VSS-block is a successful attempt at saving computational cost for 3D object detection. The effectiveness is remarkable, although the modification is very slight in respect with the entire network. We can further explore to apply Mamba-based backbone to more feature encoder in traditional 3D object detection framework. Inspired by Point Mamba [Liu+24a], we can try to use octree-based sampling method to process instance-level raw point cloud and apply Mamba-based approach to encode 3D features. Inspired by Mamba-in-Mamba [Che+24], we can try to encode local features and global features of a scene in different levels hierarchically and fuse them in the end.

In the future we can implement our proposed Mamba-based 2D backbone and object-relation module to more two-stage 3D object detection networks. Since Mamba already got great success in image-related tasks, we can conduct more exploration applying Mamba-based methods in multi-modal detection.

Appendix A

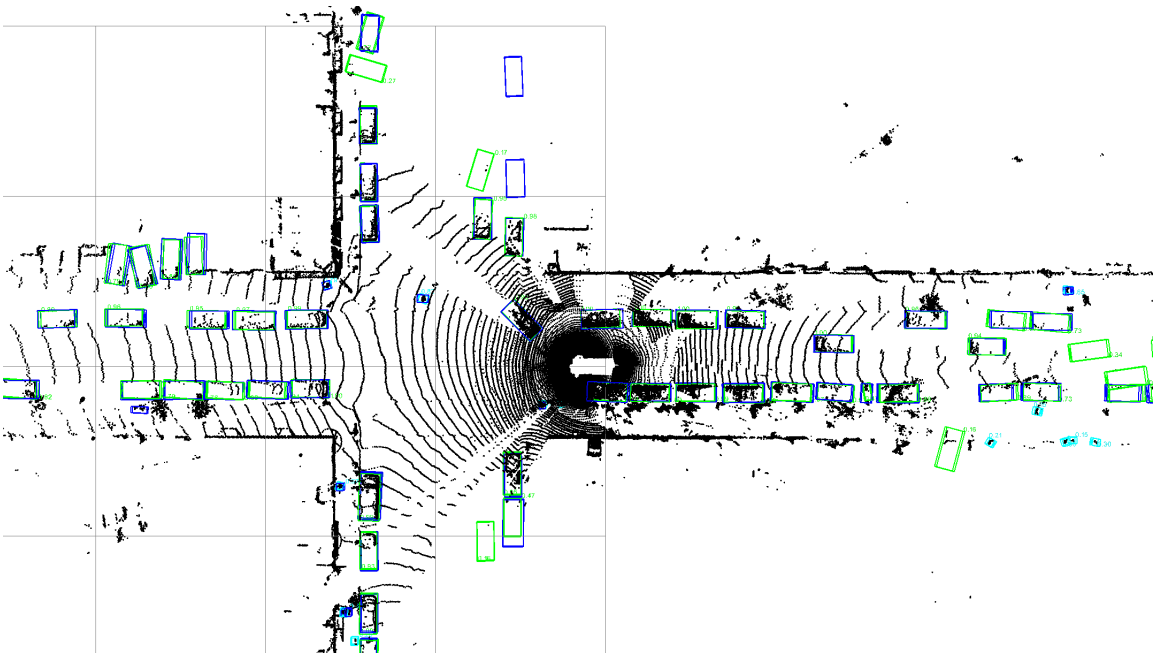
Appendix

Table A.1: Comparison of the baseline models (PV-RCNN [Shi+21] and PartA2 [Shi+20]) with our proposed object relation module and Mamba-based 2D backbone. Results reported on the KITTI [Gei+13] **validation set** in terms of BEV AP_{R40}. For the car, pedestrian and cyclist categories, the IoU thresholds are 0.7, 0.5 and 0.5, respectively.

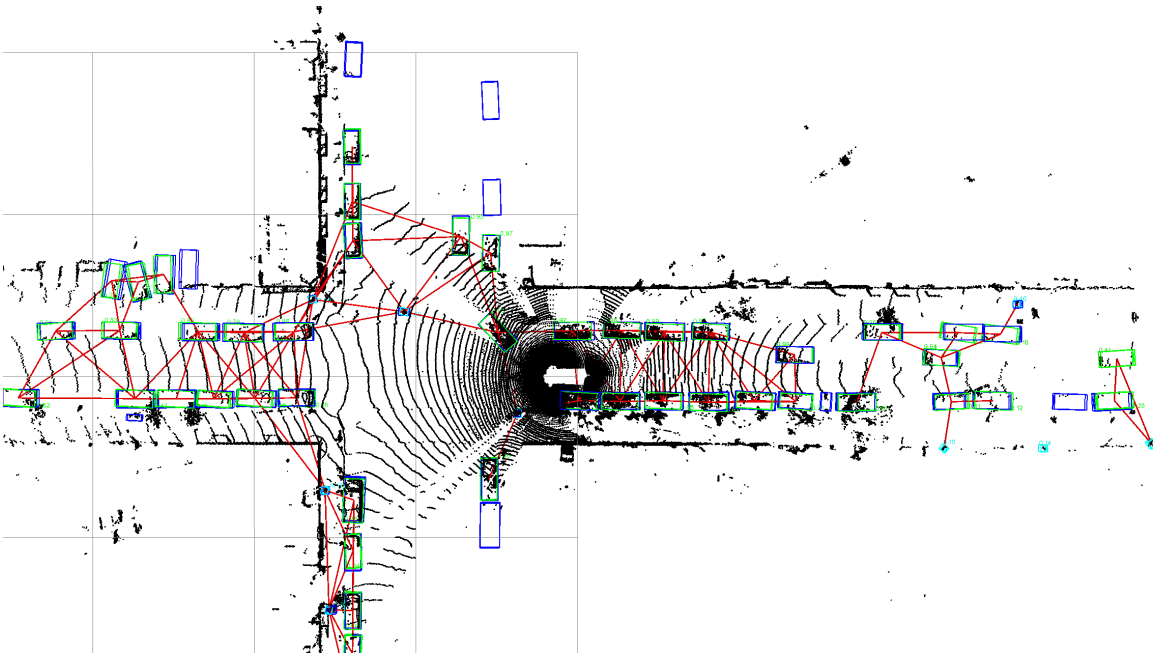
Method	Car BEV AP 0.7 (%) ↑			Pedestrian BEV AP 0.5 (%) ↑			Cyclist BEV AP 0.5 (%) ↑		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
PV-RCNN	94.58	91.00	88.51	**68.47**	**61.37**	**56.51**	91.26	73.94	69.53
PV-RCNN_Rel_Mamba	**95.56**	**91.55**	**89.23**	66.49	58.77	53.64	**94.94**	**76.91**	**71.96**
Improvements	+0.98	+0.55	+0.72	-1.98	-2.60	-2.87	+3.68	+2.97	+2.43
PartA2	94.05	88.33	88.04	**66.73**	**61.47**	**56.71**	89.14	72.87	69.59
PartA2_Rel_Mamba	**95.20**	**90.76**	**88.83**	62.66	54.65	50.42	**93.84**	**76.39**	**73.19**
Improvements	+1.15	+2.43	+0.79	-4.07	-6.82	-6.29	+4.70	+3.52	+3.60

Table A.2: Comparison of the baseline model PV-RCNN_Rel [Shi+21] and the PV-RCNN_Rel_Mamba with our proposed substitution of Mamba-based 2D backbone. Trained with a 20% split of Waymo [Mei+22] training set, results reported on the full Waymo **validation set** in terms of L1_mAP, L1_mAPH, L2_mAP, and L2_mAPH.

Methods	Vehicle (%) ↑			
	Level 1 mAP	Level 1 mAPH	Level 2 mAP	Level 2 mAPH
PV-RCNN_Rel	70.28	69.54	61.60	60.94
PV-RCNN_Rel_Mamba	**74.86**	**74.28**	**66.15**	**65.63**
Improvements	+4.58	+4.74	+4.55	+4.69
	Pedestrian (%) ↑			
	Level 1 mAP	Level 1 mAPH	Level 2 mAP	Level 2 mAPH
PV-RCNN_Rel	61.87	31.53	53.14	27.09
PV-RCNN_Rel_Mamba	**70.92**	**34.41**	**61.98**	**30.10**
Improvements	+9.05	+2.88	+8.84	+3.01
	Cyclist (%) ↑			
	Level 1 mAP	Level 1 mAPH	Level 2 mAP	Level 2 mAPH
PV-RCNN_Rel	33.66	14.8	32.37	14.24
PV-RCNN_Rel_Mamba	**44.34**	**24.88**	**42.64**	**23.92**
Improvements	+10.68	+10.08	+10.27	+9.68

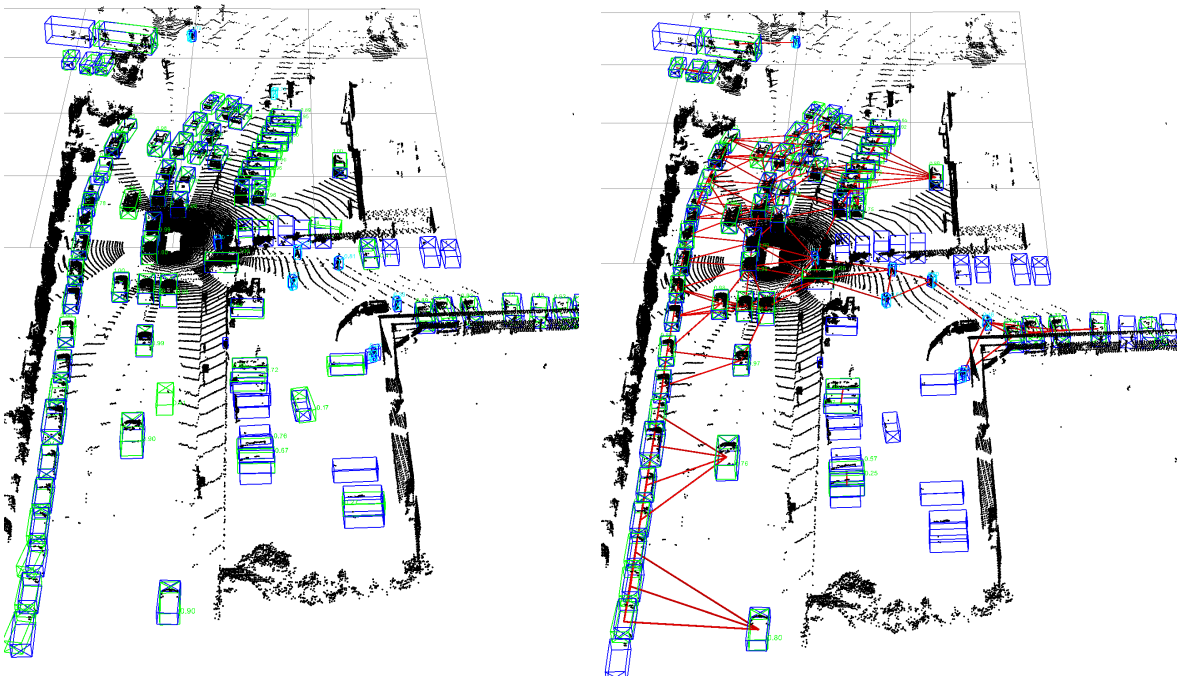


(a) PV-RCNN



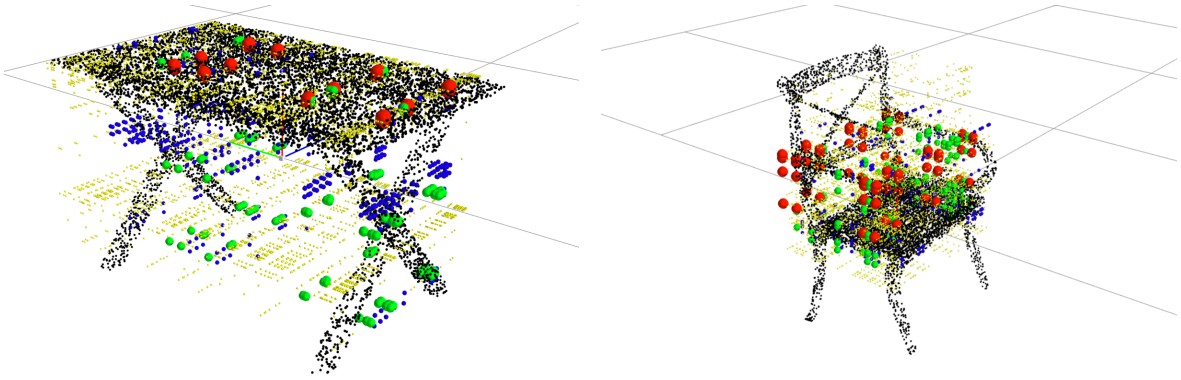
(b) PV-RCNN_Rel

Figure A.1: Bird-Eye-View of qualitative results on Waymo dataset. Bounding boxes of **ground truths**, the detection results of are shown in blue, and green respectively. The object relation edges are shown in red, which are displayed for visualization purpose but not counted as detection results.



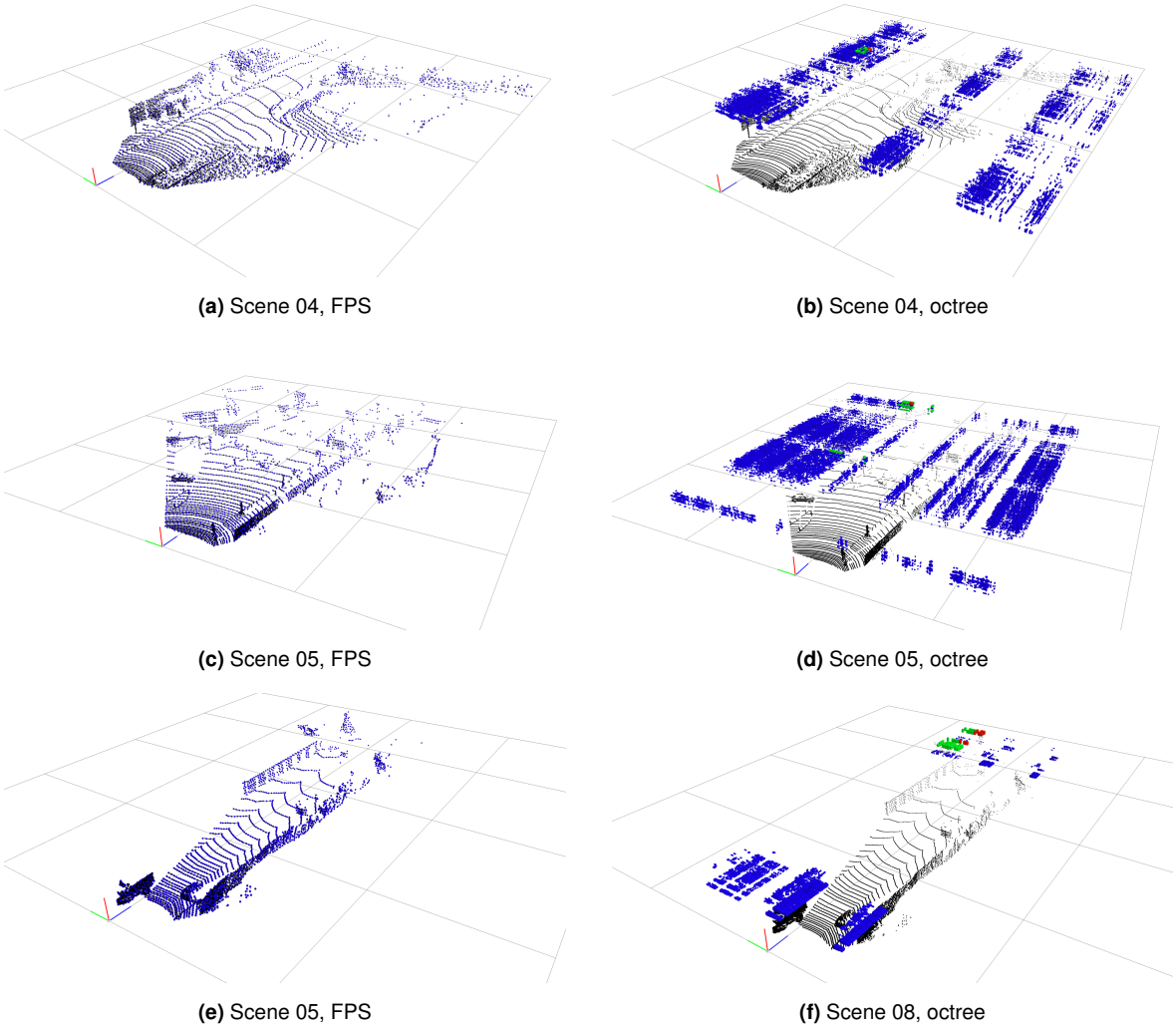
(a) PV-RCNN **(b)** PV-RCNN_Rel

Figure A.2: Side view of qualitative results on Waymo dataset. Bounding boxes of **ground truths**, the detection results of are shown in blue, and green respectively. The object relation edges are shown in red, which are displayed for visualization purpose but not counted as detection results.



(a) Desk **(b)** Chair

Figure A.3: Octants of indoor objects Visualized octants in level 2, level 3, level 4 and level 6 in red, green, blue and yellow spheres respectively.



(a) Scene 04, FPS

(b) Scene 04, octree

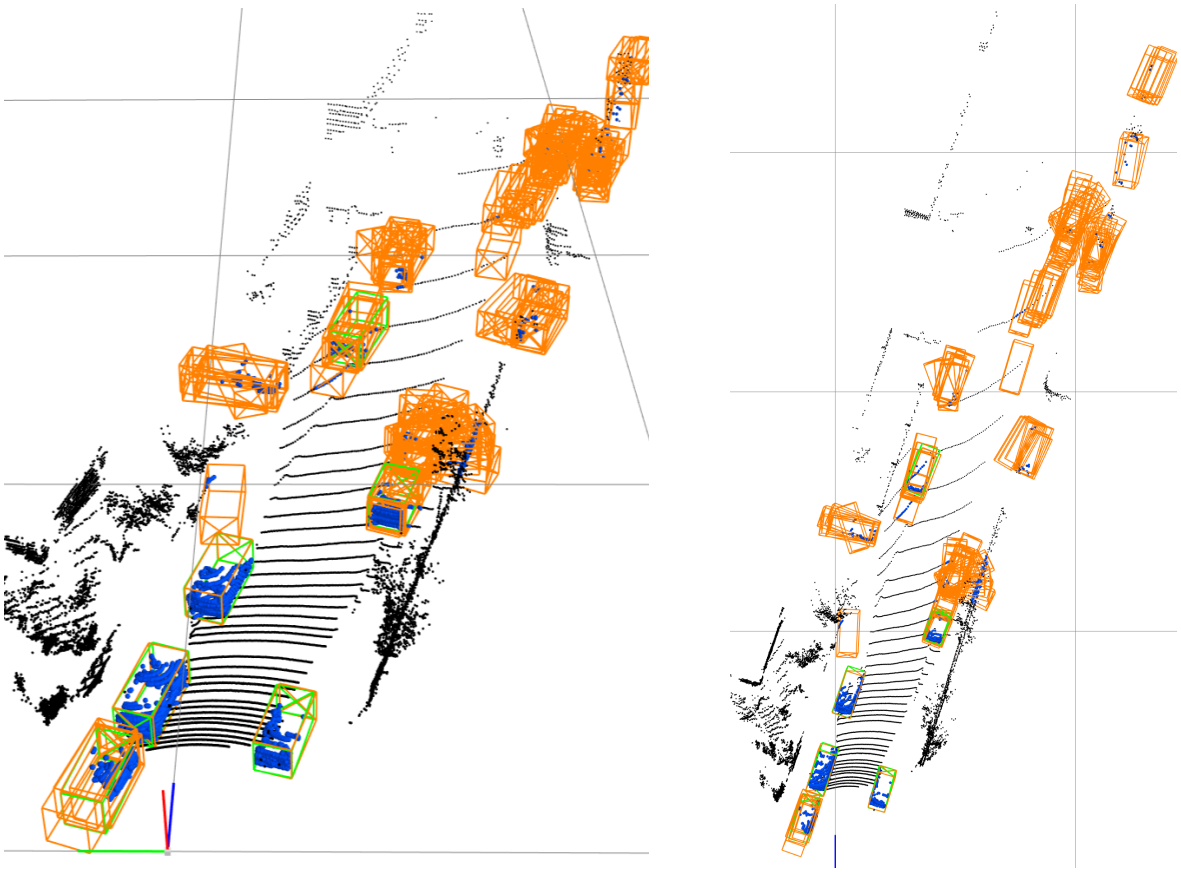
(c) Scene 05, FPS

(d) Scene 05, octree

(e) Scene 05, FPS

(f) Scene 08, octree

Figure A.4: Comparison downsampling outcomes of FPS and octree for KITTI point cloud scene. For FPS method, the 2048 downsampled points are displayed in blue spheres. For octree method, octants in level 2, level 4, and level 8 are displayed in red, green, and blue spheres respectively.



(a) Side view **(b)** Bird-eye-view

Figure A.5: Visualization of proposal boxes and points for instance-specific feature extraction. Each scene contains 100 proposals (orange) before NMS. Note that most of the proposal boxes are generated in non-object area (ground truth boxes displayed in green). Features encoded from points (blue spheres) within these non-object proposals introduce noise to node feature.

Listing A.1: Mamba-based 2D Backbone (css-format)

```
(backbone_2d): BaseBEVBackboneMamba(
    (blocks): ModuleList(
      (0): Sequential(
        (0): ZeroPad2d((1, 1, 1, 1))
        (1): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1),
           bias=False)
        (2): BatchNorm2d(128, eps=0.001, momentum=0.01,
           affine=True, track_running_stats=True)
        (3): ReLU()
        (4): Permute()
        (5): VSSBlock(hidden_dim=128)  # Stage I VSS-block
        (6): Permute()
      )
      (1): Sequential(
        (0): ZeroPad2d((1, 1, 1, 1))
        (1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2),
           bias=False)
        (2): BatchNorm2d(256, eps=0.001, momentum=0.01,
           affine=True, track_running_stats=True)
        (3): ReLU()
        (4): Permute()
        (5): VSSBlock(hidden_dim=256)  # Stage II VSS-block
        (6): Permute()
      )
    )
    (deblocks): ModuleList(
      (0): Sequential(
        (0): ConvTranspose2d(128, 256, kernel_size=(1, 1),
           stride=(1, 1), bias=False)
        (1): BatchNorm2d(256, eps=0.001, momentum=0.01,
           affine=True, track_running_stats=True)
        (2): ReLU()
      )
      (1): Sequential(
        (0): ConvTranspose2d(256, 256, kernel_size=(2, 2),
           stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=0.001, momentum=0.01,
           affine=True, track_running_stats=True)
        (2): ReLU()
      )
    )
  )
```

Listing A.2: Stage I VSS-block (css-format)

```
VSSBlock(
  (norm): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
  (op): SS2D(
    (out_norm): LayerNorm((256,), eps=1e-05,
       elementwise_affine=True)
    (in_proj): Linear(in_features=128, out_features=512,
       bias=False)
    (act): SiLU()
    (conv2d): Conv2d(256, 256, kernel_size=(3, 3), stride=(1,
       1), padding=(1, 1), groups=256)
    (out_act): Identity()
    (out_proj): Linear(in_features=256, out_features=128,
       bias=False)
    (dropout): Identity()
  )
  (drop_path): timm.DropPath(0)
  (norm2): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
  (mlp): Mlp(
    (fc1): Linear(in_features=128, out_features=512, bias=True)
    (act): GELU(approximate='none')
    (fc2): Linear(in_features=512, out_features=128, bias=True)
    (drop): Dropout(p=0.0, inplace=False)
  )
)
```

Listing A.3: Stage II VSS-block (css-format)

```
VSSBlock(
  (norm): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
  (op): SS2D(
    (out_norm): LayerNorm((512,), eps=1e-05,
       elementwise_affine=True)
    (in_proj): Linear(in_features=256, out_features=1024,
       bias=False)
    (act): SiLU()
    (conv2d): Conv2d(512, 512, kernel_size=(3, 3), stride=(1,
       1), padding=(1, 1), groups=512)
    (out_act): Identity()
    (out_proj): Linear(in_features=512, out_features=256,
       bias=False)
    (dropout): Identity()
  )
  (drop_path): timm.DropPath(0)
  (norm2): LayerNorm((256,), eps=1e-05, elementwise_affine=True)
  (mlp): Mlp(
    (fc1): Linear(in_features=256, out_features=1024, bias=True)
    (act): GELU(approximate='none')
    (fc2): Linear(in_features=1024, out_features=256, bias=True)
    (drop): Dropout(p=0.0, inplace=False)
  )
)
```

Bibliography

[BWL20] Bochkovskiy, A., Wang, C.-Y., and Liao, H.-Y. M. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. 2020. arXiv: 2004.10934 [cs.CV].

[Cao+21] Cao, H., Wang, Y., Chen, J., Jiang, D., Zhang, X., Tian, Q., and Wang, M. *Swin-Unet: Unet-like Pure Transformer for Medical Image Segmentation*. 2021. arXiv: 2105.05537 [eess.IV].

[Che+24] Chen, T., Tan, Z., Gong, T., Chu, Q., Wu, Y., Liu, B., Ye, J., and Yu, N. *MiM-ISTD: Mamba-in-Mamba for Efficient Infrared Small Target Detection*. 2024. arXiv: 2403.02148 [cs.CV].

[Che+19] Chen, Y., Liu, S., Shen, X., and Jia, J. *Fast Point R-CNN*. 2019. arXiv: 1908.02990 [cs.CV].

[CS23] Chib, P. S. and Singh, P. *Recent Advancements in End-to-End Autonomous Driving using Deep Learning: A Survey*. 2023. arXiv: 2307.04370 [cs.RO].

[CZ20] Cui, Z. and Zhang, Z. "PVF-NET: Point-Voxel Fusion 3D Object Detection Framework for Point Cloud". In: *2020 17th Conference on Computer and Robot Vision (CRV)*. 2020, pp. 125–133. DOI: 10.1109/CRV50864.2020.00025.

[Cyb88] Cybenko, G. *Approximation by Superpositions of a Sigmoidal Function*. Tech. rep. Medford, MA: Department of computer Science, Tufts University, Oct. 1988.

[Den+21] Deng, J., Shi, S., Li, P., Zhou, W., Zhang, Y., and Li, H. *Voxel R-CNN: Towards High Performance Voxel-based 3D Object Detection*. 2021. arXiv: 2012.15712 [cs.CV].

[DMH23] Desislavov, R., Martínez-Plumed, F., and Hernández-Orallo, J. "Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning". In: *Sustainable Computing: Informatics and Systems* 38 (Apr. 2023), p. 100857. ISSN: 2210-5379. DOI: 10.1016/j.suscom.2023.100857. URL: http://dx.doi.org/10.1016/j.suscom.2023.100857.

[Dos+21] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV].

[Erç+22] Erçelik, E., Yurtsever, E., Liu, M., Yang, Z., Zhang, H., Topçam, P., Listl, M., Caylı, Y. K., and Knoll, A. "3d object detection with a self-supervised lidar scene flow backbone". In: *European Conference on Computer Vision*. Springer. 2022, pp. 247–265.

[Gei+13] Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. "Vision meets robotics: The kitti dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.

[Gir15] Girshick, R. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV].

[Gir+14] Girshick, R., Donahue, J., Darrell, T., and Malik, J. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV].

[20] "GRNet: Geometric relation network for 3D object detection from point clouds". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 165 (2020), pp. 43–53. ISSN: 0924-2716. DOI: https://doi.org/10.1016/j.isprsjprs.2020.05.008.

[GD23] Gu, A. and Dao, T. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. 2023. arXiv: 2312.00752 [cs.LG].

[GGR22] Gu, A., Goel, K., and Ré, C. *Efficiently Modeling Long Sequences with Structured State Spaces*. 2022. arXiv: 2111.00396 [cs.LG].

[He+20] He, C., Zeng, H., Huang, J., Hua, X.-S., and Zhang, L. "Structure aware single-stage 3d object detection from point cloud". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11873–11882.

[HKW22] Hu, J. S. K., Kuai, T., and Waslander, S. L. *Point Density-Aware Voxels for LiDAR 3D Object Detection*. 2022. arXiv: 2203.05662 [cs.CV].

[IKK21] Ibing, M., Kobsik, G., and Kobbelt, L. *Octree Transformer: Autoregressive 3D Shape Generation on Hierarchically Structured Sequences*. 2021. arXiv: 2111.12480 [cs.CV].

[Kir+19] Kirillov, A., Girshick, R., He, K., and Dollár, P. *Panoptic Feature Pyramid Networks*. 2019. arXiv: 1901.02446 [cs.CV].

[KSH12] Krizhevsky, A., Sutskever, I., and Hinton, G. E. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[Lan+22] Lan, Y., Duan, Y., Liu, C., Zhu, C., Xiong, Y., Huang, H., and Xu, K. *ARM3D: Attention-based relation module for indoor 3D object detection*. 2022. arXiv: 2202.09715 [cs.CV].

[Lan+21] Lan, Y., Duan, Y., Shi, Y., Huang, H., and Xu, K. "3DRM: Pair-wise relation module for 3D object detection". In: *Computers amp; Graphics* 98 (Aug. 2021), pp. 58–70. ISSN: 0097-8493. DOI: 10.1016/j.cag.2021.04.033. URL: http://dx.doi.org/10.1016/j.cag.2021.04.033.

[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*. 2019. arXiv: 1812.05784 [cs.LG].

[Lec+98] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[Lev+11] Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., Kolter, J. Z., Langer, D., Pink, O., Pratt, V., Sokolsky, M., Stanek, G., Stavens, D., Teichman, A., Werling, M., and Thrun, S. "Towards fully autonomous driving: Systems and algorithms". In: *2011 IEEE Intelligent Vehicles Symposium (IV)*. 2011, pp. 163–168. DOI: 10.1109/IVS.2011.5940562.

[Li+21] Li, J., Dai, H., Shao, L., and Ding, Y. *From Voxel to Point: IoU-guided 3D Object Detection for Point Cloud with Voxel-to-Point Decoder*. 2021. arXiv: 2108.03648 [cs.CV].

[LLY23] Li, J., Luo, C., and Yang, X. *PillarNeXt: Rethinking Network Designs for 3D Object Detection in LiDAR Point Clouds*. 2023. arXiv: 2305.04925 [cs.CV].

[Li+20] Li, Z., Yang, W., Peng, S., and Liu, F. *A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects*. 2020. arXiv: 2004.02806 [cs.CV].

[LWW21] Li, Z., Wang, F., and Wang, N. *LiDAR R-CNN: An Efficient and Universal 3D Object Detector*. 2021. arXiv: 2103.15297 [cs.CV].

[Lin+18] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

[Liu+24a] Liu, J., Yu, R., Wang, Y., Zheng, Y., Deng, T., Ye, W., and Wang, H. *Point Mamba: A Novel Point Cloud Backbone Based on State Space Model with Octree-Based Ordering Strategy*. 2024. arXiv: 2403.06467 [cs.CV].

[Liu+24b] Liu, M., Yurtsever, E., Fossaert, J., Zhou, X., Zimmer, W., Cui, Y., Zagar, B. L., and Knoll, A. C. *A Survey on Autonomous Driving Datasets: Statistics, Annotation Quality, and a Future Outlook*. 2024. arXiv: 2401.01454 [cs.CV].

[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: *Lecture Notes in Computer Science*. Springer International Publishing, 2016, pp. 21–37. ISBN: 9783319464480. DOI: 10.1007/978-3-319-46448-0_2. URL: http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[Liu+24c] Liu, Y., Tian, Y., Zhao, Y., Yu, H., Xie, L., Wang, Y., Ye, Q., and Liu, Y. *VMamba: Visual State Space Model*. 2024. arXiv: 2401.10166 [cs.CV].

[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*. 2021. arXiv: 2103.14030 [cs.CV].

[Liu+19] Liu, Z., Tang, H., Lin, Y., and Han, S. *Point-Voxel CNN for Efficient 3D Deep Learning*. 2019. arXiv: 1907.03739 [cs.CV].

[Mao+23] Mao, J., Shi, S., Wang, X., and Li, H. *3D Object Detection for Autonomous Driving: A Comprehensive Survey*. 2023. arXiv: 2206.09474 [cs.CV].

[MWL19] Mao, J., Wang, X., and Li, H. *Interpolated Convolutional Networks for 3D Point Cloud Understanding*. 2019. arXiv: 1908.04512 [cs.CV].

[Mei+22] Mei, J., Zhu, A. Z., Yan, X., Yan, H., Qiao, S., Zhu, Y., Chen, L.-C., Kretzschmar, H., and Anguelov, D. *Waymo Open Dataset: Panoramic Video Panoptic Segmentation*. 2022. arXiv: 2206.07704 [cs.CV].

[Mia+21] Miao, Z., Chen, J., Pan, H., Zhang, R., Liu, K., Hao, P., Zhu, J., Wang, Y., and Zhan, X. "PVGNet: A Bottom-Up One-Stage 3D Object Detector with Integrated Multi-Level Features". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 3278–3287. DOI: 10.1109/CVPR46437.2021.00329.

[Ngu+22] Nguyen, E., Goel, K., Gu, A., Downs, G. W., Shah, P., Dao, T., Baccus, S. A., and Ré, C. *S4ND: Modeling Images and Videos as Multidimensional Signals Using State Spaces*. 2022. arXiv: 2210.06583 [cs.CV].

[NLH21] Noh, J., Lee, S., and Ham, B. *HVPR: Hybrid Voxel-Point Representation for Single-stage 3D Object Detection*. 2021. arXiv: 2104.00902 [cs.CV].

[NVI15] NVIDIA. *Achieved FLOPs*. https://docs.nvidia.com/gameworks/content/developertools/desktop/analysis/report/cudaexperiments/kernellevel/achievedflops.htm. 2015.

[PA24] Patro, B. N. and Agneeswaran, V. S. *Mamba-360: Survey of State Space Models as Transformer Alternative for Long Sequence Modelling: Methods, Applications, and Challenges*. 2024. arXiv: 2404.16112 [cs.LG].

[Pow20] Powers, D. M. W. *Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation*. 2020. arXiv: 2010.16061 [cs.LG].

[Qi+17a] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. 2017. arXiv: 1612.00593 [cs.CV].

[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*. 2017. arXiv: 1706.02413 [cs.CV].

[Red+16] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[RF16] Redmon, J. and Farhadi, A. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV].

[RF18] Redmon, J. and Farhadi, A. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV].

[Ren+16] Ren, S., He, K., Girshick, R., and Sun, J. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].

[Sch+23] Schinagl, D., Krispel, G., Fruhwirth-Reisinger, C., Possegger, H., and Bischof, H. *GACE: Geometry Aware Confidence Enhancement for Black-Box 3D Object Detectors on LiDAR-Data*. 2023. arXiv: 2310.20319 [cs.CV].

[Sha20] Shazeer, N. *GLU Variants Improve Transformer*. 2020. arXiv: 2002.05202 [cs.LG].

[Shi+21] Shi, S., Guo, C., Jiang, L., Wang, Z., Shi, J., Wang, X., and Li, H. *PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection*. 2021. arXiv: 1912.13192 [cs.CV].

[Shi+22] Shi, S., Jiang, L., Deng, J., Wang, Z., Guo, C., Shi, J., Wang, X., and Li, H. *PV-RCNN++: Point-Voxel Feature Set Abstraction With Local Vector Representation for 3D Object Detection*. 2022. arXiv: 2102.00463 [cs.CV].

[SWL19] Shi, S., Wang, X., and Li, H. *PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud*. 2019. arXiv: 1812.04244 [cs.CV].

[Shi+20] Shi, S., Wang, Z., Shi, J., Wang, X., and Li, H. *From Points to Parts: 3D Object Detection from Point Cloud with Part-aware and Part-aggregation Network*. 2020. arXiv: 1907.03670 [cs.CV].

[Sun+20] Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., Vasudevan, V., Han, W., Ngiam, J., Zhao, H., Timofeev, A., Ettinger, S., Krivokon, M., Gao, A., Joshi, A., Zhao, S., Cheng, S., Zhang, Y., Shlens, J., Chen, Z., and Anguelov, D. *Scalability in Perception for Autonomous Driving: Waymo Open Dataset*. 2020. arXiv: 1912.04838 [cs.CV].

[Tan+20] Tang, H., Liu, Z., Zhao, S., Lin, Y., Lin, J., Wang, H., and Han, S. *Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution*. 2020. arXiv: 2007.16100 [cs.CV].

[Uij+13] Uijlings, J. R., Van De Sande, K. E., Gevers, T., and Smeulders, A. W. "Selective search for object recognition". In: *International journal of computer vision* 104 (2013), pp. 154–171.

[Vas+23] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[Wan+24a] Wang, A., Chen, H., Liu, L., Chen, K., Lin, Z., Han, J., and Ding, G. *YOLOv10: Real-Time End-to-End Object Detection*. 2024. arXiv: 2405.14458 [cs.CV].

[WBL22] Wang, C.-Y., Bochkovskiy, A., and Liao, H.-Y. M. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. arXiv: 2207.02696 [cs.CV].

[Wan+20] Wang, J., Lan, S., Gao, M., and Davis, L. S. *InfoFocus: 3D Object Detection for Autonomous Driving with Dynamic Information Modeling*. 2020. arXiv: 2007.08556 [cs.CV].

[Wan23] Wang, P.-S. "OctFormer: Octree-based Transformers for 3D Point Clouds". In: *ACM Transactions on Graphics* 42.4 (July 2023), pp. 1–11. ISSN: 1557-7368. DOI: 10.1145/3592131. URL: http://dx.doi.org/10.1145/3592131.

[Wan+17] Wang, P.-S., Liu, Y., Guo, Y.-X., Sun, C.-Y., and Tong, X. "O-CNN: octree-based convolutional neural networks for 3D shape analysis". In: *ACM Transactions on Graphics* 36.4 (July 2017), pp. 1–11. ISSN: 1557-7368. DOI: 10.1145/3072959.3073608. URL: http://dx.doi.org/10.1145/3072959.3073608.

[Wan+24b] Wang, X., Wang, S., Ding, Y., Li, Y., Wu, W., Rong, Y., Kong, W., Huang, J., Li, S., Yang, H., Wang, Z., Jiang, B., Li, C., Wang, Y., Tian, Y., and Tang, J. *State Space Model for New-Generation Network Alternative to Transformers: A Survey*. 2024. arXiv: 2404.09516 [cs.LG].

[WS21] Wang, Y. and Solomon, J. *Object DGCNN: 3D Object Detection using Dynamic Graphs*. 2021. arXiv: 2110.06923 [cs.CV].

[Wan+19] Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. *Dynamic Graph CNN for Learning on Point Clouds*. 2019. arXiv: 1801.07829 [cs.CV].

[Wu+22] Wu, Y.-H., Zhang, D., Zhang, L., Zhan, X., Dai, D., Liu, Y., and Cheng, M.-M. *Ret3D: Rethinking Object Relations for Efficient 3D Object Detection in Driving Scenes*. 2022. arXiv: 2208.08621 [cs.CV].

[Wu+20] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. "A comprehensive survey on graph neural networks". In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[YML18] Yan, Y., Mao, Y., and Li, B. "SECOND: Sparsely Embedded Convolutional Detection". In: *Sensors* 18.10 (2018). ISSN: 1424-8220. DOI: 10.3390/s18103337. URL: https://www.mdpi.com/1424-8220/18/10/3337.

[YZK21] Yin, T., Zhou, X., and Krähenbühl, P. *Center-based 3D Object Detection and Tracking*. 2021. arXiv: 2006.11275 [cs.CV].

[Yur+20] Yurtsever, E., Lambert, J., Carballo, A., and Takeda, K. "A Survey of Autonomous Driving: Common Practices and Emerging Technologies". In: *IEEE Access* 8 (2020), pp. 58443–58469. ISSN: 2169-3536. DOI: 10.1109/access.2020.2983149. URL: http://dx.doi.org/10.1109/ACCESS.2020.2983149.

[Zhe+21] Zheng, W., Tang, W., Chen, S., Jiang, L., and Fu, C.-W. "Cia-ssd: Confident iou-aware single-stage object detector from point cloud". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 4. 2021, pp. 3555–3562.

[Zho+23] Zhou, X., Liu, M., Zagar, B. L., Yurtsever, E., and Knoll, A. C. “Vision language models in autonomous driving and intelligent transportation systems”. In: *arXiv preprint arXiv:2310.14414* (2023).

[ZT17] Zhou, Y. and Tuzel, O. *VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection*. 2017. arXiv: 1711.06396 [cs.CV].

[Zou+23] Zou, Z., Chen, K., Shi, Z., Guo, Y., and Ye, J. *Object Detection in 20 Years: A Survey*. 2023. arXiv: 1905.05055 [cs.CV].