Karlsruhe Institute of Technology

3D Voxel Reconstruction and World Model for Autonomous Driving

Master Thesis

Yitian Yang

Department of Computer Science
Institute for Anthropomatics
and
FZI Research Center for Information Technology

Reviewer:	Prof. Dr. E Sax
Second reviewer:	Prof. Dr. J. Becker, Prof. Dr. J. M. Zöllner
Advisor:	Daniel Bogdoll, M.Sc.

Research Period: 30. May 2023 – 31. December 2023

3D Voxel Reconstruction and World Model for Autonomous Driving

by
Yitian Yang

Master Thesis
im December 2023

Master Thesis, FZI
Department of Computer Science, 2023
Reviewers: Prof. Dr. E Sax, Prof. Dr. J. Becker, Prof. Dr. J. M. Zöllner

Abteilung Technisch Kognitive Assistenzsysteme
FZI Research Center for Information Technology

Affirmation

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe,
im December 2023

Yitian Yang

Abstract

The real world is complex, variable, and highly interactive. Therefore, for autonomous driving, understanding the scene and predicting plausible futures is crucial. World models have the capability to learn the environment through observations and make predictions based on actions. Also, the choice of sensors is very important in order to get comprehensive information about the environment. However, existing world models mostly use low resolution images as input and operate only in simple environments.

To address this challenge, we propose an innovative world model that combines point cloud data from the LiDAR and RGB image data from the camera sensor as observation inputs. Additionally, the model not only decodes observations from latent states but also reconstructs 3D voxel occupancy as outputs to enhance the understanding of the environment's geometric features. Unlike most existing world models, our model not only contains 2D visual information but also incorporates 3D spatial data and represents the environment more accurately and comprehensively through voxels.

In our study, we implemented various methods to extract features from point clouds and images, analyzing their effects. We used a transformer-based architecture to fuse features from different sensors and compared it to two other simple methods: one is directly averaging features from various sensors, while the other is concatenating features followed by a fully connected layer. Moreover, we adapted the model to 2D latent states. We collected training and evaluation data in the CARLA simulation environment. Then, we evaluated the reconstruction and prediction performance of different combinations of feature extraction and feature fusion methods, as well as 1D and 2D latent states. The study demonstrates the good performance of our novel world model framework in reconstruction and prediction.

Contents

1 Introduction

In recent years, autonomous driving has become more and more popular and there has been a lot of research on it. For us humans, From early childhood, we are constantly observing and interacting with the world, developing an understanding of the environment through continuous learning. This innate ability allows us to understand scenarios, predict potential changes in them and react quickly. This ability to learn and adapt is critical in the field of autonomous driving, where vehicles must understand their surroundings and respond effectively and safely to unforeseen events.

Therefore, autonomous driving demands a deep comprehension of the vehicle's environment, as well as the ability to make predictions about possible futures. So-called world models [38, 40, 39, 41] have the capability to learn from observations, reconstruct the current state, and make predictions about the future. Traditional world models [38, 40, 39, 41] typically extract features from observations to obtain the current latent state and subsequently use the current latent state to predict the next latent state determined by actions. This newly predicted state can then be used for planning, or decoding to reconstruct the observations possible in the future. The key to the world model is its two functions - reconstruction and prediction. These enable the vehicle to understand its environment and to react and predict.

However, there are some limitations to existing world models [38, 40, 39, 41]. Most existing world models use only RGB images as input and operate in simple, low-resolution environments [8, 119, 118], which makes them unable to cope with the complexity of autonomous driving. Autonomous driving mostly operates in cities with a large number of unpredictable variables. Since collecting real-world data is extremely expensive, we turned to the CARLA [27] driving simulator for data collection. CARLA can not only provide the variety of sensors needed for autonomous driving but also covers complex urban driving scenarios. This allows us to collect training data more easily and comprehensively. To better understand the environment, our model elevates the standard by not only utilizing high-resolution RGB images but also integrating LIDAR point cloud data. While RGB images provide valuable color information, which helps to more easily extract semantic information like traffic signals and signs, they lack the capacity to offer 3D spatial information. Conversely, LIDAR point clouds present a detailed 3D geometric snapshot of the environment but fall short of capturing traffic semantics. By combining these two types of data, we can get comprehensive information about the environment, helping our model to achieve a more holistic understanding.

Fusion of images and point clouds can be achieved in a number of ways. Data-level fusion, feature-level fusion, and decision-level fusion [23, 127, 51]. Direct data-level fusion at the structure level, either by projecting the LIDAR point cloud onto RGB images or by augmenting the LIDAR point cloud with image characteristics, has its limitations: the former introduces significant geometric distortions, while the latter sacrifices rich image information [74]. Meanwhile, decision-level fusion processes the data of different modalities separately to obtain the output and then integrates the results. In this method, the data of various modalities are processed separately, and the information of each modality is not merged interactively. Therefore we chose feature-level fusion. For image feature extraction, we adopted two strategies: using ResNet [44] to extract features from images directly or employing a deep neural network to derive a bird's eye view (BEV) representation of features [86]. Regarding the feature extraction of LiDAR point clouds, the field is evolving rapidly with many advanced techniques [88, 89, 155, 143, 63, 134, 135, 139, 140]. We employed two of them. One is to represent the point clouds in range view [134, 135, 139, 81, 29] and then use ResNet for feature extraction, and the other is to get their BEV features through PointPillars [63]. For feature fusion, we have also adopted various approaches. One involves simply averaging the features [18], and another entails stacking the features and passing them through a fully connected layer [74]. Alternatively, we employed the attention mechanism of a Transformer [121] to enable feature interaction between different modalities [20, 98, 141]. As for the latent states of the model, we adapted them to 2-dimensional to obtain a more comprehensive environmental contextual compared to the 1D version.

To further deepen the model's understanding of the environment, we reconstructed not only the input image and point cloud data but also the occupancy voxel representation [150, 69, 10, 52]. Voxels are structured data that provide a clearer representation of the environment than images and point clouds.

In the following, we first provide an introduction to the basic methods and building blocks of our approach in depth in Chapter 2. Then in Chapter 3, we introduce some state-of-the-art methods in the fields covered by our model to explain our choices. Afterwards, we detail the complete architecture of our approach in Chapter 4. In Chapter 5 we present the experimental setup we use to test the approaches and subsequently evaluate those experiments in Chapter 6. Finally, we summarize the conclusions of this thesis and provide an outlook on future research in Chapter 7.

2 Background

In this chapter, we begin with a comprehensive introduction to the world model [38, 40, 39, 41], which serves as the foundational framework for our model. This can provide readers with a basic concept of our research. Subsequently, we offer a brief overview of ResNet [44] as a feature extraction model and the range view as a representation of point clouds. Finally, we recap the Transformer [121], an advanced model that has proven its effectiveness in various tasks and has been widely applied.

2.1 World Model

World model [38] was first proposed in 2018 as part of model-based reinforcement learning. It is inspired by human perception of the world. The image of the world in the human mind is just a model. Instead of imagining specific images of objects, we humans conceptualize complex images and remember the relationships among them [30]. For example, when we see a car waiting at a red light, what we remember is not this complete image, but the abstraction of the phrase "a car is waiting at a red light." When needed, we can reimagine the image from this phrase through our mental model. We are able to observe a scene and remember its abstract description [91, 16], and then predict future sensory data based on current actions [58, 65]. Our predictions do not only rely on current observations but also take into account what has happened in the past. Therefore, the world model consists of two parts, the vision (V) model and the memory (M) model, as shown in Figure 2.1. The V model is based on a Variational Autoencoder (VAE) [60], which is used to encode observations into low-dimensional latent states; the M model is based on Recurrent Neural Network (RNN), which is used to predict future latent states based on historical information.

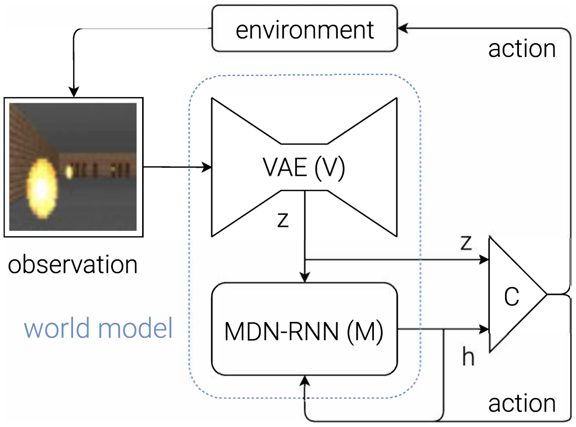


Figure 2.1: World Model. reprinted from [38].

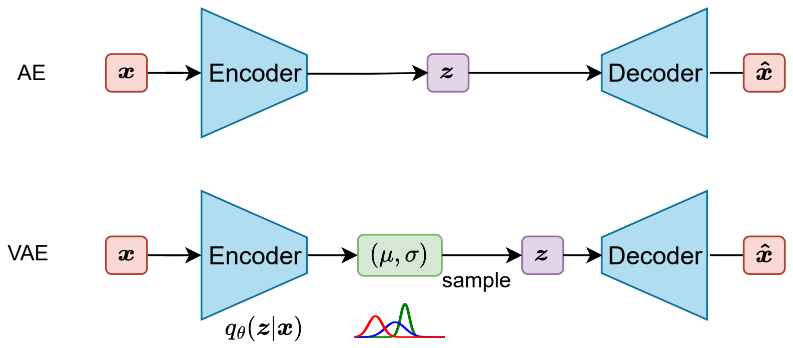


Figure 2.2: Variational Autoencoder (VAE) compared to Autoencoder (AE).

2.1.1 Vision (V) Model

During the process of interacting with the environment, the model receives observations as input at each time step. The observations are high-dimensional representations such as images, point clouds, etc. What the V model does is learn abstract, compressed representations of those high-dimensional inputs. The world model uses a Variational Autoencoder (VAE) as the V model.

VAE itself is a generative model that discerns the latent variables $\boldsymbol{z}$ from the data $\boldsymbol{x}$ and then generates the data $\hat{\boldsymbol{x}}$ from these latent variables $\boldsymbol{z}$. Unlike the Autoencoder (AE) [96], which constructs the latent variable directly with a concrete numerical vector, VAE samples the latent variable from a Gaussian distribution obtained from the encoder, as shown in Figure 2.2. That is to say, the encoder of VAE essentially uses a neural network $q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) \sim \mathcal{N}(\mu, \sigma \boldsymbol{I})$ to approximate the posterior probability distribution $p(\boldsymbol{z}|\boldsymbol{x})$ of latent variable within the original data.

2.1.2 Memory (M) Model

After the V model compresses the observation information, the role of the M model is to integrate all these abstract, compressed representations over time and make predictions about future latent variables. M model can also be called transition model.

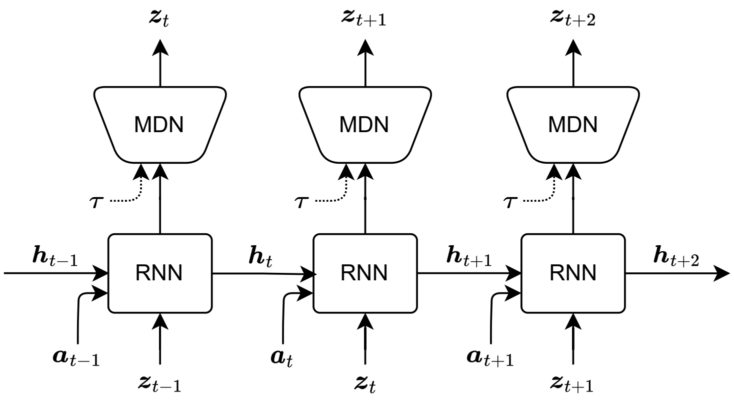


Figure 2.3: Visualization of MDN-RNN. reprinted from [38].

MDN-RNN

In the original world model, the M model consists of a Recurrent Neural Network (RNN) and a Mixture Density Network (MDN) [4, 5, 36] as shown in Figure 2.3. The RNN is utilized to integrate the historical information $\boldsymbol{h}_{t+1} = \text{RNN}(\boldsymbol{h}_t, \boldsymbol{z}_t, \boldsymbol{a}_t)$; the MDN is used to predict the probability distribution $P(\boldsymbol{z}_{t+1}|\boldsymbol{h}_{t+1})$ of the next latent variables $\boldsymbol{z}_{t+1}$ based on the historical information $\boldsymbol{h}_{t+1}$. When there is observational input, the M model accepts the observation latent variables $\boldsymbol{z}_t$, outputting $(\boldsymbol{h}_t, \boldsymbol{z}_t)$ as the state $\boldsymbol{s}_t$ at time t. In the absence of observation input, the MDN can predict the distribution of the latent variable $P(\boldsymbol{z}_t)$ and sample $\hat{\boldsymbol{z}}_t$ from it as input. Where $\boldsymbol{a}$ is the action, $\boldsymbol{h}$ is the hidden state of the RNN, $\boldsymbol{z}$ is the latent variables of observations obtained from the V model, P is the probability distribution, and the subscript denotes the time step.

Recurrent State Space Model (RSSM)

The Recurrent State Space Model (RSSM) was introduced in PlaNet [40]. By incorporating the deterministic path into the stochastic model (SSM), RSSM enables the model to remember historical information over multiple time steps while predicting multiple futures. RSSM has been applied in many world models that followed, such as the Dreamer series [39, 41, 42], MILE [48], and the model in this thesis also uses RSSM as the transition model.

RSSM uses a sequence of deterministic historical states $\boldsymbol{h}_t$ to calculate two distributions for stochastic states: the posterior and the prior. The posterior state $\boldsymbol{s}_t$ integrates the current observation information $\boldsymbol{z}_t$, while the prior state $\hat{\boldsymbol{s}}_t$ is predicted in the absence of observations. As shown in Figure 2.4. The components can be expressed as:

$$\begin{aligned}
&\text{Deterministic state model:} && \boldsymbol{h}_t = f_\phi(\boldsymbol{h}_{t-1}, \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}) \\
&\text{Representation model:} && \boldsymbol{s}_t \sim q_\phi(\boldsymbol{s}_t|\boldsymbol{h}_t, \boldsymbol{z}_t) \\
&\text{Transition predictor:} && \hat{\boldsymbol{s}}_t \sim p_\phi(\hat{\boldsymbol{s}}_t|\boldsymbol{h}_t) \\
&\text{Observation model:} && \hat{\boldsymbol{o}}_t \sim p_\phi(\hat{\boldsymbol{o}}_t|\boldsymbol{h}_t, \boldsymbol{s}_t)
\end{aligned} \tag{2.1}$$

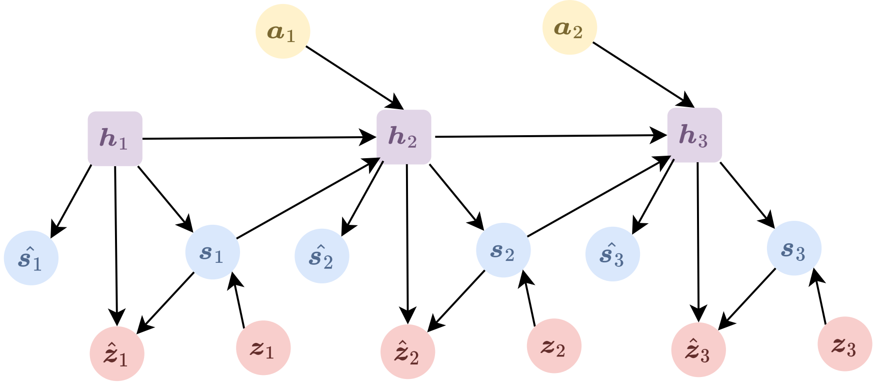


Figure 2.4: Visualization of Recurrent State Space Model (RSSM). Adapted from [40, 39].

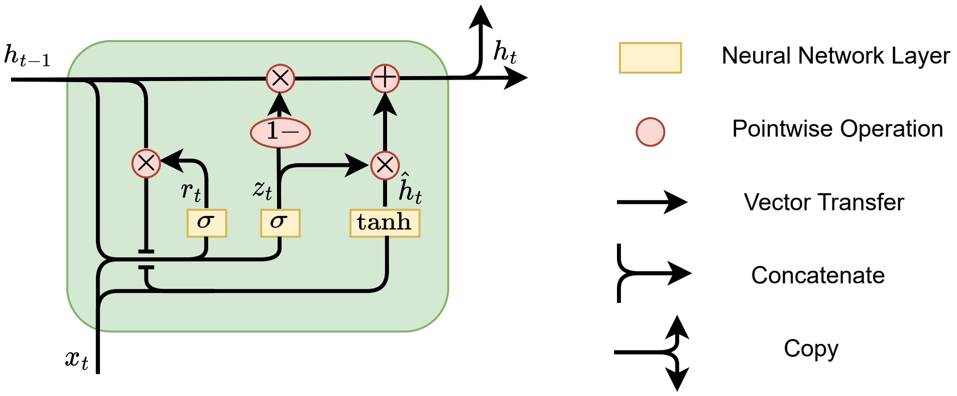


Figure 2.5: Structure of Gated Recurrent Unit (GRU).

All components are approximated by neural networks, where ϕ represents the combined learnable parameters of these networks. The deterministic state model $f_\phi(\boldsymbol{h}_{t-1}, \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1})$ is modelled as an RNN, to retain deterministic historical information. The representation model derives the distribution of stochastic state $\boldsymbol{s}_t$ from historical information $\boldsymbol{h}_t$ as well as current observations $\boldsymbol{z}_t$, serving as the posterior distribution. The transition predictor guesses the prior distribution of stochastic state $\hat{\boldsymbol{s}}_t$ only from the previous stochastic state $\boldsymbol{s}_{t-1}$ with the action $\boldsymbol{a}_{t-1}$, without relying on the observations $\boldsymbol{z}_t$. So it can be used for prediction. The observation model reconstructs the observation features $\hat{\boldsymbol{o}}_t$ based on both deterministic state $\boldsymbol{h}_t$ and stochastic state $\boldsymbol{s}_t$. During training, the RSSM always takes observations as inputs and simultaneously computes both the posterior and the prior stochastic state distributions. The observations are always reconstructed from the posterior states $\boldsymbol{s}_t$. The posterior states are trained through the reconstruction error, and then the prior state distribution is trained to approximate the posterior state distribution.

2.1.3 Gated Recurrent Unit (GRU)

The RNN in the transition model (M model) has many implementations, and the most commonly used of them are long short-term memory (LSTM) [46] and gated recurrent unit (GRU) [21]. Compared to LSTM, GRU has similar performance but more efficient. In this thesis, we used a GRU as the recurrent model, so the next is a review of GRU.

In the Gated Recurrent Unit (GRU), the most crucial components are the two gate units: the reset gate and the update gate. The reset gate determines how to incorporate historical information into the new input to obtain the candidate hidden state. The update gate controls how historical memory is carried into the current state. Figure 2.5 shows the structure of a GRU, and the following equations demonstrate how the GRU updates its state:

$$\begin{aligned} u_t &= \sigma(W_u \cdot [h_{t-1}, x_t]) \\ r_t &= \sigma(W_r \cdot [h_{t-1}, x_t]) \\ \hat{h}_t &= \tanh(W_h \cdot [r_t \odot h_{t-1}, x_t]) \\ h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot \hat{h}_t \end{aligned} \qquad [2.2]$$

Where σ and tanh represent the sigmoid and tanh activation functions, respectively; W denotes the parameters of each neural network; $\odot$ denotes element-wise multiplication; x_t is the input information at the current time step; h_t is the hidden state of GRU, which also serves as historical memory, retaining past information; u_t is the update gate, and r_t is the reset gate, both determined by the previous hidden state h_{t-1} and the current input x_t; $\hat{h}_t$ is the candidate hidden state, formed by combining the current input x_t with the previous hidden state h_{t-1} selected by the reset gate r_t; the final output is the hidden state h_t, which is composed of the addition of two parts: the previous hidden state h_{t-1} after forgetting and the candidate hidden state $\hat{h}_t$ after further selection, both by the update gate u_t.

2.2 ResNet

Much evidence [103, 110] shows that the depth of the neural network is crucial to the model's performance. When the number of network layers is increased, the network can extract more complex feature patterns, so in theory, better results can be achieved when the model is deeper. However, training deep networks is not easy. The first is the well-known exploding/vanishing gradient problem [3, 33], which fundamentally hinders convergence. But this problem has been solved to a large extent by batch normalization [53]. However, even so, deep networks suffer from degradation problems [43, 108, 44]. As shown in Figure 2.6, after exceeding a certain number of layers, the training error of the model will increase. The original intention of ResNet is to solve this problem [44].

ResNet solves the degradation problem of deep neural networks by introducing residual learning and identity mapping. ResNet does not directly fit the required underlying mapping by stacking multiple nonlinear layers, but lets them fit the residual mapping. Denote stacked nonlinear layers as

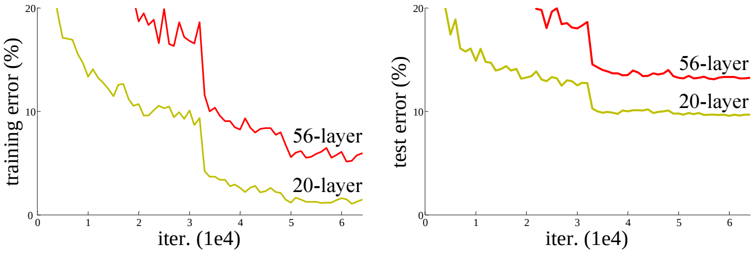


Figure 2.6: degradation phenomena. reprinted from [44].

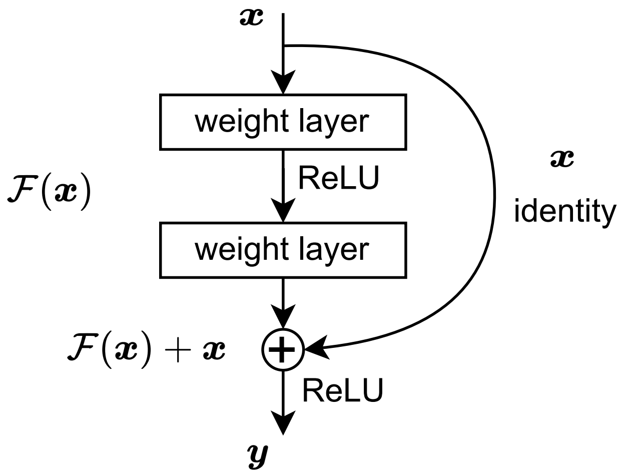


Figure 2.7: A building block of residual learning. Reprinted from [44].

$\mathcal{F}(\boldsymbol{x})$, assuming that it can asymptotically approximate any complex function. $\mathcal{H}(\boldsymbol{x})$ denotes the required underlying mapping, i.e. the learned feature representation. According to the hypothesis, if $\mathcal{F}(\boldsymbol{x})$ can fit $\mathcal{H}(\boldsymbol{x})$, then it can also fit $\mathcal{H}(\boldsymbol{x}) - \boldsymbol{x}$. Therefore, compared with traditional methods that directly use $\mathcal{F}(\boldsymbol{x})$ to fit $\mathcal{H}(\boldsymbol{x})$, ResNet explicitly makes these layers fit the residual function $\mathcal{F}(\boldsymbol{x}) := \mathcal{H}(\boldsymbol{x}) - \boldsymbol{x}$. Then the feature becomes $\mathcal{F}(\boldsymbol{x}) + \boldsymbol{x}$. The network needs to learn the difference between the feature and the input, rather than learning the feature directly.

In deeper networks, if the added layers can be learned as identity mappings, then the performance of deeper networks should not decrease, which is not the case. This shows that the stacked nonlinear layers have difficulty learning identity mapping. Thus, in the case of using a residual structure, if the identity mapping of input $\boldsymbol{x}$ is the optimal solution, then the identity mapping can be simulated by simply driving the residual to zero $\mathcal{F}(\boldsymbol{x}) \rightarrow 0$. This is much easier than learning identity mapping directly. If the identity mapping is not optimal, then it is only necessary to refer to identity $\boldsymbol{x}$ for fine-tuning instead of learning an entirely new function.

The basic architecture of ResNet consists of building blocks, as shown in Figure 2.7. Each block has two paths, one for the residual mapping through the stacked nonlinear layers and one for the identity mapping. The outputs of the two paths are then added element-wise. It can be expressed as $\boldsymbol{y} = \sigma(\mathcal{F}(\boldsymbol{x}, \{W_i\}) + \boldsymbol{x})$, where $\boldsymbol{x}, \boldsymbol{y}$ are the input and output of the residual block. The function $\mathcal{F}(\boldsymbol{x}, \{W_i\})$ represents the stacked nonlinear layers used to learn the residual mapping. In the example of Figure 2.7, $\mathcal{F}$ contains two weight layers with non-linearity provided by a ReLU function in the middle. $\mathcal{F}(\boldsymbol{x}) + \boldsymbol{x}$ is a shortcut connection. When $\mathcal{F}$ and $\boldsymbol{x}$ have the same dimension, they are directly added element by element. When the dimensions are different, a linear projection W_s is performed in the shortcut connections to match the dimensions: $\mathcal{F}(\boldsymbol{x}, \{W_i\}) + W_s\boldsymbol{x}$. σ is the nonlinear layer ReLU [82] and is used as the activation layer for the final output. In ResNet, the weight layer is the convolutional layer.

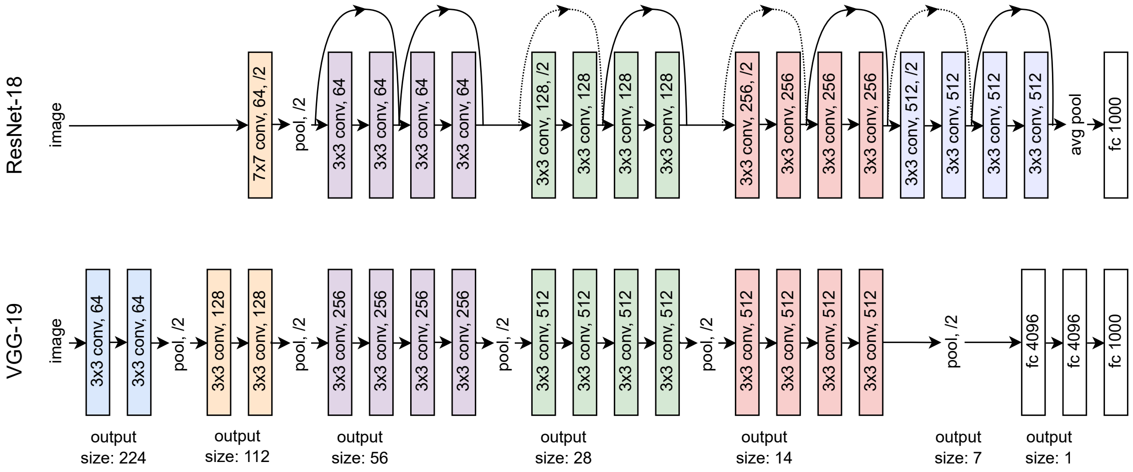


Figure 2.8: ResNet-18 compared to VGG-19. Adapted from [44].

Table 2.1: Architectures of ResNet with the different number of layers. Reprinted from [44].

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

The structure of ResNet is similar to VGG net [103] but incorporates residual learning through shortcut connections. Figure 2.8 shows a comparison of the 18-layer ResNet with the VGG19 network. ResNet follows several design principles: 1) the convolutional layers mostly use 3×3 kernels and use the same number of kernel filters for feature maps with the same output size, represented by the same colours; 2) the number of kernel filters is doubled when the size of the feature map is halved. Convolutional layers with a stride of 2 are used for downsampling. Shortcut connections with dashed indicate that downsampling is performed and the dimensionality is changed. Table 2.1 shows the architectures of ResNet with the different number of layers. Brackets indicate the shape of the convolutional kernel within the residual building block and the number of blocks stacked in each layer. The output shapes in the table are when using the ImageNet dataset [97]. In practice, when using ResNet for feature extraction, any shape can be adapted, and the output of a layer can be used directly as a feature map.

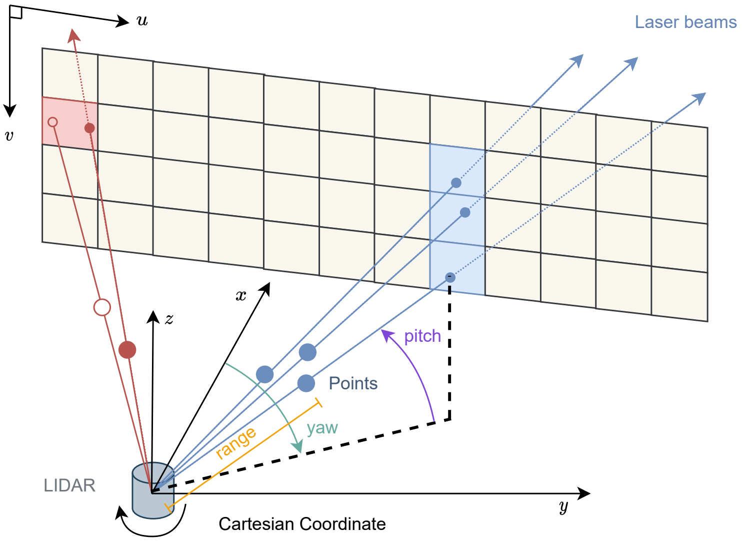


Figure 2.9: The illustration of range view projection. Adapted from [29, 61].

2.3 range view

Range view [134, 135, 139, 81, 29, 140, 61] is a representation of point clouds. Since the quantity of point clouds generated by LiDAR scanning is not fixed, it becomes challenging to utilize neural networks for feature extraction. Range view transforms point clouds into an image-like representation, allowing for the straightforward application of convolutional networks such as ResNet for feature extraction, similar to how they are used with images.

LiDAR emits N laser beams at the same angular intervals in the vertical direction (pitch), and these beams rotate around the LiDAR at a certain frequency in the horizontal direction (yaw). The laser beams are partially reflected back when encountering an object, providing the distance from that point to the LiDAR. Consequently, the point cloud can be represented not only by Cartesian coordinates with (x, y, z) but also by cylindrical coordinates formed by the yaw, pitch, and distance between the point and the LiDAR, as illustrated in Figure 2.9. The conversion relationship between the two coordinate systems can be expressed as:

$$\begin{aligned} x &= r \cdot cos(\phi) \cdot cos(\theta) \\ y &= r \cdot cos(\phi) \cdot sin(\theta) \\ z &= r \cdot sin(\phi) \end{aligned} \tag{2.3}$$

Here, θ, ϕ, and $r = \sqrt{x^2 + y^2 + z^2}$ are the yaw angle, pitch angle, and distance between the point and LiDAR sensor, respectively. Subsequently, each point can be projected onto a 2D cylindrical projection $\mathcal{R}(u, v)$ with dimensions $H_r \times W_r$ based on yaw angel $\theta = \tan^{-1}\frac{y}{x}$ and pitch angel $\phi = \sin^{-1}\frac{z}{r}$:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\left(1 - \arctan(y, x)\pi^{-1}\right) W_r \\ (1 - (\arcsin(z, r^{-1}) + |f_{\text{down}}|) f_{\text{v}}^{-1}) H_r \end{pmatrix} \tag{2.4}$$

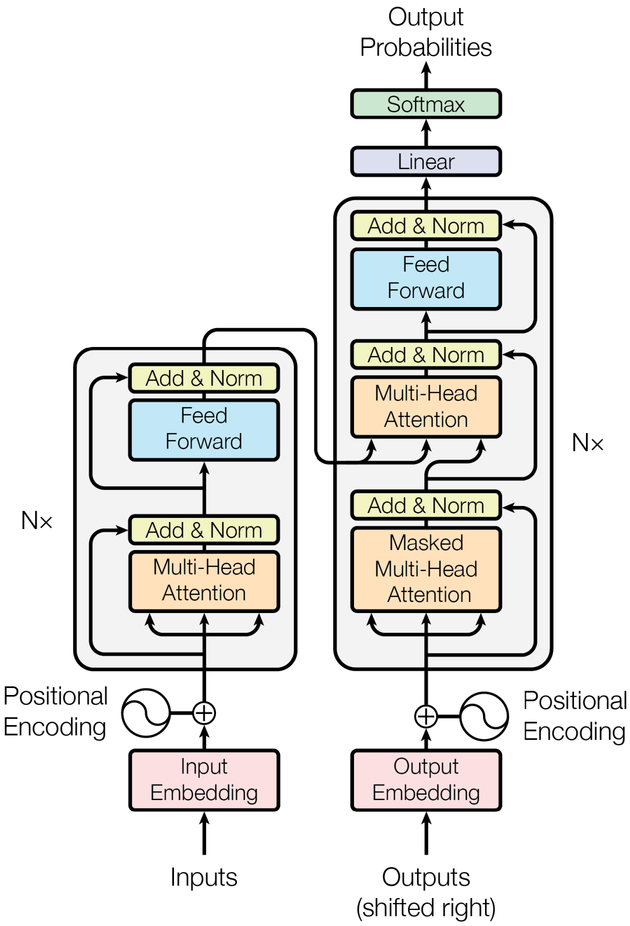
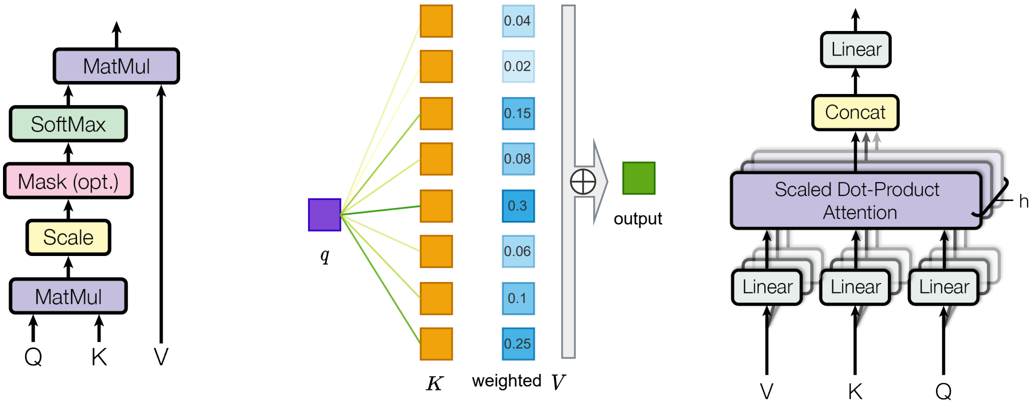


Figure 2.10: The structure of Transformer. Reprinted from [121].

where $f_v = |f_{up} + f_{down}$ represents the vertical filed-of-views (FOVs) of the LiDAR sensor and f_{up}, f_{down} are the FOVs at upward and downward directions, respectively. (u, v) denotes the pixel coordinate of each point in the 2D range view image $\mathcal{R}(u, v)$. The (x, y, z, r), formed by stacking the Cartesian coordinates (x, y, z) and distance r of each projected point along the dimension axis, serves as the features of that pixel, resulting in $\mathcal{R}(u, v) \in \mathbb{R}^{4\times H\times W}$. For pixels without projected points, the coordinates (x, y, z) are set to 0, and r is designated as -1, effectively indicating that there are no points at these coordinates. If two points are projected onto the same pixel, the closer point is taken to project, as shown by the red projection in Figure 2.9, where the solid circle indicates that it is selected. In practice, the dimensions of the range view image are determined based on the LiDAR's beam count and rotation frequency to prevent multiple points from being projected onto the same pixel.

2.4 Transformer

The Transformer, introduced by Vaswani et al. [121] in 2017, was originally designed for processing sequential data. Its self-attention mechanism can dynamically focus on different parts of an input sequence and their relationship, thereby improving its ability to concentrate on crucial information.



(a) Scaled Dot-Product Attention (b) Example of Attention with one query (c) Multi-Head Attention

Figure 2.11: Attention. (a) and (c) are reprinted from [121].

Therefore, it was subsequently expanded to other fields like computer vision and sensor fusion, demonstrating impressive performance. The Transformer consists of an encoder (left part of Figure 2.10) and a decoder (right part of Figure 2.10), each constructed from multiple stacked encoder layers and decoder layers respectively. Each encoder layer and decoder layer contains some Feed-Forward layers, Add&Norm layers, and Multi-Head Attention layers. The feed-forward layer is implemented as a simple fully connected network. The Add&Norm layer employs a residual connection followed by layer normalization.

2.4.1 Attention

Vaswani et al. [121] named the attention mechanism they used as "Scaled Dot-Product Attention" (Figure 2.11a). The inputs to the attention module are queries (Q), keys (K), and values (V), where the dimensions of Q and K are d_k, and the dimension of V is d_v. By querying each key, the correlation (attention weight) of each key to that query is determined. Then the values corresponding to the keys are weighted and added according to this correlation to produce the output for that query (Figure 2.11b). In practice, this is computed synchronously using matrices:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad [2.5]$$

In Addition, the dimensions of Q, K, V are split into h parts, making the dimensions of each Q, K become d_k/h and for each V is d_v/h. Scaled Dot-Product Attention is performed synchronously on each Q, K, V group, yielding h outputs with dimension d_v/h. These outputs are then concatenated to produce the final output with dimension d_v, as shown in Figure 2.11c.

2.4.2 Positional Encoding

The attention module is not sensitive to position, in other words, changing the order of tokens in the input sequence does not affect the attention weights. Hence, it's necessary to inject position information about the tokens into the input sequence. In Transformer, a sine and cosine positional encoding is added to the input embeddings. The formula for this positional encoding is:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \end{aligned} \quad [2.6]$$

where pos represents the position and i represents the dimension. d_{model} is the total dimension of the position encoding, which matches the input's dimension so that the two can be directly added together.

3 Related Work

In this chapter, we delve into the current state of research in the three domains: world models, multimodal sensor fusion, and 3D occupancy mapping. Our work is at the intersection of these three research fields. By reviewing these areas, we analyze the strengths and limitations of related works and draw inspiration from them.

3.1 World Model

World models are generative models that encode observations into latent states and predict future states based on actions. These latent states can then be decoded back into the observation space [7, 6, 64]. Originating from model-based reinforcement learning [109], world models serve as simulators to emulate environmental dynamics, reducing the direct interaction between the controller and the environment. This allows the agent to model the complex world while maintaining a simplified action search space.

3.1.1 RNN-based World Model

Ha et al. [38] initially introduced the template for modern world models. They used Variational AutoEncoders (VAEs) [60] to encode observations and Recurrent Neural Networks (RNNs) to model latent state dynamics. PlaNet [40] subsequently introduced the Recurrent State Space Model (RSSM), enhancing the model's ability to capture dynamics by combining stochastic states and deterministic paths. This allows the model to retain the environmental randomness while remembering long-term historical information. In the subsequent Dreamer series [39, 41, 42], they continuously improved this approach by using discrete latent state representations, employing the KL balancing trick, and adding symlog predictions.

3.1.2 Transformer-based World Model

Given the effectiveness of Transformers [121] in tasks involving long-term dependencies and memory-based predictions compared to RNN models, many methods [17**?** , 79, 149] have chosen to use Transformers for dynamic modelling. TransDreamer [17] directly replaces the Gated Recurrent Units (GRUs) [21] in Dreamer with Transformer structure. IRIS [79] treats dynamic learning of latent stats as a sequence modelling problem, using VQ-VAE [120] to encode images into tokens and employing a spatial-temporal Transformer to capture information between images and make predictions. STORM [149] utilizes the stochastic nature of VAEs to improve the operational efficiency of Transformer-based world models. Additionally, many studies [37, 105, 77, 24] have focused on improving the long dependencies and operational efficiency of world models based on Structured State Space Sequence (S4) [37] layers.

3.1.3 High-Resolution Image as Input

These methods have demonstrated excellent performance in simple simulated environments [8, 119, 118], but they typically use low-resolution images as inputs, which may not be sufficient for simulations in autonomous driving. The demand for higher-resolution and more complex environmental understanding in autonomous driving applications necessitates models capable of processing and predicting based on high-resolution data. Consequently, recent research has been dedicated to world models that run in complex environments with high-resolution images, and significant progress has been made in this area. MILE [48] is a model-based imitation learning method that adopts an architecture similar to Dreamer. It operates within the CARLA simulation environment [27], using high-resolution images to learn expert driving strategies. MILE enhances the model's understanding of the environment by leveraging predicted BEV (Bird's Eye View) semantic segmentation labels during training. This additional task guides the model to focus on relevant features and structures within the environment that are crucial for making informed decisions. Similarly, Gao et al. [31] proposed a semantic masked recurrent world model (SEM2), which trains an additional latent feature filter by reconstructing BEV semantic masks. This process aims to eliminate task-irrelevant information from high-resolution observations. By focusing on BEV semantic masks, SEM2 can distil the important features from the environment that are necessary for the task at hand, reducing the noise and complexity that come with high-resolution inputs. However, they rely on additional information to enhance their effectiveness, which limits their scalability.

3.1.4 Video Generation

Compared to using privileged information as supervision to reinforce the model's learning of driving strategies, some world models [59, 125, 49, 67] focus on video generation, which encompasses image reconstruction and prediction. This shift in focus represents a different approach to understanding and interacting with the environment. DriveGAN [59] is trained unsupervised based on real-world data and can generate high-resolution and high-fidelity images conditioned on actions. It employs VAE and GAN [34, 57] to learn the latent space of images. A distinctive feature of DriveGAN is its ability to disentangle the latent space into content-related and style-related factors, thereby providing additional control over the environment.

Due to the impressive performance of diffusion-based models [95, 106, 45, 144] in image and video generation, attention has turned towards diffusion models as a powerful tool for creating high-quality visual content. DriveDreamer [125] is one such model that enhances the representation of complex environments by incorporating a diffusion model. It employs a two-stage training process. In the first stage, the model is trained to understand structured traffic information by using HD maps and 3D bounding boxes as conditions, and in the second stage, the environment dynamics, i.e., the model's ability to predict futures, are trained. They further utilise CLIP [92] embedding to guide the style of predictions.

Hu et al. interpreted world models as single sequence models [19, 54, 79, 93, 72, 136, 22] and demonstrated their Generative AI for Autonomy (GAIA-1) [49] model. This model is trained on proprietary real-world camera data and can be modulated through actions and textual inputs. They utilized vector quantization [120] to label data and enrich semantic content through DINO distillation [12]. With a video diffusion decoder, they achieved temporally consistent high-resolution predictions.

3.1.5 3D Application

Zhang et al. [148] integrated a diffusion model into a spatial-temporal Transformer for predictions, applying it to point cloud prediction. OccWorld [154] expands the concept of world models to 3D voxel occupancy, offering a more comprehensive understanding of the environment by representing not just where objects are, but also the 3D space they occupy.

3.2 Sensor Fusion

In autonomous driving, sensor fusion typically involves combining data from cameras and LiDAR. During the fusion process, feature extraction is performed on data from both sources.

3.2.1 Encoder for Camera and LiDAR

Camera

For images, traditional convolution-based feature extractors [44, 110, 111, 112, 47, 62] are commonly used. However, with the evolution of Transformers [121] in computer vision in recent years, Transformer-based image feature extraction methods [26, 73] have become popular.

LiDAR

For point clouds, there are point-based methods like PointNet [88] and PointNet++ [89], which operate directly on points through Multilayer Perceptrons (MLPs). Subsequent methods [1, 117, 137, 128] have optimized this type of approach further. Another popular trend involves assigning points to voxels and then integrating the points within each voxel to derive voxel features. The most notable among these is VoxelNet [155], which has been applied as a voxel feature extractor in many models. Recently, much work [25, 35, 50, 63, 99, 100, 114, 146] has been dedicated to optimizing the efficiency of voxel-based point cloud decoders. PointPillars [63] use pillars to replace voxels.

Some approaches [134, 135, 139, 81, 29, 61] opt to project point clouds into 2D spaces like range view or BEV, after which image decoders can be directly applied. RPV [140] combines three types of methods to get performance and efficiency gains.

3.2.2 Fusion Method

Many methods [141, 20, 98] encode the features extracted from both sensors into tokens for fusion using a Transformer [121]. Using Bird's Eye View (BEV) as an intermediary space for data of both modalities is a popular approach [71, 74, 76, 55, 147]. Features from both sensors are mapped to BEV [86] for fusion. Other methods [68, 152] opt for fusion in the information-richer voxel space, which retains a detailed 3D context for the data.

For 3D object detection tasks, dense feature representations might be redundant, prompting some methods [9, 66, 138] to choose sparse representations to filter out background noise. Some approaches [87, 145, 153] generate candidate boxes from images to guide LiDAR prediction. Conversely, Transfusion [2] employs point clouds to prevent failures caused by low-quality images. PointPainting [122] enhances point cloud data by directly appending semantic segmentation results obtained from images to the corresponding points in the cloud, thus enriching the point cloud with additional semantic information. Similarly, MVX-Net [104] appends features extracted from images using pre-trained convolutional networks to the point cloud.

Other works focus on different aspects of sensor fusion and processing. For instance, Fast-CLOCs [85] emphasizes the real-time performance of the model, making it suitable for applications where rapid processing is critical. RPEFlow [124] deals with optical flow and scene flow, providing detailed motion analysis. SupFusion [90] introduces auxiliary supervision to enhance learning and improve model performance.

3.3 3D Voxel Reconstruction

Although 2D Bird's Eye View (BEV) representations are prevalent in autonomous driving, they can't fully capture the complexity of environments. 3D occupancy, typically modelled by voxels, offers a geometrically perceptive representation of the scene [7, 102].

MonoScene [10] estimates 3D semantic voxels using a single-view camera image, not only reconstructing visible areas captured by the camera but also imagining occluded regions that are not visible. It introduced new losses to refine Scene-class Affinity, optimizing both globally and locally.

3.3.1 Transformer-based

Some approaches have introduced Transformers [121] to enhance the 3D voxel occupancy reconstruction effect. Voxformer [69] begins by estimating a set of sparse voxels as queries, continuously inquiring about spatial information from image observations to generate denser voxels. Occformer [150], on the other hand, extracts different aspects of information through a global path and a local path, ultimately aggregating to form a more complete spatial representation. Symphonies [56] recognize the impact of instance semantics and scene context on voxels, encoding instance-centered semantics to strengthen the connection between voxels and pixels belonging to the same instance. It also promotes the model's understanding of the overall scene through scene context.

3.3.2 Unsupervised Training

Annotating real-world 3D voxel occupancy grids is extremely time-consuming and labour-intensive. Therefore, Tan et al. proposed a zero-shot learning approach, Open Vocabulary Occupancy (OVO) [113, 83, 84, 133], eliminating the dependence on supervision during training. Some innovative methods [11] employ NeRF [80] to complete the rendering of 3D voxel grids from 2D images, no longer relying on annotations for 3D occupancy. CLONeR [15] employs both Lidar sensors and cameras, enhancing NeRF's performance under sparse sensor input by decoupling the learning of color and occupancy. It utilizes images to supervise color learning and point cloud for geometry learning, thereby improving the overall effectiveness and accuracy of NeRF in scenarios with limited sensor data.

3.3.3 Multi-view

There are also many methods [28, 52, 78, 70, 123, 126, 130] specifically designed for multi-view image inputs. OccDepth [78] reconstructs voxels by extracting the implicit depth information from stereo cameras. TPVFormer [52] extends BEV feature mapping to Tri-Perspective Views (TPV), integrating the features of a point's projection positions in three views as its comprehensive feature. In contrast to mapping the features from each view's image to TPV 2D planes, SurroundOcc [130] operates directly in 3D space through 2D-3D spatial attention, obtaining 3D features. There are also some works [94, 107, 142] completing the semantic scene based on point clouds.

4 Method

In the following sections, we provide a detailed introduction to our model. As illustrated in Figure 4.1, our model's foundational pipeline is divided into four parts: the Encoder, sensor fusion, transition model, and decoder. The model accepts RGB images and point clouds, which are typical for autonomous vehicles [32, 129], as inputs. These data are then separately processed through the Encoder for feature extraction. Subsequently, the features from both sensors are fused together. The fused features are then fed into the transition model, which retains historical information and generates latent states determined by actions. Finally, these latent states are decoded into the required multimodal data. In the following sections, we further explain these four components.

4.1 Encoder

ResNet is used for feature extraction in many methods and shows excellent performance. Thus, our Encoder also employs ResNet as the backbone. In practice, we extract three feature maps of varying sizes from three layers of ResNet and add them together (by upsampling, downsampling, and convolutional layers to match their dimensions and sizes) to obtain features with different scales.

4.1.1 Image

For image feature extraction, a simple way is to utilize ResNet to extract multiscale features directly. An additional option is to map the features extracted through ResNet into Bird's Eye View (BEV). The advantage is that BEV is a shared space for both images and point clouds, allowing both to be projected into BEV for fusion in this unified space. [71, 74, 76]

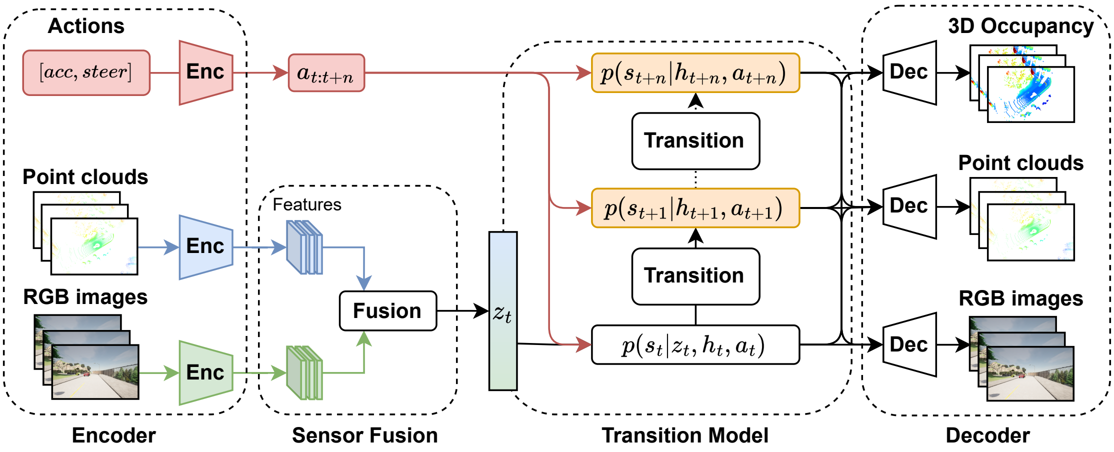


Figure 4.1: The pipeline of our model. Reprinted from [7].

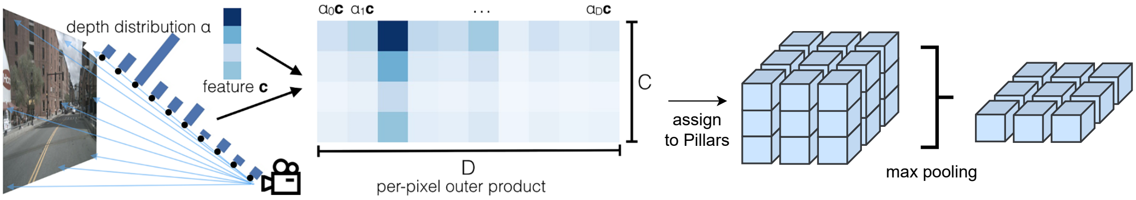


Figure 4.2: The BEV Mapping in LSS. Reprinted from [86].

BEV Mapping

We follow the procedure outlined by J. Philion et al. in LSS [86] for mapping image features to BEV. For each pixel p with feature dimension C in the feature map $\mathcal{F}(h, w) \in \mathbb{R}^{C \times H_z \times W_z}$, we assign $|D|$ depths, where D is a set of discrete depths $D = \{d_0 + \Delta, \ldots, d_0 + |D|\Delta\}$. Thus resulting in a point cloud collection of size $|D| \cdot H_z \cdot W_z$. For each pixel p in the feature map, there are features $\boldsymbol{c} \in \mathcal{R}^C$, and depth distribution $\alpha \in \Delta^{|D|}$ estimated by a neural network (converted to probabilities via softmax). For the point p_d at depth d corresponding to pixel p, define its features $\boldsymbol{c}_d$ as features $\boldsymbol{c}$ of pixel p scaled by its probability at this depth α_d:

$$\boldsymbol{c}_d = \alpha_d \boldsymbol{c}. \tag{4.1}$$

This operation yields a point cloud $\mathcal{P}(h, w, d) \in \mathbb{R}^{C \times H_z \times W_z \times |D|}$. For the image, its camera's extrinsic matrices $\boldsymbol{E}_k \in \mathbb{R}^{3\times4}$ and intrinsic matrices $\boldsymbol{I}_k \in \mathbb{R}^{3\times3}$ jointly define the mapping from world coordinates (x, y, z) to local pixel coordinates (h, w, d). With this mapping relation, the point cloud $\mathcal{P}(h, w, d)$ can be mapping to $\mathcal{P}(x, y, z) \in \mathbb{R}^{C \times X \times Y \times Z}$. The point cloud space is then divided into $H_b \times W_b$ pillars in the $x - y$ plane. By assigning each point to its nearest pillar and performing sum-pooling for each pillar, we can obtain a feature map with size $C \times H_b \times W_b$ in Bird's Eye View.

4.1.2 Point Cloud

Processing of point clouds typically falls into three categories: point-based, range view-based, and voxel-based. Point-based methods [88, 89] extract features of each point through multi-layer MLPs and obtain their local features by aggregating features of neighbouring points. However, point clouds are sparse, unstructured data with variable order and quantity, making the efficiency of searching for neighbouring points extremely low. Range view-based methods [134, 135, 139, 81, 29] project the distances of point clouds onto a 2D image, alleviating sparsity to some extent. Subsequent convolution towers can aggregate information over a large receptive field to further mitigate the sparsity problem of point clouds. Moreover, it is very efficient due to highly optimized 2D convolutions. However, since cylindrical projection, the size of objects in range view images is not distance-invariant. Voxel-based methods [155, 63, 143] maintain the physical size of objects with clear structure but are relatively sparse and require high resolution to prevent information loss due to quantization. An increase in 3D resolution results in a cubic consumption of memory and computational resources. Overall, the range view offers the best efficiency and is also convenient for reconstruction since it's image-like data; voxel-based methods perform well at high resolutions, but the efficiency depends on the resolution [140]. Hence, we chose range view and PointPillars [63], an efficiency-optimized voxel-based-like method, to process point clouds.

Range View

We straightforwardly use the Formula [2.4] described in Section 2.3 to project the point cloud onto a range view image $\mathcal{R}(u, v) \in \mathbb{R}^{4 \times H_r \times W_r}$. For this pseudo-image $\mathcal{R}(u, v)$, we can directly use ResNet to extract multiscale features. Since the range view projection is just a simple mapping transformation and does not involve any learnable parameters, it does not bring additional computational overhead and can be prepared in advance.

PointPillars

Firstly, discretize the point cloud space into uniformly distributed pillars on the x-y plane, where each pillar is equivalent to a voxel extending infinitely in the z-direction. Then assign each point to the corresponding pillar according to coordinates. For the point with coordinate (x, y, z), its global coordinates (x, y, z), its relative coordinates (x_c, y_c, z_c) to the geometric center of all points in the pillar it's located in, and the offsets (x_p, y_p) to the x, y center of the pillar it's located in are used as its features. Let D represent the number of point features, P the number of pillars, and N the limit of points per pillar. If a pillar contains more than N points, points are randomly sampled; if fewer, it is filled with zero-padding. From this, we obtain a dense tensor of size (D, P, N). Then use MLPs to extract features of each point, yielding point features $\mathcal{F}_n \in \mathbb{R}^{C \times P \times N}$ and perform max pooling within each pillar to obtain features $\mathcal{F}_p \in \mathbb{R}^{C \times P}$. Finally, unfolding these features back to their original pillar positions results in a 2D feature map $\mathcal{F}_p \in \mathbb{R}^{C \times H_p \times W_p}$ [63]. This 2D pseudo-image can then undergo further feature extraction using ResNet.

4.1.3 Others

In addition to images and point clouds, our model can also accept route maps or measurements such as velocity and Global Navigation Satellite System (GNSS) data as input. Route maps are image-like 2D data, so we use ResNet for feature extraction as well and use average pooling to

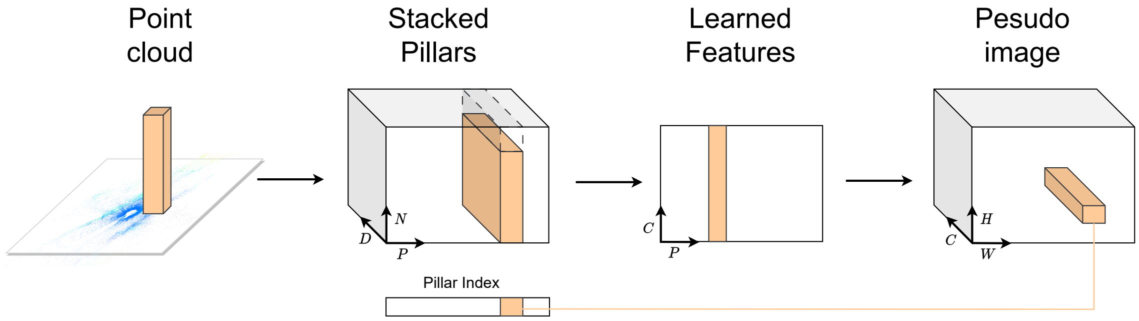


Figure 4.3: The Illustration of PointPillars. Reprinted from [63].

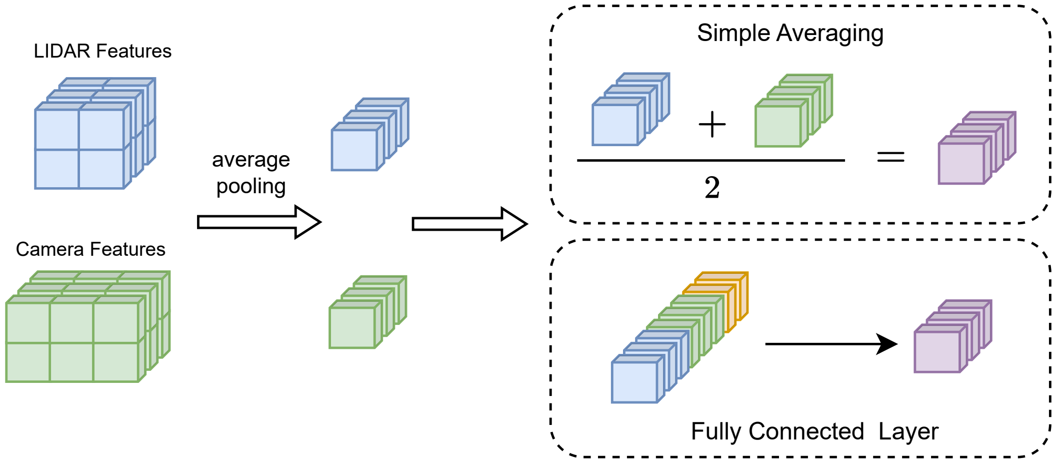


Figure 4.4: The Illustration of Simple Averaging Fusion (top) and Fully Connected Layer Fusion (bottom).

compress it to a 1D vector. The rest 1D data is processed by MLPs. The action vectors required for the transition model are also obtained through MLPs.

4.2 Sensor Fusion

After feature extraction through the Encoder, we obtain 2D image features $\mathcal{F}_i \in \mathbb{R}^{C\times H_i\times W_i}$ and point cloud features $\mathcal{F}_p \in \mathbb{R}^{C\times H_p\times W_p}$. Some other 1D features are uniformly represented as $\boldsymbol{f}_o \in \mathbb{R}^{D_o}$. We implement three fusion approaches for these features: 1). Simple averaging of features; 2). Concatenating features and followed by a fully connected layer; 3). Fusion using a Transformer [121].

4.2.1 Simple Averaging

Since the shapes of the image and point cloud features are inconsistent, it is impossible to average them directly. Therefore, we first use a convolutional layer to unify their dimensions and then average the feature values on all pixels to obtain a global feature $\boldsymbol{f}_g \in \mathbb{R}^D$, which is a 1D vector. Then we perform an element-wise averaging to global features of two sensors to obtain the fused feature (the top frame of Figure 4.4):

$$z = \frac{\text{avg}(\text{Conv}(\mathcal{F}_i)) + \text{avg}(\text{Conv}(\mathcal{F}_p))}{2} \quad [4.2]$$

4.2.2 Fully Connected Layer

Similar to simple averaging, this method first compresses the 2D image and point cloud features into 1D vectors. Afterwards, the two 1D vectors and other 1D features $\boldsymbol{f}_o$ are concatenated to form a vector of dimension $2D + D_o$. This vector is then passed through a fully connected layer for information interaction to obtain the fused feature with the required dimensions (the bottom frame of Figure 4.4):

$$z = \text{FC}([\text{avg}(\text{Conv}(\mathcal{F}_i)), \text{avg}(\text{Conv}(\mathcal{F}_p)), f_o], W) \quad [4.3]$$

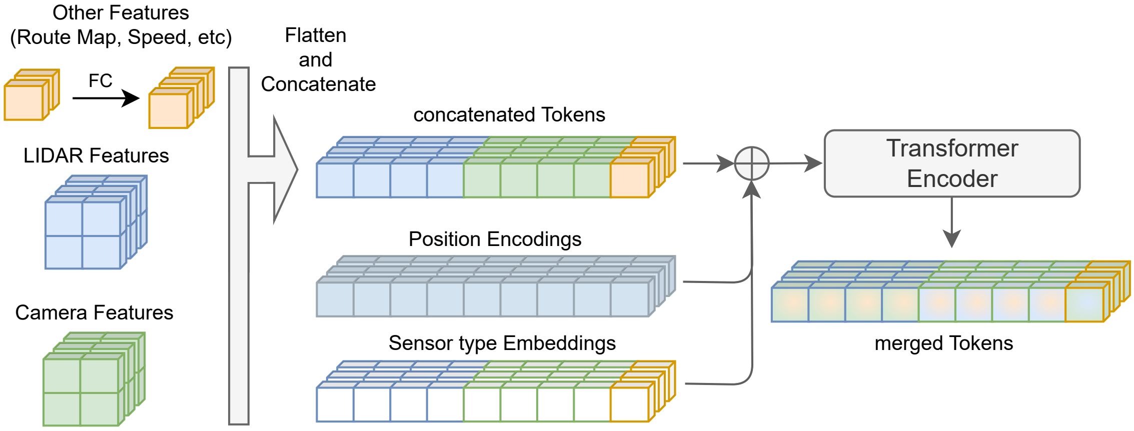


Figure 4.5: The Illustration of Transformer Fusion.

4.2.3 Transformer

The attention module of a Transformer can focus on the correlation between all input tokens. This feature is highly suitable for sensor feature fusion, as it allows features from different sensors to complement each other. Therefore, many recent methods use it for data fusion [141, 20, 98], and we also introduce it into our model. The input required by Transformer is a sequence of tokens $\mathcal{T}_{in} \in \mathbb{R}^{D_t \times N_t}$, where D_t is the dimension of each token and N_t is the number of tokens in the sequence. Since the properties of image features $\mathcal{F}_i$ and point cloud features $\mathcal{F}_p$ is similar, we use a uniform symbol $\mathcal{F} \in \mathbb{R}^{C \times H \times W}$ to illustrate the same operation for both. In order to match the shape of the input to the Transformer, we first use a convolutional layer to change the channel number of feature $\mathcal{F}$ and then flatten it along the H and W dimensions to obtain tokens of shape $D \times HW$. To compensate for the Transformer's insensitivity to position, we add 2D sine and cosine position encoding $\boldsymbol{e} \in \mathbb{R}^{D \times HW}$ [2.6] to each token. Moreover, We incorporate learnable sensor embeddings $\boldsymbol{s} \in \mathbb{R}^{D \times N_s}$ into each token to distinguish the sensor category they belong to, where N_s is the number of sensors. So the resulting token sequence for a single sensor is $\mathcal{T} \in \mathbb{R}^{D \times HW}$, where each token is $t(x, y) = f(x, y) + e(x, y) + s$, and x, y indicate the pixel coordinate of that token within its sensor feature. In addition, for 1D features $\boldsymbol{f}_o$ from other sensors, we use a fully connected layer to match the Transformer's dimension, obtaining $\mathcal{T}_o \in \mathbb{R}^{D \times 1}$. Finally, we concatenate these token sequences from all sensors and pass them into a Transformer encoder, resulting in fused tokens $\mathcal{T}_{fused} \in \mathbb{R}^{D \times (H_i W_i + H_p W_p + N_o)}$, where N_o is the number of 1D features. The illustration is shown in Figure 4.5

4.3 Transition Model

Our transition model follows the design in MILE [48], differing slightly from the Recurrent State Space Model (RSSM) proposed by Hafner et al [40]. In the original RSSM, actions are introduced into the deterministic history via an RNN $\boldsymbol{h}_t = \text{RNN}(\boldsymbol{h}_{t-1}, \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1})$, and the stochastic state $\boldsymbol{s}_t$ is derived from the deterministic state $\boldsymbol{h}_t$ as $\boldsymbol{s}_t \sim p(\boldsymbol{s}_t | \boldsymbol{h}_t, \boldsymbol{z}_t)$ (Section 2.1.2). In contrast, we do not introduce actions $\boldsymbol{a}_t$ into the deterministic history $\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{s}_{t-1})$. Instead, actions

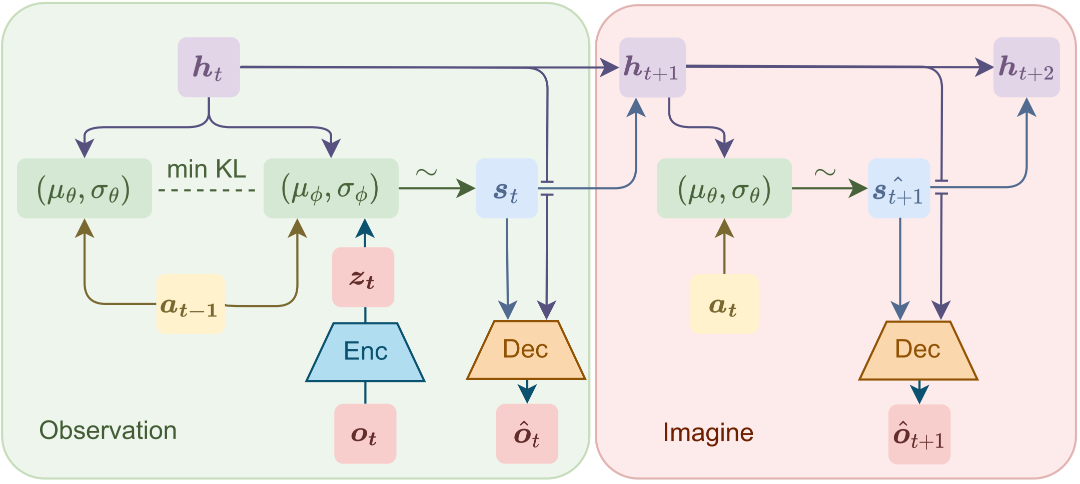


Figure 4.6: The Visualization of Transition Model. (left) With observation. (right) Without observation

are introduced in the subsequent stochastic state model $s_t \sim p(s_t|h_t, z_t, a_{t-1})$. The complete structure is shown in Figure 4.6, with the specific model given by the following equations:

$$\begin{array}{lll}
\text{Initial deterministic state:} & \boldsymbol{h}_0 & \sim \delta(0) \\
\text{Deterministic state model:} & \boldsymbol{h}_t & = f_\alpha(\boldsymbol{h}_{t-1}, \boldsymbol{s}_{t-1}) \\
\text{Initial stochastic state:} & \boldsymbol{s}_0 & \sim \mathcal{N}(0, \boldsymbol{I}) \\
\text{Representation model:} & \boldsymbol{s}_t & \sim \mathcal{N}(\mu_\phi, \sigma_\phi \boldsymbol{I}), \quad (\mu_\phi, \sigma_\phi) = g_\phi(\boldsymbol{h}_t, \boldsymbol{z}_t, \boldsymbol{a}_{t-1}) \\
\text{Transition predictor:} & \hat{\boldsymbol{s}}_t & \sim \mathcal{N}(\mu_\theta, \sigma_\theta \boldsymbol{I}, \quad (\mu_\theta, \sigma_\theta) = g_\theta(\boldsymbol{h}_t, \boldsymbol{a}_{t-1})) \\
\text{reconstruction model:} & \hat{\boldsymbol{o}}_t & \sim p_\alpha(\hat{\boldsymbol{o}}_t|\boldsymbol{h}_t, \boldsymbol{s}_t)
\end{array} \qquad [4.4]$$

Where $\boldsymbol{h}_0$ is the initial deterministic historical state, also served as the hidden state for RNN, initialized by $\delta(0)$, with δ the Dirac delta function. $\boldsymbol{s}_0$ is the initial stochastic state, sampled from the standard normal distribution $\mathcal{N}(0, \boldsymbol{I})$. f_α is the RNN model, we implement it as GRU [21] (Section 2.1.3). The representation model, with parameters ϕ, estimates the posterior distribution of the stochastic state based on the observation features $\boldsymbol{z}_t$, combined with historical information $\boldsymbol{h}_t$ and action $\boldsymbol{a}_t$, as shown on the left side of Figure 4.6. $\boldsymbol{z}_t$ is the fused observation feature obtained through encoding (Section 4.1) and fusion (Section 4.2). The transition predictor, with parameters θ, predicts the prior distribution of the stochastic state based on historical information $\boldsymbol{h}_t$ and action $\boldsymbol{a}_t$, without $\boldsymbol{z}_t$. The right side of Figure 4.6 shows this imagining process. We assume both stochastic state distributions to be multivariate Gaussian, allowing the Kullback-Leibler (KL) divergence to be calculated in closed form. Furthermore, the neural network estimates the mean μ and variance σ of the distribution, and states are sampled with the reparameterization trick so that gradients are able to be backpropagated. Finally, the deterministic historical state $\boldsymbol{h}_t$ and the stochastic state $\boldsymbol{s}_t$ together constitute the current latent state, from which the reconstruction distribution $\hat{\boldsymbol{o}}_t \sim p(\hat{\boldsymbol{o}}_t|\boldsymbol{h}_t, \boldsymbol{s}_t)$ can be decoded. The reconstruction $\hat{\boldsymbol{o}}_t$ includes images, point clouds, and voxels, which we describe in Section 4.4.

4.3.1 1D Latent States Version

In the transition model, the latent states are typically one-dimensional, corresponding to the one-dimensional input observational features [40, 39, 41, 42, 48]. For the fused features obtained through Simple Averaging (Section 4.2.1) or a Fully Connected Layer (Section 4.2.2) we previously mentioned, they have already been compressed into 1D vectors and thus can be directly fed into the transition model for calculation. However, in the case of Transformer Fusion, the obtained fused features are a sequence of tokens, which can't be directly input into the transition model. According to Section 4.2.3, we can get $\mathcal{T}_{\text{fused}} \in \mathbb{R}^{D\times(H_iW_i+H_pW_p+N_o)}$ through Transformer Fusion. We first split this $\mathcal{T}_{\text{fused}}$ into token sequences of lengths H_iW_i, H_pW_p, and N_o according to the shapes of the input sensor features in their input order. Subsequently, we reshape these token sequences back to the original sensor feature shapes, yielding $\mathcal{F}_{i,\text{fused}} \in \mathbb{R}^{D\times H_i\times W_i}$, $\mathcal{F}_{p,\text{fused}} \in \mathbb{R}^{D\times H_p\times W_p}$, and $\mathcal{F}_{o,\text{fused}} \in \mathbb{R}^{D\times N_o}$. Lastly, we employ the same method as the Fully Connected Layer Fusion in Section 4.2.2 to compress and merge all features into a 1D vector.

4.3.2 2D Latent States Adaption

Although we can compress features into 1D vectors, the information contained in these 1D vectors is significantly limited compared to the complex images and point clouds, which is a large challenge for reconstruction tasks. If we simply increase the dimension of the 1D vector, it would lead to an overly complex network, greatly increasing computation and memory consumption. Therefore, we modified the transition model a bit to adapt 2D variables. These modifications involve two parts: the model that generates the stochastic state distribution and the GRU as the deterministic state model. In the modified transition model, we use 2D tensors with a shape similar to the fused observation tokens $\mathcal{T}_{\text{fused}}$ to represent both types of states $\boldsymbol{h}_t, \boldsymbol{s}_t \in \mathbb{R}^{D\times N}$.

Convolutional GRU

In the GRU cell, all neural networks are implemented as fully connected layers. We replace them with convolutional layers suitable for 2D tensors[115, 101]. Since convolution is a local operation with a small receptive field and cannot provide global information, we additionally average all inputs across spatial dimensions to obtain global features[116]. The modified ConvGRU can be represented as follows:

$$\begin{aligned}
u_t &= \sigma(\text{Conv}([\boldsymbol{h}_t, \boldsymbol{s}_t], W_u) + \text{Conv}([\boldsymbol{g}_t], W_u^{global})) \\
r_t &= \sigma(\text{Conv}([\boldsymbol{h}_t, \boldsymbol{s}_t], W_r) + \text{Conv}([\boldsymbol{g}_t], W_r^{global})) \\
\tilde{\boldsymbol{h}}_t &= \tanh(\text{Conv}([r_t \odot \boldsymbol{h}_t, \boldsymbol{s}_t], W_h) + \text{Conv}([\boldsymbol{g}_t], W_h^{global})) \\
\boldsymbol{h}_{t+1} &= (1 - u_t) \odot \boldsymbol{h}_t + u_t \odot \tilde{\boldsymbol{h}}_t \\
\boldsymbol{g}_t &= \text{mean}([\boldsymbol{h}_t, \boldsymbol{s}_t] \odot \sigma(\text{Conv}([\boldsymbol{h}_t, \boldsymbol{x}_t], W_g)))
\end{aligned} \tag{4.5}$$

Where $\boldsymbol{g}_t$ is the global feature combining $\boldsymbol{h}_t$ and $\boldsymbol{s}_t$, controlled by a gate-like operation to regulate its proportion. As $\boldsymbol{h}_t$ and $\boldsymbol{s}_t$ can be considered as a sequence of tokens, we use 1D convolutions

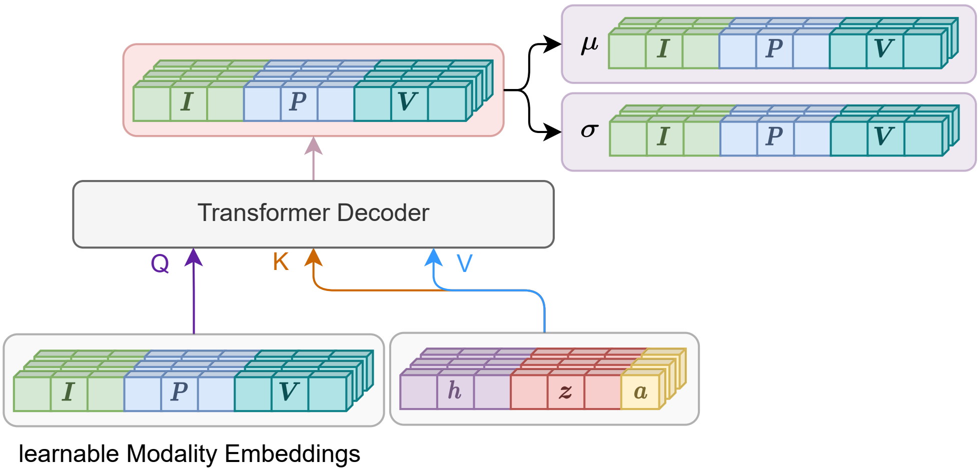


Figure 4.7: Estimate Stochastic State Distribution with Transformer Decoder. Take the example of accepting observation features as input and reconstructing images, point clouds, and voxels.

with a kernel size of 1, which actually performs as an MLP on each token.

Stochastic State Model

The transition model includes two types of stochastic state distributions, with similar models for generating both. The only difference is whether they use observation features as inputs. In the following, we use the posterior distribution, which accepts observations as input, as an example to illustrate.

We use a Transformer Decoder to generate the stochastic state. We combine $\boldsymbol{h}_t$, $\boldsymbol{z}_t$, and $\boldsymbol{a}_t$ into a sequence of tokens to generate key-value pairs, with learnable embedding as the query[98]. The number of query embeddings depends on the desired reconstruction modalities. For example, if we want to reconstruct RGB images, range view images of point clouds, and voxels, we need three types of features: $\mathcal{F}_i \in \mathbb{R}^{D \times H_i \times W_i}$, $\mathcal{F}_p \in \mathbb{R}^{D \times H_p \times W_p}$, and $\mathcal{F}_v \in \mathbb{R}^{D \times X \times Y \times Z}$. Thus, we create corresponding queries $\mathcal{Q}_i \in \mathbb{R}^{D \times H_i W_i}$, $\mathcal{Q}_p \in \mathbb{R}^{D \times H_p W_p}$, and $\mathcal{Q}_v \in \mathbb{R}^{D \times XYZ}$, combined into $\mathcal{Q} \in \mathbb{R}^{D \times (H_i W_i + H_p W_p + XYZ)}$. Since the Transformer Decoder is also insensitive to position, we add sine and cosine position encodings and learnable modality embeddings to these query embeddings, and type embeddings to $(\boldsymbol{h}_t, \boldsymbol{z}_t, \boldsymbol{a}_t)$. Then, in each layer of the Transformer decoder, we can use these queries to retrieve state information in the key-value pairs generated by $(\boldsymbol{h}_t, \boldsymbol{z}_t, \boldsymbol{a}_t)$ through the attention mechanism. After multi-layer information querying, we can obtain the stochastic state tokens $\mathcal{T}_s \in \mathbb{R}^{D \times (H_i W_i + H_p W_p + XYZ)}$, which integrates the information of $(\boldsymbol{h}_t, \boldsymbol{z}_t, \boldsymbol{a}_t)$. We then expand the feature dimension of these tokens via an MLP and then split them into two parts, respectively serving as the mean μ and variance σ of the stochastic state distribution. Since the learnable query embeddings are set according to the modalities of reconstructions, the resulting stochastic states can be divided into states corresponding to different modalities: $\boldsymbol{s_t} = (\mathcal{T}_i^s \in \mathbb{R}^{D \times H_i W_i}, \mathcal{T}_p^s \in \mathbb{R}^{D \times H_p W_p}, \mathcal{T}_v^s \in \mathbb{R}^{D \times XYZ})$. Figure 4.7 is an illustration of this process.

4.4 Decoder and Training Loss

Our model aims not only to reconstruct common images [38, 41, 49] and point clouds [148] but also voxels [10, 69, 150], a data form capable of representing the geometric properties of world space. When dealing with images, the goal is to recreate 3-channel RGB images identical to the inputs. In the case of point clouds, given the uncertain number and order of the original data points, we choose to reconstruct 4-channel pseudo-images projected onto the range view. As for voxels, they are stored in a tensor of size $X \times Y \times Z$, where each voxel stores the category (presence, absence, or type of object) to which it belongs. During training, categories are stored as one-hot vectors, and the reconstruction estimates the probabilities of each grid cell belonging to each category, similar to semantic segmentation in images. Both the reconstructed images and point clouds are 2D data, while voxels belong to 3D data. Next, we introduce two decoders for these two data formats.

4.4.1 2D Decoder for Images and Point Clouds

For a target with shape $H \times W$, we start with a feature map of initial shape $\frac{H}{d} \times \frac{W}{d}$, where d is the downscale factor. Through a series of upscaling layers, the resolution is progressively increased to achieve the target resolution. These upscaling layers involve transposed convolutions with stride 2 for upsampling, followed by an activation layer to maintain the non-linearity of the model [38, 39, 41]. The final layer is customized with a specific head to suit different outputs.

According to Section 4.3, the transition model can generate the latent states $(\boldsymbol{h}_t, \boldsymbol{s}_t)$ in 1D or 2D versions. For the 2D version, since the states can be divided corresponding to each modality of reconstruction, it can be directly reshaped into the feature shape $\frac{H}{d} \times \frac{W}{d}$ required for the reconstruction of that modality.

For the 1D version, the latent states $(\boldsymbol{h}_t, \boldsymbol{s}_t) \in \mathbb{R}^D$ is first reshaped into $D \times 1 \times 1$ and then convolved with kernels of shape $\frac{H}{d} \times \frac{W}{d}$ to produce a feature map with initial shape $\frac{H}{d} \times \frac{W}{d}$ for the decoder.

4.4.2 3D Decoder for Voxels

For the 2D version of the latent states, it can also be reshaped into $\frac{X}{d} \times \frac{Y}{d} \times \frac{Z}{d}$ as the initial feature, and then upsampled to the target shape using 3D transposed convolutions.

For the 1D version of the latent states, we employ an architecture similar to StyleGAN [57]. We use a learnable tensor of shape $\frac{X}{d} \times \frac{Y}{d} \times \frac{Z}{d}$ as the start. This tensor is then gradually upsampled to the target voxel grid size. At each upsampling step, the states $(\boldsymbol{s}_t, \boldsymbol{h}_t)$ undergoes an affine transformation into means and variances, which are then injected into the tensor via adaptive instance normalization[57, 48].

4.4.3 Training Losses

Our world model can be trained without supervision using the standard sensor setup of autonomous vehicles [32, 129], which includes stereo cameras and LiDAR. It does not require expensive manually annotated labels, thereby permitting training on any large datasets without labels. For the target of each modality, we reduce its resolution by factors of 2 and 4, resulting in targets at three distinct scales, including the original one. As for the decoder, it operates by progressively upsampling small-scale feature maps to the final resolution, allowing us to flexibly add output heads at different layers to produce outputs corresponding to targets of various sizes. By employing this multi-scale approach, we can calculate losses at different scales, thereby enhancing the network's ability to learn features at different scales.

Image

We reconstruct RGB images of the same size as the input and use the common L1 loss $\mathcal{L}_{\text{img}}$ to reduce the absolute difference between the target and reconstruction.

Point Cloud

We generate range view images of size $4 \times H_r \times W_r$, where the four channels represent (x, y, z, d) (Section 2.3). We use the range view images converted from the ground truth of point clouds as targets. For points coordinate (x, y, z), we apply an MSE loss $\mathcal{L}_{\text{p, xyz}}$ to minimize the Euclidean distance. For distance r, we use an L1 Loss $L_{\text{p, r}}$ to get the minimum absolute distance difference.

3D Voxel Occupancy

We output Voxel grids of size $192 \times 192 \times 64$, each voxel being 0.5m in size, where each voxel uses binary to indicate if it's occupied. We create targets by voxelizing the fusion of depth maps from the depth camera and point clouds from the LiDAR. The loss function we used for voxels is the Scene-Class Affinity Loss (SCAL) [10] $\mathcal{L}_{\text{v}}$ proposed by Cao et al. This loss function combines Precision, Recall, and Specificity to enable the network to better understand Semantic Scene Completion (SSC).

Stochastic State Distribution

we make predictions by estimating the prior distribution of the stochastic state (Section 4.3). The prior distribution without observations should approximate the posterior distribution based on observations. We employ the KL balancing loss $\mathcal{L}_{\text{KL}}$ introduced in Dreamer V2 [41] as the loss between these two distributions. This loss encourages the prior to more aggressively match the posterior, rather than having both distributions move towards each other symmetrically.

Finally, all losses are combined by:

$$\mathcal{L}_{\text{total}} = \sum_i \lambda_i (\lambda_{\text{img}} \mathcal{L}^i_{\text{img}} + \lambda_{\text{pcd}} (\mathcal{L}^i_{\text{pcd, xyz}} + \mathcal{L}^i_{\text{pcd, r}}) + \lambda_{\text{v}} \mathcal{L}^i_{\text{v}}) + \mathcal{L}_{\text{KL}} \tag{4.6}$$

Where λs are weighting factors for balancing the contributions of different loss components, and i is the downsampling rate. This comprehensive approach, by addressing multiple modalities and scales, aims to robustly train the model to understand and reconstruct the complex nature of real-world environments.

4.4.4 Extra Outputs

Although our model is designed to reconstruct unsupervised data, so that can train on any large fleet dataset, the latent states generated by our transition model (Section 4.3.2) can be flexibly determined by the desired output modality. Additionally, the decoder's heads can be readily altered to accommodate different types of data. Consequently, our model can also be effortlessly adapted to produce other types of outputs with labels.

For instance, our model can output semantic segmentation of images, with cross entropy as the loss. Similarly, the bird's-eye-view semantic segmentation [48, 98, 74] can also be generated, which is an abstract representation of the vehicle environment, again using cross entropy as the loss. Furthermore, the model is also capable of generating policies (accelerate, brake and steering) for imitation learning [48, 20, 151].

5 Experiment Setup

In this chapter, we present how we set up experiments to evaluate our model. Starting with the dataset we employed, we describe how we collected the data, the pre-processing we did on the data and how we established the training and testing sets. Subsequently, we explain what experiments we have performed and the purpose of the experiments. Next comes the parameter settings for training and the metrics for the evaluation. Finally is some implementation details.

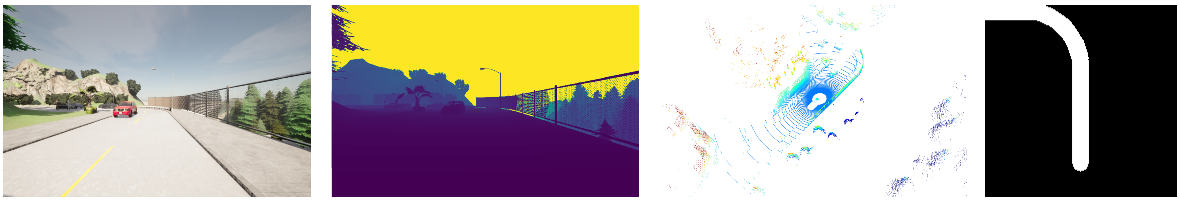
5.1 Dataset

5.1.1 Simulation Environment CARLA

Collecting driving data in the real world is expensive and difficult, so we turned to CARLA [27], a popular driving simulator environment, for data collection. CARLA has build-in a variety of urban, suburban, and rural maps, as well as various traffic participants, providing a detailed driving environment, including streets, buildings, pedestrians, vehicles, etc., so that can simulate the complexity of the real world. Moreover, CARLA supports a wide range of virtual sensors, allowing for the arbitrary placement of necessary sensors and the automatic generation of precise annotations. CARLA offers the user a high degree of freedom to customise the maps, environmental weather, the type and number of pedestrians and vehicles, etc.

5.1.2 Data Collection

We used a pre-trained expert reinforcement learning agent, Roach [151, 48], as the autopilot in CARLA to collect data. Compared to CARLA's built-in autopilot [27], Roach operates more diversely. We followed the typical sensor setup of autonomous vehicles [32, 129]: a front-facing stereo camera and a 360-degree LiDAR (Appendix **??** provides specific settings for the sensors). While the vehicle navigates in CARLA, we collected data at a frequency of 10FPS. For each frame, we saved the following types of data: RGB images $\mathcal{I}_t \in \mathbb{R}^{3\times600\times960}$ (Figure 5.1a) and depth maps



(a) RGB Image (b) Depth Map (c) Point Cloud (d) Route Map

Figure 5.1: Collected data. (a) RGB images and (b) Depth map from the stereo camera; (c) Point cloud from LiDAR, shown in height heat map; (d) Route map shows the planned route in BEV

$\mathcal{D}_t \in \mathbb{R}^{3\times600\times960}$ (Figure 5.1b) from the stereo camera; point clouds $\mathcal{P}_t \in \mathbb{R}^{\leq 60,000\times3}$ (Figure 5.1c) derived from the 64-line LiDAR; route maps $\mathbf{route}_t \in \mathbb{R}^{1\times64\times64}$ (Figure 5.1d) showing the planned route in the BEV perspective; the vehicle's current speed $v_t \in \mathbb{R}$ in m/s; and the actions $\boldsymbol{a}_t \in \mathbb{R}^2$ applied to ego-vehicle, which includes two values - the first for acceleration in $[-1, 1]$, negative for braking, positive for acceleration) and the second for steering in $[-1, 1]$, where negative for left, positive for right).

5.1.3 Data Pre-Processing

The collected data underwent some preprocessing:

Image

The collected images from CARLA have a resolution of 600 × 960. Following MILE's [48] operation, we cropped them to 320 × 832 to discard unimportant areas like the sky. During the training phase, normalization and augmentation operations like blurring, sharpening, and jittering are applied to the input images with a certain probability.

LiDAR Point Cloud

In CARLA, the point cloud is stored in a left-handed coordinate system with the LiDAR as the origin, specifically with forward as the positive x-axis, right as the positive y-axis, and up as the positive z-axis. We converted this into a right-handed coordinate system with the ego-vehicle as the origin, i.e. reversed the direction of the y-axis:

$$\begin{bmatrix} x_{\text{vehicle}} \\ y_{\text{vehicle}} \\ z_{\text{vehicle}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & l_{\text{f}} \\ 0 & -1 & 0 & -l_{\text{r}} \\ 0 & 0 & 1 & l_{\text{u}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{\text{carla}} \\ y_{\text{carla}} \\ z_{\text{carla}} \\ 1 \end{bmatrix} \tag{5.1}$$

where (x, y, z) is the coordinate of the point with the subscript indicating which coordinate system it belongs to. $(l_{\text{f}}, l_{\text{r}}, l_{\text{u}})$ represents the offset of the LiDAR in the front, upper, and right directions relative to the ego-vehicle. When mapping point clouds to range view images, all values are normalised to [-2, 2], and the size of range view image is 64 × 1024.

Voxel Grid with Depth Map and Lidar Point Cloud

CARLA stores depth values by codifying them with RGB channels, and they could be decoded back into actual depth values in meters using the following equation:

$$\mathcal{D} = \frac{(\text{R} + \text{G} \times 256 + \text{B} \times 256^2)}{256^3 - 1} \times 1000 \tag{5.2}$$

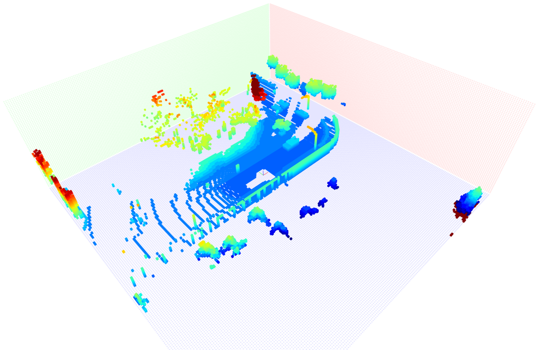


Figure 5.2: The Visualization of Voxel Grid.

The point cloud in camera coordinates was then obtained by converting it with the following formulas:

$$\begin{aligned} x_{\text{camera}} &= \frac{(u - c_x) \cdot z_{\text{camera}}}{f} \\ y_{\text{camera}} &= \frac{(v - c_y) \cdot z_{\text{camera}}}{f} \\ z_{\text{camera}} &= \mathcal{D}(u, v) \\ f &= \frac{W}{2\tan(\frac{\text{FOV} \cdot \pi}{360})} \end{aligned} \tag{5.3}$$

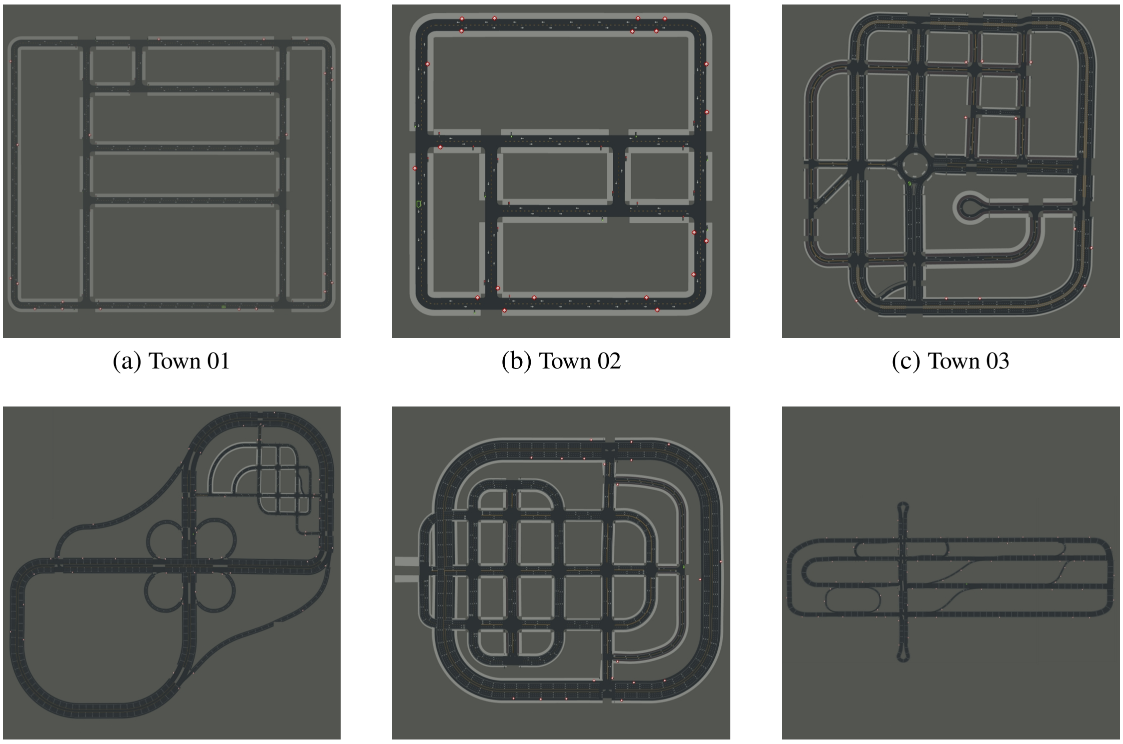
where (u, v) is the pixel coordinate in the depth map, $(x_{\text{camera}}; y_{\text{camera}}, z_{\text{camera}})$ represents this pixel in camera coordinate system; (c_x, c_y) denotes the camera's optical center, which is typically at the center of the depth map, i.e. $(\frac{W}{2}, \frac{H}{2})$; f is the focal length of the camera, calculated from the field of view (FOV), which is represented in angles.

Subsequently, we transformed this point cloud into the ego vehicle's coordinate system via the equation below:

$$\begin{bmatrix} x_{\text{vehicle}} \\ y_{\text{vehicle}} \\ z_{\text{vehicle}} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & c_{\text{f}} \\ -1 & 0 & 0 & -c_{\text{r}} \\ 0 & -1 & 0 & c_{\text{u}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{\text{camera}} \\ y_{\text{camera}} \\ z_{\text{camera}} \\ 1 \end{bmatrix} \tag{5.4}$$

where $(l_{\text{f}}, l_{\text{r}}, l_{\text{u}})$ denotes the offset of the stereo camera in the front, upper, and right directions relative to the ego-vehicle.

Finally, the LiDAR point cloud and depth map point cloud are merged and voxelized. The voxel grid volume is 192 × 192 × 64, with each voxel being a 0.5m cube. The ego vehicle is placed at the center of the xy-plane but offsets downwards by 20 voxels in the z-direction, so the coordinate of the ego vehicle in the voxel grid is (96, 96, 12) and the final voxel grids actually contain objects in the horizontal range (-48m, 48m) and vertical range (-6m, 26m) relative to the vehicle.



(a) Town 01

(b) Town 02

(c) Town 03

(d) Town 04

(e) Town 05

(f) Town 06

Figure 5.3: The Maps of Different Towns in CARLA [27, 13].
(a) Town 01: A small town with many T-intersections, including a river and some commercial and residential buildings.
(b) Town 02: A similar town to Town 01, but simpler.
(c) Town 03: A large city with complex traffic routes such as a roundabout, multi-lane intersections, underpasses, etc.
(d) Town 04: A small town in the mountains with short streets and an '8'-shaped highway winding through the mountains.
(e) Town 05: A square grid city with a bridge, and many crossroads.
(f) Town 06: A town with four long multi-lane highways accompanied by Michigan Lefts [132].

5.1.4 Training Set

The training set is collected across four different towns and four weather conditions. The towns are Town 01, Town 03, Town 04, and Town 06 (as described in Figure 5.3). The four weather conditions include Clear Noon, Wet Noon, Hard Rain Noon, and Clear Sunset. The autopilot operated unplanned in each town, resulting in random driving routes. We conducted 25 runs in each town, each lasting 300 seconds with randomly selected weather conditions. Therefore, in total, we conducted 100 runs across the four towns, equivalent to 8.3 hours, collecting 300,000 frames of data.

5.1.5 Validation Set

To comprehensively validate our model performance, we established three datasets: Diverse Town Diverse Weather, Same Town Diverse Weather, and Same Town Same Weather.

$\mathcal{D}^{\text{DTDW}}$: Diverse Town Diverse Weather

In this set, we introduced two new towns (Town 02 and Town 05 as shown in Figure 5.3) and four new weather conditions (Soft Rain Sunset, Wet Sunset, Cloudy Noon, Mid Rain Sunset). Thus, this validation set represents a completely different scenario from the training set. The introduction of this dataset aims to evaluate our model's ability to perform with a complete domain shift.

$\mathcal{D}^{\text{STDW}}$: Same Town Diverse Weather

This dataset maintains the same towns as in the training set but uses different weather conditions (Soft Rain Sunset, Wet Sunset, Cloudy Noon, Mid Rain Sunset). The autopilot in this set follows a planned path predefined by CARLA AD Leaderboard [14]. The main objective of this dataset is to assess our model's performance in the face of minor environmental changes, evaluating its generalization capabilities and robustness [7].

$\mathcal{D}^{\text{STSW}}$: Same Town Same Weather

This set has the same towns and weather conditions as the training set. The driving route is also predetermined. The purpose is to evaluate the efficacy of representation learning (RL) of our model in familiar situations [7].

5.2 Training Parameters

We sampled data in the training set at 0.2s seconds intervals to create training sequences. For experiments without voxel reconstruction, we used sequences of length 12 frames as training input. For experiments that included voxel reconstruction, we reduced the sequence length to 6 due to speed and memory constraints. Each training session was performed on a 40G A100 GPU with 50,000-100,000 iterations and a batch size of 16. We optimised using the AdamW optimiser [75] with a learning rate of 10^{-4} and a weight decay of 0.01.

For training, we treated all input sequences as known data. For validation, for experiments without voxel reconstruction, the first 6 frames are the receptive field, which serves as given observations feeding into the network, and the last 6 frames are the future horizon, which serves as a ground truth for the prediction reconstruction. For voxel reconstruction, a combination of 4 frames of receptive field plus 2 frames of future horizon was used. Receptive field frames were used to evaluate the reconstruction and future horizon frames were used to evaluate the prediction.

5.3 Metrics

For evaluating the performance of our network in various reconstruction tasks, we employed specific metrics for each modality:

Camera

We used the Peak Signal-to-Noise Ratio (PSNR) as the metric for images. PSNR is an established measure in image processing that indicates the fidelity of the reconstructed image compared to the original. A higher PSNR value generally signifies better reconstruction quality, reflecting the completeness of the information retained in the reconstructed image.

LiDAR

The Chamfer Distance (CD), which is lower is better, was chosen as the metric for point cloud reconstruction. The Chamfer Distance is a common metric measuring the average nearest distance between points in the reconstructed point cloud and the corresponding points in the ground truth. It effectively quantifies the accuracy of the spatial positions of the points in the reconstructed cloud.

3D Voxel Occupancy

For the voxel occupancy reconstruction task, we used Intersection over Union (IoU), Precision, and Recall as the metrics. These metrics are widely used in classification and segmentation tasks. IoU measures the overlap between the predicted voxel occupancy and the ground truth, providing a sense of overall accuracy. Precision reflects the proportion of correctly predicted occupied voxels in relation to all voxels predicted as occupied. Recall indicates the proportion of actual occupied voxels that were correctly identified by the model. All of them are higher is better.

Table 5.1: The Abbreviations for different Methods

Module	Abbreviation	Description
Image Encoder	**WOB**	**W**ith **O**ut **B**EV: Section 4.1.1 Only extracting image features by ResNet
	WBM	**W**ith **B**EV **M**apping: Section 4.1.1 Converting image features to BEV space
Point Cloud Encoder	**RV**	**R**ange**V**iew: Section 4.1.2 Mapping point clouds into range view images
	PP	**P**oint**P**illars: Section 4.1.2 Using PointPillars for point clouds feature extraction
Fusion Method	**AVG**	**AV**era**G**ing: Section 4.2.1 Fusion with simple averaging
	FCL	**F**ully **C**onnected **L**ayer: Section 4.2.2 Fusion with fully connected layer
	TFE	**T**rans**F**ormer **E**ncoder: Section 4.2.3 fusion with Transformer encoder

Table 5.2: The Combinations of Different Methods. The left are fusion methods and the upper are encoders

	WOB + RV	WOB + PP	WBM + RV	WBM + PP
AVG			×	×
FCL	×	×	×	×
TFE	×	×	×	×

5.4 Experiments

As described in Chapter 4, we implemented a variety of methods across the various modules of our model and reconstructed 3D voxel occupancy in addition to the images and point clouds used as inputs. We therefore wished to test the impact of these different schemes on the performance of the model. We have set up the following experiments.

5.4.1 Different Combinations with 1D latent states

In this experiment, we evaluate the impact of different encoders for images and point clouds, as well as various fusion methods, on reconstruction and prediction performance. We only tested the case when the transition model was used in the 1d version and the reconstruction of voxels was not performed. In Table 5.1 we defined some abbreviations to denote the different methods in order to easier description later, and Table 5.2 shows the combinations we have tested.

5.4.2 3D Voxel Occupancy

For 3D voxel occupancy reconstruction, we would like to know whether pre-trained weights trained only on camera and LiDAR data can aid voxel reconstruction training. Conversely, we are also interested in whether voxel reconstruction can influence the quality of reconstruction of camera and LiDAR data.

Based on our previous experimental results, we chose the combination of WOB+RV+TFE for the experiment of voxel reconstruction. We trained this model for 50,000 steps only with image and point cloud reconstruction to obtain the pre-trained weights, and then conducted the following experiments:

Pre-Trained weights Frozen (PTF)

we used the pre-trained weights for voxel reconstruction training. But we froze the network parts where these weights are applied. This means the network parts with pre-trained weights would not be trained, only the weights of the voxel decoder could be updated. This setting enables us to evaluate the effect of fine-tuning only the voxel-specific elements of the model while maintaining the remaining network unchanged. In other words, all encoders and the transition model were constant in this training, we can determine if the model has already learned any information about the geometric features of the world solely from camera and LiDAR data.

Pre-Trained weights Open (PTO)

we started training also with pre-trained weights but allowed all parts of the network, including the pre-trained sections and voxel decoder to update their weights. Through this experiment, we can comprehend the impact of pre-trained weights on the learning process when the entire network adjusts and develops during training.

WithOut Pre-trained weights (WOP)

we didn't use any pre-trained weights, instead, trained the entire network from scratch. This provides a baseline to understand the impact of using pre-trained weights versus training entirely from the beginning.

5.4.3 2D Latent State with Voxel

As we adapted our model to 2d latent state, we are interested to compare the performance difference between it and the 1d latent state. Because of time constraints, we selected only the WOB+RV+TFE combination, which performed well in the 1d version, to experiment the 2d version. We compared it with the 1d versions in terms of its ability to reconstruct and predict the image, point cloud and voxels.

5.5 Implementation Details

We used pre-trained 18-layer ResNet from PyTorch Image Models (timm) [131] as Backbone to extract features. While mapping the image features to BEV, 37 discrete depths were assigned and the size of the BEV feature map was 48×48 with 0.8m per pixel. When using PointPillars, it created pillars containing 5 pixels per meter in the horizontal range [-48m, 48m], i.e. 480×480 pillars. The output feature dimension of PointPillars is 32. The Transformer encoder and decoder were implemented with 8 heads, 6 layers and 384 feature dimensions. More detailed network parameters are placed in Appendix A.3.

6 Evaluation

In the subsequent sections, we evaluate the experiments that we presented in Chapter 5. Through these experiments, we analysed the advantages and disadvantages of various encoders and fusion methods in the hope of identifying the optimal model architecture. We also explore the impact of pre-training on voxel reconstruction and the voxel reconstruction task on other aspects of the model's performance. We then compared the use of the 2d latent state with the 1d version. Finally, we analysed some additional factors and the dataset we used.

6.1 Combinations with 1D latent states

The first is a comparison of different combinations of encoder and fusion methods with 1d states. We evaluate different model architectures by comparing the ability of the models to reconstruct and imagine the two observed inputs under different environmental conditions.

6.1.1 Image Reconstruction and Prediction

Figure 6.1 demonstrates the quality of the reconstructed and predicted images for the different architectures of the model for the 3 environmental conditions. It is clear that as the similarity between the test and training environments decreased, the performance of all models underwent a different degree of degradation. Among them, in the dataset with different weather in completely different towns, the models showed significant overfitting. Whereas for shifts in weather conditions only, the models can do some level of migration. It can also be noticed that different models showed similar gaps in prediction and reconstruction.

For the image encoders, using only ResNet for feature extraction (WOB) is significantly better than performing an additional BEV mapping (WBM). Obviously, some information will be lost in the process of mapping image features into BEV space, which will affect the reconstruction performance. For the point cloud encoders, range view methods (RV) overall outperformed PointPillars (PP), but PP combined with WOB showed a little advantage in reconstruction under weather shifts only. For the feature fusion approaches, the Transformer-based fusion method (TFE) showed an advantage in the representation learning of the exact same environment ($\mathcal{D}^{\mathrm{STSW}}$), where the WOB_RV_TFE combination performed the best. However, in migration learning with changing environments ($\mathcal{D}^{\mathrm{STDW}}$ and $\mathcal{D}^{\mathrm{DTDW}}$), the overfitting tendency of the Transformer was more severe, and the fully connected layer fusion (FCL) performed better. This may be due to the more complex structure and the larger number of parameters of the Transformer. Whereas simple averaging (AVG) performed the worst in all situations.

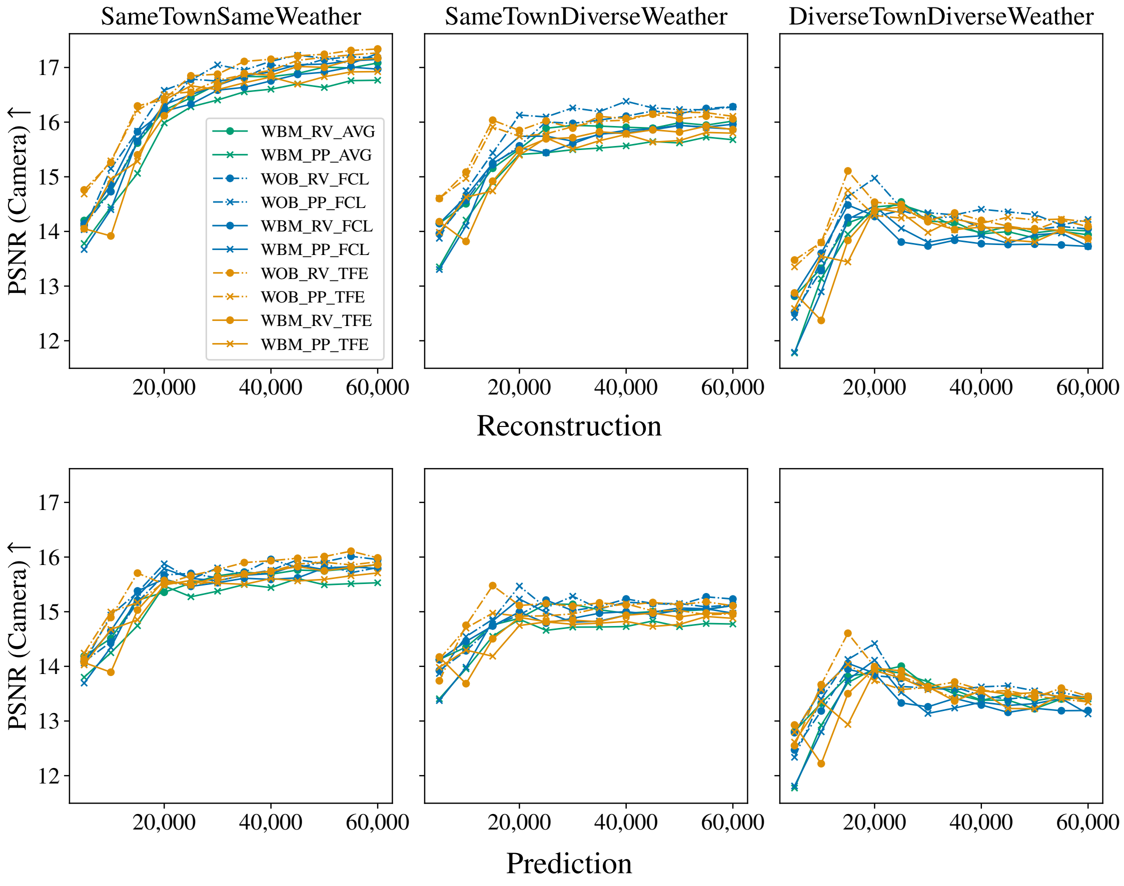


Figure 6.1: Evaluation of different model architectures with the quality of reconstructed and predicted images in 3 validation sets. The validation set is introduced in Section 5.1.5. The upper texts identify the validation set for each column. The first row is reconstruction and the second row is prediction. PSNR is used as a metric of image quality, which higher means better. The combinations of methods and their abbreviations are presented in Section 6.1, Tables 5.1 and 5.2. The line styles indicate the image encoders, the markers indicate the point cloud encoders, and the colors indicate the fusion methods.

Overall, for the tasks of image reconstruction and prediction, image encoders without BEV mapping (WOB) outperform BEV mapping (WBM). Using the range view (RV) to process point clouds is overall better than PointPillars (PP). Transformer fusion (TFE) and fully connected layer fusion (FCL) each have their own advantages and disadvantages.

6.1.2 Point Cloud Reconstruction and Prediction

Examination of Figure 6.2 reveals that in the point cloud reconstruction and prediction, no overfitting was exhibited during weather changes ($\mathcal{D}^{\text{STDW}}$), while the performance dropped significantly during town changes ($\mathcal{D}^{\text{DTDW}}$). This is understandable because the laser reflections of the point cloud do not react much to weather changes compared to the images where the weather can significantly affect its color space. In contrast, the generated point clouds are completely different in different cities.

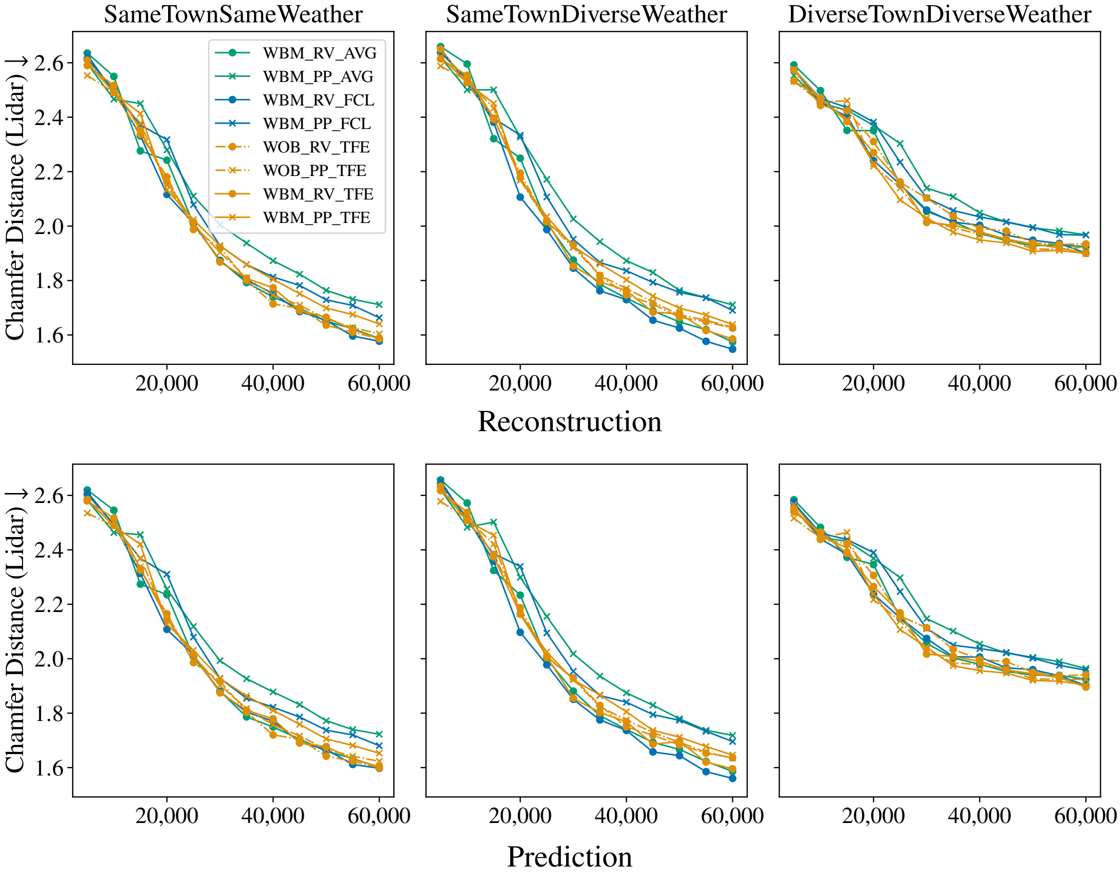


Figure 6.2: Evaluation of different model architectures with the reconstruction of point cloud in 3 validation sets. The Chamfer Distance is used for comparison, which lower indicates better.

For image encoders, BEV mapping (WBM) no longer showed a significant disadvantage, and even occasionally led. It shows that mapping image features to BEV space somewhat affects the model's understanding of spatial features. For the point cloud encoder, PointPillars (PP) was again at a significant disadvantage. This is probably because reconstructing the point cloud is done by reconstructing the range view image of the point cloud. For feature fusion, it is similar to the image task. Simple averaging fusion (AVG) remained at the bottom of the list, while the Transformer (TFE), compared to the fully connected layer (FCL), once again leads in the validation set of the same towns with the same weathers ($\mathcal{D}^{\mathrm{STSW}}$) and lags behind in the validation set of the same towns with different weathers ($\mathcal{D}^{\mathrm{STDW}}$).

In general, point cloud reconstruction and prediction did not perform much differently from image reconstruction and prediction. However, the point cloud prediction and reconstruction are less sensitive to weather shifts, and the Transformer shows more severely overfitting cases in both.

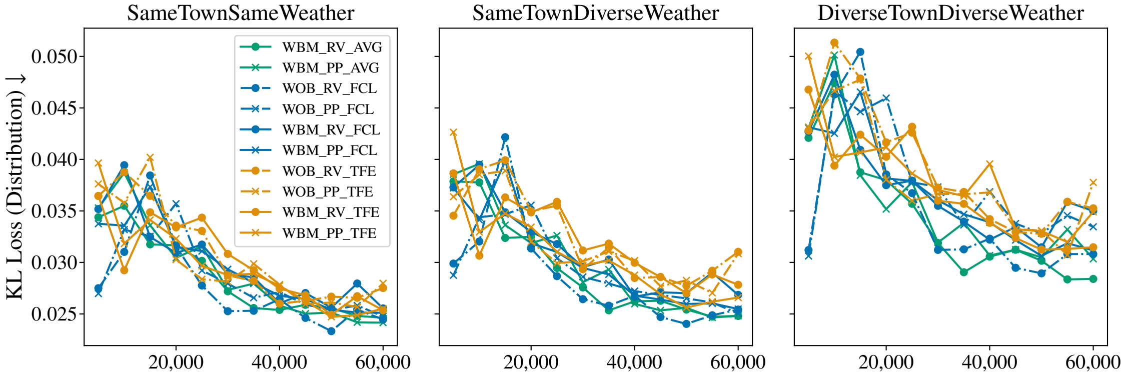


Figure 6.3: Evaluation of different model architectures with the KL balancing loss between the posterior and prior distributions of stochastic states.

6.1.3 Distribution of Stochastic States

As can be seen in Figure 6.3, the models with the different architectures performed similarly for the validation set with the same towns and the same weathers ($\mathcal{D}^{\text{STSW}}$). This suggests that the proximity of the prior to the posterior distribution of the stochastic state does not depend on the way the model acquires features. And under the remaining two different levels of domain shift ($\mathcal{D}^{\text{STDW}}$ and $\mathcal{D}^{\text{DTDW}}$), the Transformer again showed overfitting.

Based on the above analyses of the plots for each evaluation metric, we can conclude that the combination of directly using ResNet to extract image features without BEV mapping (WOB), projecting the point cloud to range view followed by feature extraction (RV), and fusing the features using Transformer (TFE) gives the best comprehensive performance for reconstruction and prediction of both observation inputs in the representation learning situation ($\mathcal{D}^{\text{STSW}}$).

6.2 with 3D Voxel Occupancy Reconstruction

Due to severe overfitting on the Same Town Same Weather validation set, we no longer evaluate the experiments with voxel reconstruction on this validation set. As discussed above, the model with the WOB_RV_TFE combination has the best performance, so for the experiments on voxels, we use the model with this architecture.

6.2.1 Impact on 3D Voxel Occupancy

As we observe Figure 6.4, we can see that the model trained without using pre-trained weights (WOP) has close performance in both $\mathcal{D}^{\text{STSW}}$ and $\mathcal{D}^{\text{STDW}}$ for all three metrics, indicating that it can maintain its performance even when migrating to weather-different environments. Whereas the other two models trained with pre-trained weights (PTF and PTO) both performed better in $\mathcal{D}^{\text{STSW}}$ than in $\mathcal{D}^{\text{STDW}}$.

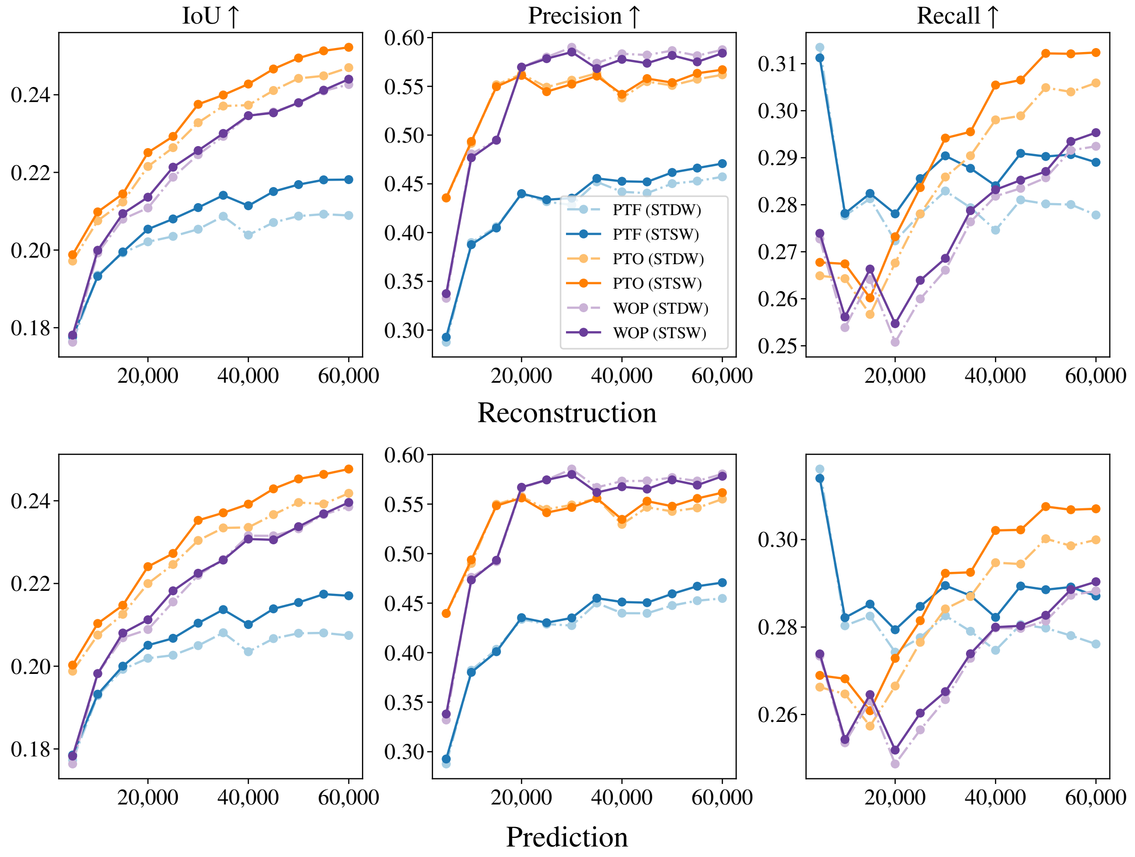


Figure 6.4: Evaluation of the impact of 50,000 steps pre-training with images and point clouds on 3D voxel occupancy reconstruction and prediction. We evaluated on $\mathcal{D}^{\text{STSW}}$ and $\mathcal{D}^{\text{STDW}}$.The purple line shows the baseline trained from the sketch, the orange line indicates that pre-trained weights were used and the weights were kept open for update, and the blue line indicates that the weights were frozen.

Comparing the metrics of using pre-trained weights and open training (PTO) and trained from scratch (WOP), the PTO showed an advantage in the early stage of training, which indicates that the pre-trained weights already contain some of the spatial structured features, which can be helpful to the training of voxel reconstruction.

However, in the later stages of training, in contrast to the PTO model in IoU and Recall, which kept the lead, the WOP model accomplished a reversal in Precision. This indicates that the WOP model performs more conservatively compared to the boldness of the PTO model for voxel occupancy grid prediction.

Finally, looking at the pre-trained weights with frozen (PTF), although it is lower than the other two in all the metrics, its performance rises after a period of training with only the Voxel Decoder, which again validates the fact that the network partially fuses the image and point cloud features to form the spatial voxel features when trained with only them.

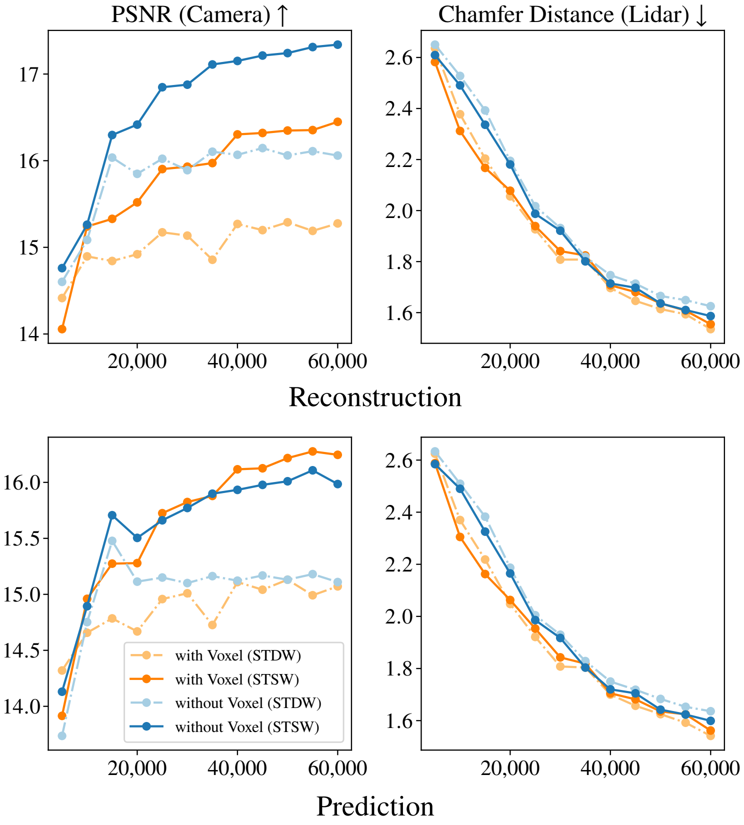


Figure 6.5: Evaluation of the impact of 3D voxel occupancy reconstruction on the quality of the image and point cloud reconstruction and prediction. We evaluated on $\mathcal{D}^{\text{STSW}}$ and $\mathcal{D}^{\text{STDW}}$. The orange line represents training with voxel reconstruction while the blue line indicates without.

6.2.2 Impact on Image and Point Cloud

From Figure 6.5, it can be noticed that for the image reconstruction task, the introduction of the voxel reconstruction hindered the reconstruction of the image. In contrast, for the rest of the tasks, i.e. the image prediction and the point cloud reconstruction and prediction, there was a small improvement in the performance of the model after the inclusion of voxel reconstruction. Clearly, the voxel reconstruction can affect the reconstruction and prediction of images and point clouds. And whether the impact is good or bad depends on the specific task.

6.3 2D Latent State

It is clear from Figure 6.6 that the model with 2D latent states has a crushing advantage over the 1D version in all missions. And on the Diverse Town Diverse Weather validation set $\mathcal{D}^{\text{DTDW}}$, where the 1D version model commonly encounters overfitting problems, the 2D version's performance is only reduced compared to itself but no overfitting and is even on par with the performance of the 1D version model in $\mathcal{D}^{\text{STSW}}$ and $\mathcal{D}^{\text{STDW}}$. This indicates that the 2D version model has very good generalisation and robustness.

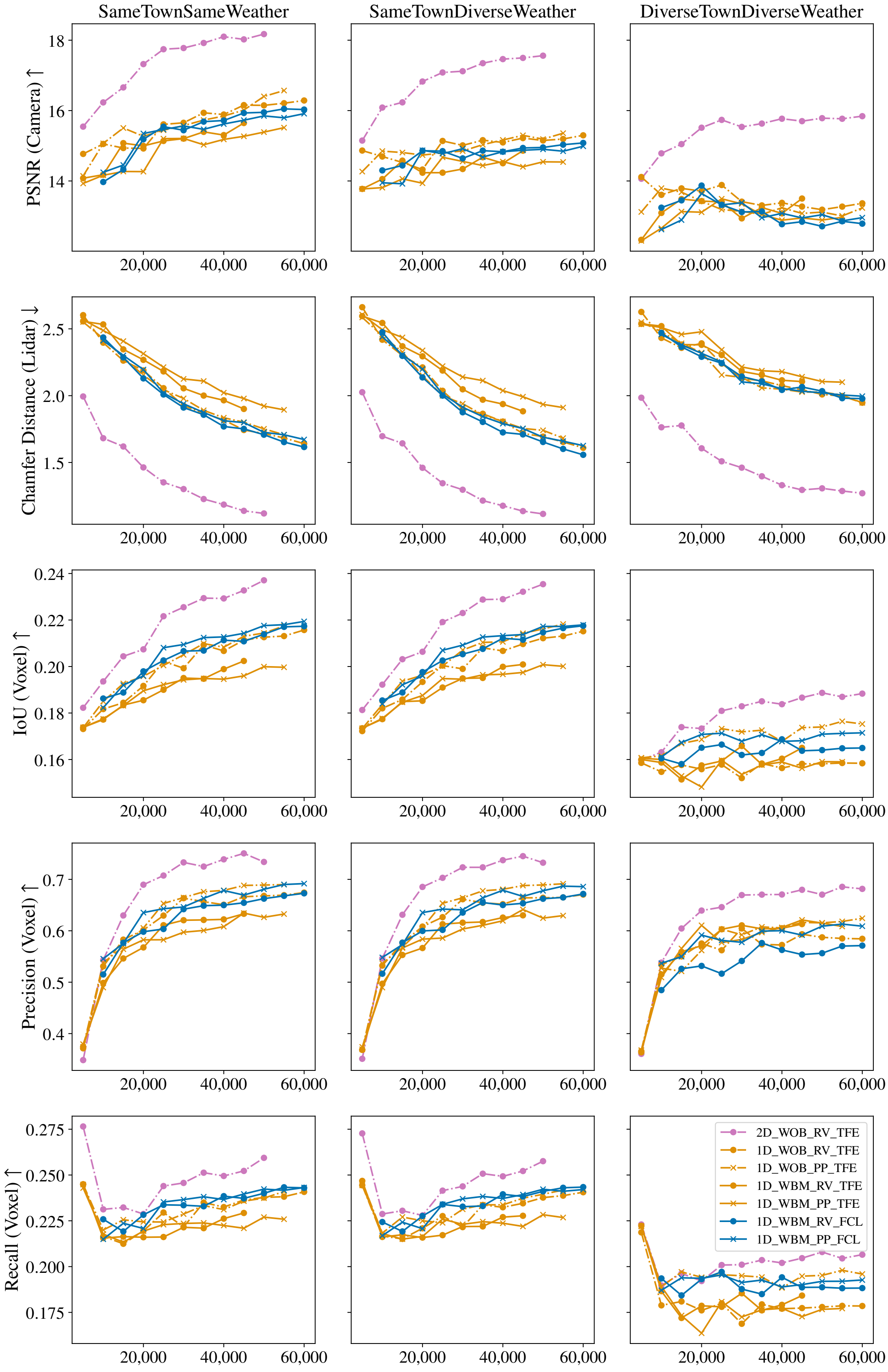


Figure 6.6: Evaluation of the model with 2D latent states compared to 1D.

Although the main purpose of this experiment was to evaluate the performance of the 2D latent state model, we also found an interesting phenomenon in it. PointPillars (PP) shows a very good ability in voxel reconstruction compared to its terrible performance in image and point cloud reconstruction. It suggests that the strength of PointPillars lies in the extraction of spatial geometric features.

6.4 Extra Experiments

We also performed some additional experiments to further understand our model. The following experiments were all conducted with the 2d hidden state version of the model that had previously performed the best.

6.4.1 Near-Field

Point clouds captured from LiDAR are characterised by being dense near and sparse far away. We therefore want to evaluate the accuracy of our model in reconstructing the point cloud in the near-field. The range of our original point cloud is 100m. We selected the horizontal range [-20m, 20m] and vertical range [-2m, 6m] as the near-field, and calculated the chamfer distance between the reconstructed point cloud and the ground truth in this field. For the voxel reconstruction, we also calculated the metrics within the near-field. For the original 192 × 192 × 64 voxels with ego vehicle centre coordinates of 96 × 96 × 12 (Section 5.1.3), we selected voxels within the voxel coordinate range [48, 144], [48, 144], [8, 24] for the calculation. This corresponds to [-24m, 24m] horizontally and [-2m, 6m] vertically around the ego vehicle. The results in Table 6.1 showed that the reconstruction accuracy of the point cloud and voxels is much better in the near-field range than in the far-field.

Table 6.1: The Performance of Point Cloud and Voxel Reconstruction in Nearfield. Testing reconstruction task under the validation set $\mathcal{D}^{\text{STSW}}$ via 2D latent state model.

	CD ↓	IoU ↑	Precision ↑	Recall ↑
original	0.989	0.247	0.767	0.267
nearfield	**0.379**	**0.491**	**0.810**	**0.555**

6.4.2 Failure with CrossEntropy Loss for Voxel Occupancy

For the voxel reconstruction loss, we also tried to use cross entropy, a very common loss in classification tasks. We found that using it in voxel reconstruction leads to training failure very easily. It may be because the gap between positive and negative samples in the voxel grid is too large. In voxels, most of them are empty, i.e., negative samples.

6.4.3 BEV Reconstruction

Although our model focuses on the reconstruction of observed sensor data and 3D voxel occupancy, we also adapt our model to output other modalities like BEV as described in Section 4.4.4.

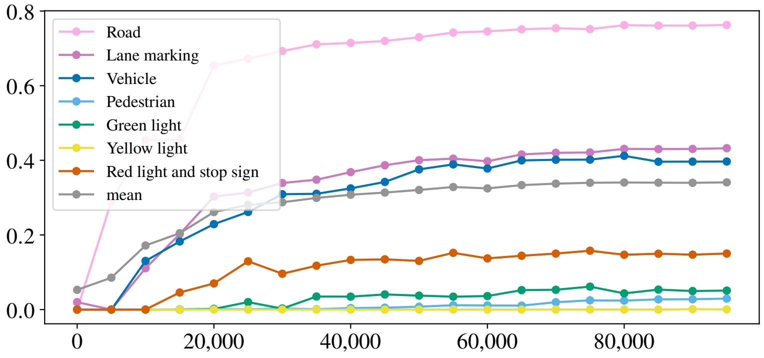


Figure 6.7: The IoU for all categories in BEV reconstruction on the same town same weather validation set.

Therefore, we tested the performance of the 2D version model in reconstructing the BEV image. The BEV image is given in the form of semantic segmentation (as shown in Figure 6.9) and contains seven categories: road, lane marking, vehicle, pedestrian, green light, yellow light, red light and stop sign [151, 48]. Figure 6.7 shows the growth of IoU for these categories with an increasing number of training steps on the validation set $\mathcal{D}^{\text{STSW}}$. The quality of the reconstruction is very excellent for the road, which occupies a large area, and also good for large or stable objects like vehicles and lane marking. However, it does not perform well for pedestrians and traffic signals, which are small dynamic targets. As can be seen in Figure 6.9 the model actually recognises and reconstructs these small targets, but their shape and position cannot be accurately reconstructed, resulting in small IoU values.

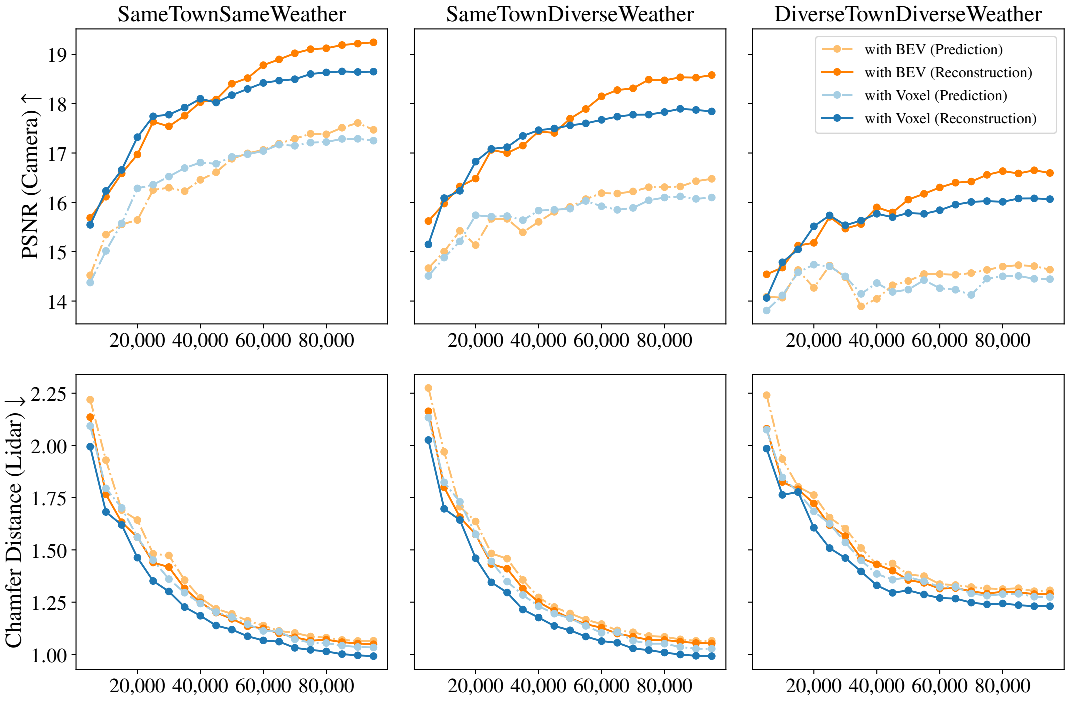
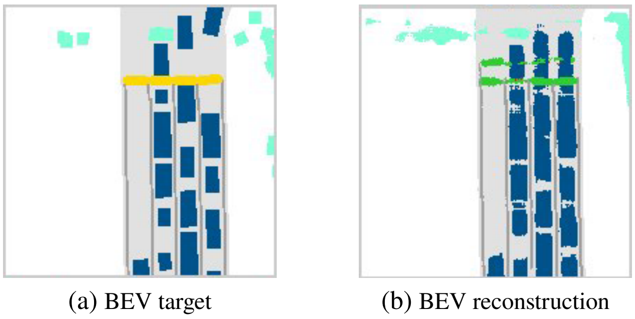


Figure 6.8: Comparison of the impact of the BEV reconstruction and Voxel reconstruction.



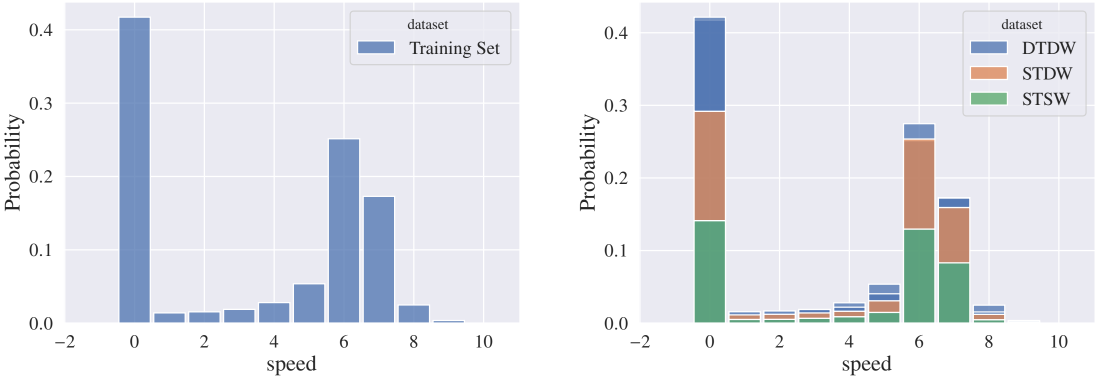
(a) BEV target (b) BEV reconstruction

Figure 6.9: The Example of BEV Semantic Segmentation.

We also compared the quality of the reconstruction of images and point clouds when performing BEV reconstruction with that when performing voxel reconstruction in the validation set $\mathcal{D}^{\text{STSW}}$. The results are shown in Figure 6.8. BEV reconstruction enhanced image reconstruction but was not as helpful as voxel reconstruction in point cloud reconstruction. This is intuitive that BEV provides semantic information and requires the model to enhance the learning of object features, whereas voxels motivate the model to favour the learning of spatial geometric features.

6.4.4 Dataset Evaluation

The quality of the dataset also plays a very important role in the training of neural networks. In our experiments, we found that vehicles often stop in place and the model reconstructs a far better image when the vehicle is stationary than when the vehicle is moving. Therefore, we made statistics on the speed of the ego vehicle in our collected dataset so that we can collect data and train the model better in the future. Figure 6.10 shows the distribution of vehicle speeds on the training and validation sets. We can find that the percentage of speeds in 0m/s reaches more than 40%. Such a large percentage of the stationary state is what we do not want.



(a) Training Set (b) Validation Set

Figure 6.10: The speed distribution in Dataset.

Table 6.2: The Performance under Different Ego Vehicle Speeds

	PSNR ↑	CD ↓	IoU ↑	Precision ↑	Recall ↑
total	18.572	0.989	0.247	0.767	0.267
< 4m/s	**19.817**	1.010	**0.252**	**0.796**	**0.269**
> 4m/s	16.881	**0.957**	0.240	0.725	0.264

We also counted the difference in performance exhibited by the model for vehicle speeds less than 4m/s and greater than 4m/s. Table 6.2 presents the statistics, using the best 2D version model from previous experiments for reconstruction tests under the validation set $\mathcal{D}^{\mathrm{STSW}}$. The performance is significantly better when the ego vehicle is near a standstill.

7 Conclusion and Outlook

In the final chapter, we summarize our work. First, We reiterate our contributions, review the results, present our conclusions, and then analyse limitations. Subsequently, we give the direction for improvement and the possible future works.

7.1 Conclusion

There are two objectives for us: One is to adapt traditional world models performing at low resolution to accommodate high-resolution camera sensors and integrate data from LiDAR sensors, allowing the model to extract environmental information from both modalities and enhance its understanding of the surroundings. Another is, that we aimed to generate geometric 3D voxel occupancy to aid the model in its representation of spatially structured information.

To achieve these goals, we began with the encoders, implementing two types for image inputs: direct use of ResNet and additional BEV mapping; and two types for point clouds: range view mapping and PointPillar feature extraction. We examined the impact of different encoder combinations on performance. For feature fusion, we designed three different schemes: simple averaging fusion, fully connected layer fusion, and Transformer-based fusion. To adapt to these fusion schemes and enhance the latent state's representation ability of the environment, we improved the transition model with a transformer decoder to use 2D latent states.

Our experiments revealed that the absence of BEV mapping and the use of range view best preserved the original information. Feature mapping to BEV and PointPillars could not retain the original information effectively. For sensor feature fusion, the Transformer was effective, but its performance dropped more clearly when shifted to unfamiliar environments. Switching to 2D latent states significantly improved performance. Higher-dimensional states could clearly learn more environmental features and transfer well when the environment changes, indicating that a larger state search space is beneficial for transfer learning. Additionally, the incorporation of voxel reconstruction indeed enhanced the model's learning of spatial information. The voxel experiments demonstrated PointPillars' capability to extract spatial information. Although it lost some metadata information, it captured more spatial information. Further experiments comparing with BEV reconstruction showed that adding voxel reconstruction indeed enhanced the model's understanding of spatial geometry, but at the cost of losing some image color information.

Finally, our analysis of the dataset revealed that the model performed better in static and near-field spaces.

Overall, we successfully achieved our goals of sensor fusion and reconstructing 3D voxel occupancy. Our improved model with a 2D latent state space significantly enhanced the model's representational capabilities.

7.2 Outlook

Firstly, it is regrettable that due to time constraints, our experiments were not as comprehensive as desired. All our comparisons regarding sensor encoders and fusion methods were conducted under the condition of 1D latent states without voxel reconstruction. In future research, we can continue to experiment with the performance of encoders and fusion methods under 2D latent states and with voxel reconstruction. This will give us a more comprehensive comparison of them and gain a better understanding of the strengths and weaknesses of these encoders and fusion methods as well as their impact on different aspects.

Secondly, our analysis of the dataset also revealed serious shortcomings in our dataset. Subsequently, we need to further improve the collection of our dataset, such as trying to replace it with a better-performing autopilot, or collecting a larger dataset and then randomly discarding some samples with lower speeds to achieve speed balance. In addition, we can also try training on real-world data.

Observing the reconstruction results (Figure 7.2), the model has poor reconstruction ability for dynamic objects and experiences issues with blurring, incorrect reconstructions, and viewpoint mismatch. In future work, the latent state space and decoders need to be further strengthened. We can try swapping in the recently impressive diffusion models in image reconstruction to improve reconstruction quality.

In addressing these issues, we should aim to refine our model's understanding of dynamic and complex environments, enhance the quality of reconstructions, and ultimately push closer to more realistic and effective world models. These improvements will not only benefit the theoretical understanding of these systems but also their practical applicability in real-world scenarios.

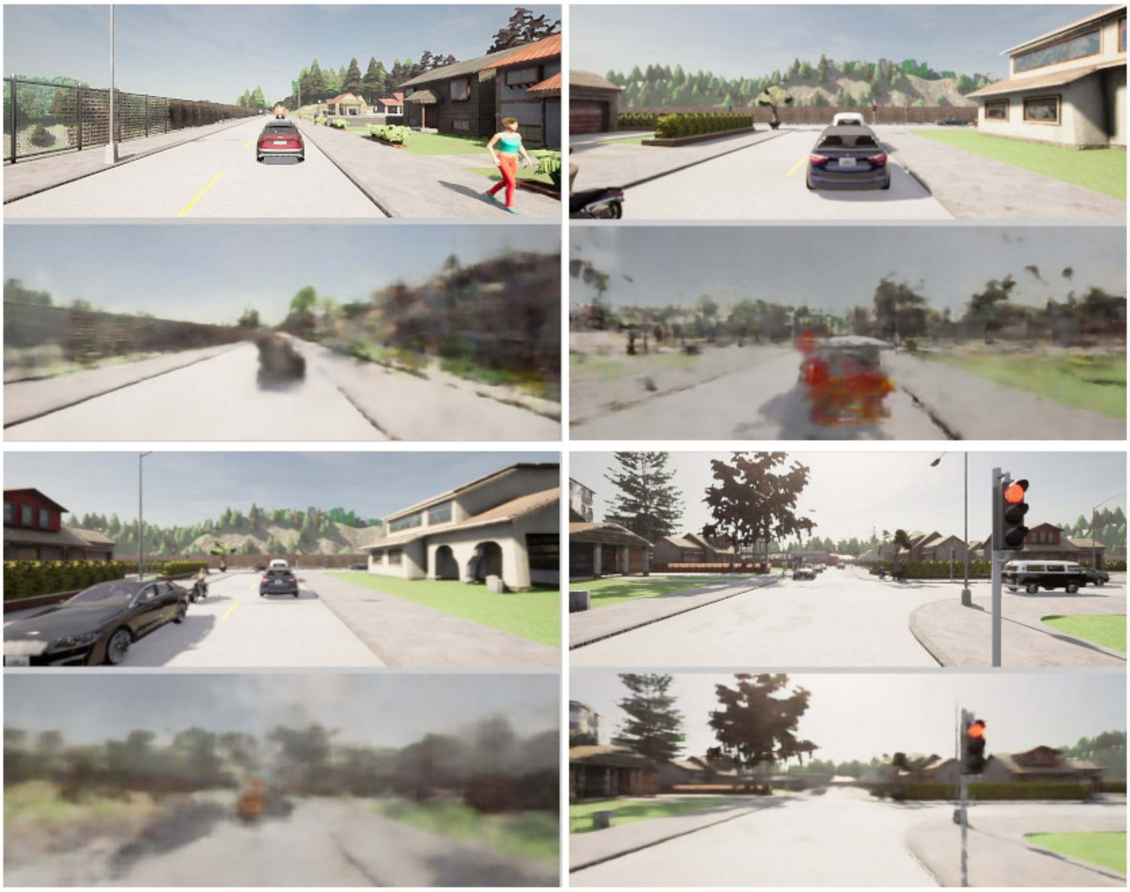


Figure 7.1: Some Reconstruction Failures. From left to right, from top to bottom: 1) pedestrian missing; 2) incorrect object; 3) blurring; 4) viewpoint mismatch. Reprinted from our previous paper [7].

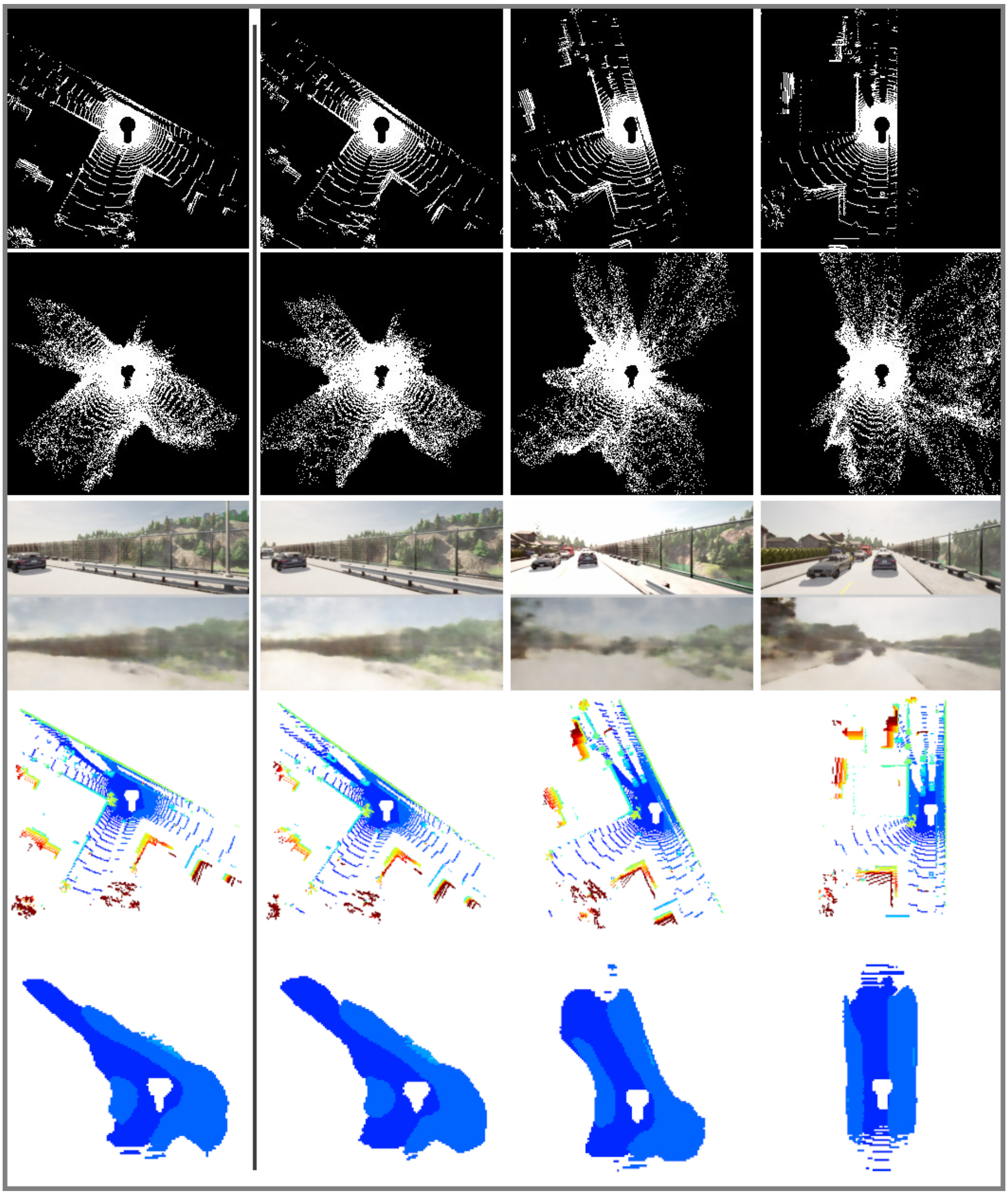


Figure 7.2: Visualization of the point cloud, image and 3D voxel Occupancy Predictions. Given several receptive frames of history observation to do predictions. The left column represents the last frame of observation, and we show the prediction of futures in 0.2s, 0.8s and 1.8s on the right. The top row of each modality is the ground truth, while the bottom is the decoded prediction. Reprinted from our previous paper [7]

A Appendix

A.1 Sensor Settings

attribute	value	description
Camera		
fov	110	Horizontal field of view in degrees.
width	960	Image width in pixels.
height	600	Image height in pixels.
location	[1, 0, 2]	[x, y, z] offset to ego vehicle in meters
rotation	[0, 0, 0)	rotation alone [x, y, z] axis to ego vehicle in rads
LiDAR		
location	[1, 0, 2]	[x, y, z] offset to ego vehicle in meters
rotation	[0, 0, 0]	rotation alone [x, y, z] axis to ego vehicle in rads
channels	64	Number of lasers.
range	100	Maximum distance to measure/ray cast in meters
rotation_frequency	10	LIDAR rotation frequency (same as collection FPS)
points_per_second	60,000	Points generated by all lasers per second.
upper_fov	10	Angle in degrees of the highest laser.
lower_fov	-30	Angle in degrees of the lowest laser.

Table A.1: The settings of sensors in CARLA with descriptions. Note: the coordinate here is the left-hand coordinate of ego_vehicle

A.2 Dataset Settings

	Town 01	Town 02	Town 03	Town 04	Town 05	Town 06
Training Set						
num of vehicles	[80, 160]	\	[40, 100]	[100, 200]	\	[80, 160]
num of walkers	[80, 160]	\	[40, 100]	[40, 120]	\	[40, 120]
Validation Set						
num of vehicles	120	70	70	150	120	120
num of walkers	120	70	70	80	120	80

Table A.2: The number of zombie vehicles and walkers in each Town on each set

A.3 Parameters in Model

parameter	value	description
	BEV Mapping	
size	[48, 48]	height and width of feature map in pixels
resolution	0.8	meters/pixel
offset forward	-64	the offset of ego vehicle to center in pixels
frustum bound	37	how many depth to estimate.
sparse count	10	only topk depth used to calculate features
	PointPillars	
num input	8	features of each point (x, y, z, xc, yc, zc, xp, yp)
num_features	[32, 32]	feature dimension of middle layer and output
min x	-48	pillars range in -x axes, in meters
max x	48	pillars range in x axes, in meters
min y	-48	pillars range in -y axes, in meters
max y	48	pillars range in y axes, in meters
pixels per meter	5	pixels per meter
	Rang View	
h	64	height in pixels, related to lidar channels
w	1024	width in pixels
	Transformer Encoder	
d_model	384	feature dimensions of tokens
num_layers	6	number of encoder layers
n_heads	8	number of attention block
drop	0.1	dropout rate
	Transformer Decoder	
d_model	384	feature dimensions of tokens
num_layers	6	number of decoder layers
n_heads	8	number of attention block
drop	0.1	dropout rate

Table A.3: The Parameters of the Networks

B List of Figures

C List of Tables

D Bibliography

[1] M. Atzmon, H. Maron, and Y. Lipman. Point convolutional neural networks by extension operators. *ACM TOG*, 37(4), 2018.

[2] X. Bai, Z. Hu, X. Zhu, Q. Huang, Y. Chen, H. Fu, and C.-L. Tai. TransFusion: Robust LiDAR-Camera Fusion for 3D Object Detection with Transformers. In *CVPR*, 2022.

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2), 1994.

[4] C. M. Bishop. Mixture density networks. 1994.

[5] C. M. Bishop. *Neural networks for pattern recognition.* Oxford university press, 1995.

[6] D. Bogdoll, L. Bosch, T. Joseph, H. Gremmelmaier, Y. Yang, and J. M. Zöllner. Exploring the Potential of World Models for Anomaly Detection in Autonomous Driving. In *IEEE Symp. Comp. Intell.*, 2023.

[7] D. Bogdoll, Y. Yang, and J. M. Zöllner. MUVO: A Multimodal Generative World Model for Autonomous Driving with Geometric Representations. *arXiv preprint arXiv:2311.11762*, 2023.

[8] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym, 2016.

[9] Q. Cai, Y. Pan, T. Yao, C.-W. Ngo, and T. Mei. ObjectFusion: Multi-modal 3D Object Detection with Object-Centric Fusion. In *ICCV*, 2023.

[10] A.-Q. Cao and R. De Charette. MonoScene: Monocular 3D Semantic Scene Completion. In *CVPR*, 2022.

[11] A.-Q. Cao and R. de Charette. SceneRF: Self-Supervised Monocular 3D Scene Reconstruction with Radiance Fields. In *ICCV*, 2023.

[12] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-End Object Detection with Transformers. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *ECCV*, 2020.

[13] CARLA team. 3rd. Maps and navigation - CARLA Documentation. `https://carla.readthedocs.io/en/0.9.13/core_map/#non-layered-maps`, 2023. Accessed: 2023-12-28.

[14] CARLA team. CARLA Autonomous Driving Leaderboard. `https://leaderboard.carla.org`, 2023. Accessed: 2023-12-28.

[15] A. Carlson, M. S. Ramanagopal, N. Tseng, M. Johnson-Roberson, R. Vasudevan, and K. A. Skinner. CLONeR: Camera-Lidar Fusion for Occupancy Grid-aided Neural Representations. *Rob. Aut. Lett.*, 8(5), 2023.

[16] L. Chang and D. Y. Tsao. The code for facial identity in the primate brain. *Cell*, 169(6), 2017.

[17] C. Chen, Y.-F. Wu, J. Yoon, and S. Ahn. Transdreamer: Reinforcement learning with transformer world models. *arXiv preprint arXiv:2202.09481*, 2022.

[18] J. Chen, S. E. Li, and M. Tomizuka. Interpretable End-to-End Urban Autonomous Driving With Latent Deep Reinforcement Learning. *IEEE Trans. Intell. Transp. Sys.*, 23, 2022.

[19] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision Transformer: Reinforcement Learning via Sequence Modeling. *arXiv:2106.01345*, 2021.

[20] K. Chitta, A. Prakash, B. Jaeger, Z. Yu, K. Renz, and A. Geiger. TransFuser: Imitation with Transformer-Based Sensor Fusion for Autonomous Driving. *IEEE TPAMI*, 2022.

[21] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[22] comma.ai. Commavq: A dataset of tokenized driving video and a GPT model, 2023.

[23] Y. Cui, R. Chen, W. Chu, L. Chen, D. Tian, Y. Li, and D. Cao. Deep learning for image and point cloud fusion in autonomous driving: A review. *IEEE Trans. Intell. Transp. Sys.*, 23(2), 2021.

[24] F. Deng, J. Park, and S. Ahn. Facing Off World Model Backbones: RNNs, Transformers, and S4. In *NeurIPS*, 2023.

[25] J. Deng, S. Shi, P. Li, W. Zhou, Y. Zhang, and H. Li. Voxel R-CNN: Towards High Performance Voxel-based 3D Object Detection. In *AAAI*, 2021.

[26] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2020.

[27] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An open urban driving simulator. In *Conf. Rob. Learn.*, 2017.

[28] A. Elluswamy. Occupancy Networks. CVPR Workshop on Autonomous Driving, 2023.

[29] L. Fan, X. Xiong, F. Wang, N. Wang, and Z. Zhang. Rangedet: In defense of range view for lidar-based 3d object detection. In *ICCV*, 2021.

[30] J. W. Forrester. Counterintuitive behavior of social systems. *Theory and decision*, 2(2), 1971.

[31] Z. Gao, Y. Mu, R. Shen, C. Chen, Y. Ren, J. Chen, S. E. Li, P. Luo, and Y. Lu. Enhance Sample Efficiency and Robustness of End-to-end Urban Autonomous Driving via Semantic Masked World Model. In *NeurIPSW*, 2022.

[32] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[33] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[34] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *NIPS*, 27, 2014.

[35] B. Graham, M. Engelcke, and L. V. D. Maaten. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. In *CVPR*, 2018.

[36] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[37] A. Gu, K. Goel, and C. Re. Efficiently Modeling Long Sequences with Structured State Spaces. In *ICLR*, 2021.

[38] D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. In *NeurIPS*, 2018.

[39] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to Control: Learning Behaviors by Latent Imagination. In *ICLR*, 2020.

[40] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning Latent Dynamics for Planning from Pixels. In *Int. Conf. Mach. Learn.*, 2019.

[41] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba. Mastering Atari with Discrete World Models. In *ICLR*, 2021.

[42] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap. Mastering Diverse Domains through World Models. *arXiv:2301.04104*, 2023.

[43] K. He and J. Sun. Convolutional neural networks at constrained time cost. In *CVPR*, 2015.

[44] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016.

[45] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *NeurIPS*, 33, 2020.

[46] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8), 1997.

[47] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[48] A. Hu, G. Corrado, N. Griffiths, Z. Murez, C. Gurau, H. Yeo, A. Kendall, R. Cipolla, and J. Shotton. Model-Based Imitation Learning for Urban Driving. In *NeurIPS*, 2022.

[49] A. Hu, L. Russell, H. Yeo, Z. Murez, G. Fedoseev, A. Kendall, J. Shotton, and G. Corrado. GAIA-1: A Generative World Model for Autonomous Driving. *arXiv:2309.17080*, 2023.

[50] J. S. K. Hu, T. Kuai, and S. L. Waslander. Point Density-Aware Voxels for LiDAR 3D Object Detection. In *CVPR*, 2022.

[51] K. Huang, B. Shi, X. Li, X. Li, S. Huang, and Y. Li. Multi-modal Sensor Fusion for Auto Driving Perception: A Survey. *arXiv:2202.02703*, 2022.

[52] Y. Huang, W. Zheng, Y. Zhang, J. Zhou, and J. Lu. Tri-Perspective View for Vision-Based 3D Semantic Occupancy Prediction. In *CVPR*, 2023.

[53] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Int. Conf. Mach. Learn.*, 2015.

[54] M. Janner, Q. Li, and S. Levine. Offline Reinforcement Learning as One Big Sequence Modeling Problem. In *NeurIPS*, 2021.

[55] X. Jia, P. Wu, L. Chen, J. Xie, C. He, J. Yan, and H. Li. Think Twice before Driving: Towards Scalable Decoders for End-to-End Autonomous Driving. In *CVPR*, 2023.

[56] H. Jiang, T. Cheng, N. Gao, H. Zhang, W. Liu, and X. Wang. Symphonize 3D Semantic Scene Completion with Contextual Instance Queries. *arXiv:2306.15670*, 2023.

[57] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019.

[58] G. B. Keller, T. Bonhoeffer, and M. Hübener. Sensorimotor mismatch signals in primary visual cortex of the behaving mouse. *Neuron*, 74(5), 2012.

[59] S. W. Kim, , J. Philion, A. Torralba, and S. Fidler. DriveGAN: Towards a Controllable High-Quality Neural Simulation. In *CVPR*, 2021.

[60] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[61] L. Kong, Y. Liu, R. Chen, Y. Ma, X. Zhu, Y. Li, Y. Hou, Y. Qiao, and Z. Liu. Rethinking range view representation for lidar segmentation. In *ICCV*, 2023.

[62] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *NeurIPS*, 25, 2012.

[63] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. PointPillars: Fast Encoders for Object Detection From Point Clouds. In *CVPR*, 2019.

[64] Y. LeCun. A Path Towards Autonomous Machine Intelligence. *OpenReview:BZ5a1r-kVsf*, 2022.

[65] M. Leinweber, D. R. Ward, J. M. Sobczak, A. Attinger, and G. B. Keller. A sensorimotor circuit in mouse cortex for visual flow predictions. *Neuron*, 95(6), 2017.

[66] X. Li, T. Ma, Y. Hou, B. Shi, Y. Yang, Y. Liu, X. Wu, Q. Chen, Y. Li, Y. Qiao, and L. He. LoGoNet: Towards Accurate 3D Object Detection with Local-to-Global Cross-Modal Fusion. In *CVPR*, 2023.

[67] X. Li, Y. Zhang, and X. Ye. DrivingDiffusion: Layout-Guided multi-view driving scene video generation with latent diffusion model. *arXiv preprint arXiv:2310.07771*, 2023.

[68] Y. Li, Y. Chen, X. Qi, Z. Li, J. Sun, and J. Jia. Unifying Voxel-based Representation with Transformer for 3D Object Detection. In *NeurIPS*, 2022.

[69] Y. Li, Z. Yu, C. Choy, C. Xiao, J. M. Alvarez, S. Fidler, C. Feng, and A. Anandkumar. Voxformer: Sparse voxel transformer for camera-based 3d semantic scene completion. In *CVPR*, 2023.

[70] Z. Li, Z. Yu, D. Austin, M. Fang, S. Lan, J. Kautz, and J. M. Alvarez. FB-OCC: 3D Occupancy Prediction based on Forward-Backward View Transformation. In *CVPRW*, 2023.

[71] T. Liang, H. Xie, K. Yu, Z. Xia, Z. Lin, Y. Wang, T. Tang, B. Wang, and Z. Tang. BEVFusion: A Simple and Robust LiDAR-Camera Fusion Framework. In *NeurIPS*, 2022.

[72] F. Liu, H. Liu, A. Grover, and P. Abbeel. Masked Autoencoding for Scalable and Generalizable Decision Making. In *NeurIPS*, 2023.

[73] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021.

[74] Z. Liu, H. Tang, A. Amini, X. Yang, H. Mao, D. L. Rus, and S. Han. BEVFusion: Multi-Task Multi-Sensor Fusion with Unified Bird's-Eye View Representation. In *ICRA*, 2023.

[75] I. Loshchilov and F. Hutter. Decoupled Weight Decay Regularization. In *ICLR*, 2018.

[76] Y. Man, L.-Y. Gui, and Y.-X. Wang. BEV-Guided Multi-Modality Fusion for Driving Perception. In *CVPR*, 2023.

[77] P. Mattes, R. Schlosser, and R. Herbrich. Hieros: Hierarchical Imagination on Structured State Space Sequence World Models. *arXiv preprint arXiv:2310.05167*, 2023.

[78] R. Miao, W. Liu, M. Chen, Z. Gong, W. Xu, C. Hu, and S. Zhou. OccDepth: A Depth-Aware Method for 3D Semantic Scene Completion. *arXiv:2302.13540*, 2023.

[79] V. Micheli, E. Alonso, and F. Fleuret. Transformers are Sample-Efficient World Models. In *ICLR*, 2022.

[80] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[81] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss. Rangenet++: Fast and accurate lidar semantic segmentation. In *IROS*, 2019.

[82] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Int. Conf. Mach. Learn.*, 2010.

[83] M. Pan, J. Liu, R. Zhang, P. Huang, X. Li, L. Liu, and S. Zhang. RenderOcc: Vision-Centric 3D Occupancy Prediction with 2D Rendering Supervision. *arXiv:2309.09502*, 2023.

[84] M. Pan, L. Liu, J. Liu, P. Huang, L. Wang, S. Zhang, S. Xu, Z. Lai, and K. Yang. UniOcc: Unifying Vision-Centric 3D Occupancy Prediction with Geometric and Semantic Rendering. In *CVPRW*, 2023.

[85] S. Pang, D. Morris, and H. Radha. Fast-CLOCs: Fast Camera-LiDAR Object Candidates Fusion for 3D Object Detection. In *WACV*, 2022.

[86] J. Philion and S. Fidler. Lift, Splat, Shoot: Encoding Images from Arbitrary Camera Rigs by Implicitly Unprojecting to 3D. In *ECCV*, 2020.

[87] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *CVPR*, 2018.

[88] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017.

[89] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *NeurIPS*, 30, 2017.

[90] Y. Qin, C. Wang, Z. Kang, N. Ma, Z. Li, and R. Zhang. SupFusion: Supervised LiDAR-Camera Fusion for 3D Object Detection. In *ICCV*, 2023.

[91] R. Q. Quiroga, L. Reddy, G. Kreiman, C. Koch, and I. Fried. Invariant visual representation by single neurons in the human brain. *Nature*, 435(7045), 2005.

[92] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *Int. Conf. Mach. Learn.*, 2021.

[93] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, and N. de Freitas. A Generalist Agent. In *Trans. Mach. Learn. Research*, 2022.

[94] C. B. Rist, D. Emmerichs, M. Enzweiler, and D. M. Gavrila. Semantic Scene Completion using Local Deep Implicit Functions on LiDAR Data. In *IEEE TPAMI*, 2021.

[95] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

[96] D. E. Rumelhart, G. E. Hinton, R. J. Williams, et al. Learning internal representations by error propagation, 1985.

[97] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115, 2015.

[98] H. Shao, L. Wang, R. Chen, H. Li, and Y. Liu. Safety-Enhanced Autonomous Driving Using Interpretable Sensor Fusion Transformer. In *Conf. Rob. Learn.*, 2022.

[99] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li. PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection. In *CVPR*, 2020.

[100] S. Shi, L. Jiang, J. Deng, Z. Wang, C. Guo, J. Shi, X. Wang, and H. Li. PV-RCNN++: Point-Voxel Feature Set Abstraction With Local Vector Representation for 3D Object Detection. *IJCV*, 131(2), 2023.

[101] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *NeurIPS*, 28, 2015.

[102] C. Sima, W. Tong, T. Wang, L. Chen, S. Wu, H. Deng, Y. Gu, L. Lu, P. Luo, D. Lin, and H. Li. Scene as occupancy. In *ICCV*, 2023.

[103] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[104] V. A. Sindagi, Y. Zhou, and O. Tuzel. MVX-Net: Multimodal voxelnet for 3D object detection. In *ICRA*. IEEE, 2019.

[105] J. T. Smith, A. Warrington, and S. Linderman. Simplified State Space Layers for Sequence Modeling. In *The Eleventh International Conference on Learning Representations*, 2022.

[106] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Int. Conf. Mach. Learn.*, 2015.

[107] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. Semantic Scene Completion from a Single Depth Image. In *CVPR*, 2017.

[108] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[109] R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4), 1991.

[110] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[111] M. Tan and Q. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *Int. Conf. Mach. Learn.* PMLR, 2019.

[112] M. Tan and Q. Le. Efficientnetv2: Smaller models and faster training. In *Int. Conf. Mach. Learn.* PMLR, 2021.

[113] Z. Tan, Z. Dong, C. Zhang, W. Zhang, H. Ji, and H. Li. OVO: Open-Vocabulary Occupancy. *arXiv:2305.16133*, 2023.

[114] H. Tang, Z. Liu, S. Zhao, Y. Lin, J. Lin, H. Wang, and S. Han. Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution. In A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, editors, *ECCV*, 2020.

[115] Z. Teed and J. Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *ECCV*, 2020.

[116] Z. Teed and J. Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *NeurIPS*, 34, 2021.

[117] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *ICCV*, 2019.

[118] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *IROS*, 2012.

[119] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, and Y. Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6, 2020.

[120] A. van den Oord, O. Vinyals, and k. kavukcuoglu. Neural Discrete Representation Learning. In *NeurIPS*, volume 30, 2017.

[121] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is All you Need. In *NeurIPS*, volume 30, 2017.

[122] S. Vora, A. H. Lang, B. Helou, and O. Beijbom. Pointpainting: Sequential fusion for 3d object detection. In *CVPR*, 2020.

[123] T. Vu, J.-H. Kim, M. Kim, S. Jung, and S.-G. Jeong. MiLO: Multi-task Learning with Localization Ambiguity Suppression for Occupancy Prediction. In *CVPRW*, 2023.

[124] Z. Wan, Y. Mao, J. Zhang, and Y. Dai. RPEFlow: Multimodal Fusion of RGB-PointCloud-Event for Joint Optical Flow and Scene Flow Estimation. In *ICCV*, 2023.

[125] X. Wang, Z. Zhu, G. Huang, X. Chen, and J. Lu. DriveDreamer: Towards Real-world-driven World Models for Autonomous Driving. *arXiv:2309.09777*, 2023.

[126] Y. Wang, Y. Chen, X. Liao, L. Fan, and Z. Zhang. PanoOcc: Unified Occupancy Representation for Camera-based 3D Panoptic Segmentation. *arXiv:2306.10013*, 2023.

[127] Y. Wang, Q. Mao, H. Zhu, J. Deng, Y. Zhang, J. Ji, H. Li, and Y. Zhang. Multi-modal 3d object detection in autonomous driving: a survey. *IJCV*, 2023.

[128] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *ACM TOG*, 38(5), 2019.

[129] Waymo. Introducing the 5th-generation waymo driver, 2020.

[130] Y. Wei, L. Zhao, W. Zheng, Z. Zhu, J. Zhou, and J. Lu. SurroundOcc: Multi-Camera 3D Occupancy Prediction for Autonomous Driving. In *ICCV*, 2023.

[131] R. Wightman. PyTorch Image Models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[132] Wikipedia contributors. Michigan left - Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Michigan_left&oldid=1188685531`, 2023. Accessed: 2023-12-28.

[133] F. Wimbauer, N. Yang, C. Rupprecht, and D. Cremers. Behind the Scenes: Density Fields for Single View Reconstruction. In *CVPR*, 2023.

[134] B. Wu, A. Wan, X. Yue, and K. Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. In *ICRA*, 2018.

[135] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer. SqueezeSegV2: Improved Model Structure and Unsupervised Domain Adaptation for Road-Object Segmentation from a LiDAR Point Cloud. In *ICRA*, 2019.

[136] P. Wu, A. Majumdar, K. Stone, Y. Lin, I. Mordatch, P. Abbeel, and A. Rajeswaran. Masked trajectory models for prediction, representation, and control. In *International Conference on Machine Learning*, 2023.

[137] W. Wu, Z. Qi, and L. Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *CVPR*, 2019.

[138] Y. Xie, C. Xu, M.-J. Rakotosaona, P. Rim, F. Tombari, K. Keutzer, M. Tomizuka, and W. Zhan. SparseFusion: Fusing Multi-Modal Sparse Representations for Multi-Sensor 3D Object Detection. In *ICCV*, 2023.

[139] C. Xu, B. Wu, Z. Wang, W. Zhan, P. Vajda, K. Keutzer, and M. Tomizuka. Squeezesegv3: Spatially-adaptive convolution for efficient point-cloud segmentation. In *ECCV*, 2020.

[140] J. Xu, R. Zhang, J. Dou, Y. Zhu, J. Sun, and S. Pu. RPVNet: A Deep and Efficient Range-Point-Voxel Fusion Network for LiDAR Point Cloud Segmentation. In *ICCV*, 2021.

[141] J. Yan, Y. Liu, J. Sun, F. Jia, S. Li, T. Wang, and X. Zhang. Cross Modal Transformer: Towards Fast and Robust 3D Object Detection. In *ICCV*, 2023.

[142] X. Yan, J. Gao, J. Li, R. Zhang, Z. Li, R. Huang, and S. Cui. Sparse Single Sweep LiDAR Point Cloud Segmentation via Learning Contextual Shape Priors from Scene Completion. In *AAAI*, 2020.

[143] Y. Yan, Y. Mao, and B. Li. SECOND: Sparsely Embedded Convolutional Detection. *Sensors*, 18, 2018.

[144] L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui, and M.-H. Yang. Diffusion models: A comprehensive survey of methods and applications. *ACM Computing Surveys*, 56(4), 2023.

[145] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia. Ipod: Intensive point-based object detector for point cloud. *arXiv preprint arXiv:1812.05276*, 2018.

[146] M. Ye, S. Xu, and T. Cao. HVNet: Hybrid Voxel Network for LiDAR Based 3D Object Detection. In *CVPR*, 2020.

[147] Y. Zeng, D. Zhang, C. Wang, Z. Miao, T. Liu, X. Zhan, D. Hao, and C. Ma. LIFT: Learning 4D LiDAR Image Fusion Transformer for 3D Object Detection. In *CVPR*, 2022.

[148] L. Zhang, Y. Xiong, Z. Yang, S. Casas, R. Hu, and R. Urtasun. Learning Unsupervised World Models for Autonomous Driving via Discrete Diffusion. *arXiv:2311.01017*, 2023.

[149] W. Zhang, G. Wang, J. Sun, Y. Yuan, and G. Huang. STORM: Efficient Stochastic Transformer based World Models for Reinforcement Learning. In *NeurIPS*, 2023.

[150] Y. Zhang, Z. Zhu, and D. Du. OccFormer: Dual-path Transformer for Vision-based 3D Semantic Occupancy Prediction. *arXiv:2304.05316*, 2023.

[151] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. Van Gool. End-to-End Urban Driving by Imitating a Reinforcement Learning Coach. In *ICCV*, 2021.

[152] Z. Zhang, Z. Zhang, Q. Yu, R. Yi, Y. Xie, and L. Ma. LiDAR-Camera Panoptic Segmentation via Geometry-Consistent and Semantic-Aware Alignment. In *ICCV*, 2023.

[153] X. Zhao, Z. Liu, R. Hu, and K. Huang. 3D object detection using scale invariant and feature reweighting networks. In *AAAI*, volume 33, 2019.

[154] W. Zheng, W. Chen, Y. Huang, B. Zhang, Y. Duan, and J. Lu. OccWorld: Learning a 3D Occupancy World Model for Autonomous Driving. *arXiv preprint arXiv:2311.16038*, 2023.

[155] Y. Zhou and O. Tuzel. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In *CVPR*, 2018.