An End-to-End Transformer Model for 3D Object Detection Further Experiments

Fei Meng, Killian Steunou
March 20, 2025
Nuages de Points et Modélisation 3D - MVA 2024/2025

Abstract

3DETR [4] is an end-to-end Transformer based object detection model for 3D point clouds. It makes minimal modifications to the Transformer based on DETR [1], a model for 2D images, to adapt it for 3D point data. At the same time, it is a one-step model, meaning it does not require a pretrained model as DETR does. Compared with other detection methods for 3D data, it uses fewer hand-tuned hyperparameters, fewer hand-designed inductive biases, and non-parametric queries, making the model much easier to implement. In this report, we explain the method, evaluate the performance of 3DETR, study the impact of the test-time number number of queries, and experiment a version of the model that uses RGB values of point clouds.

1. Introduction

In 3D object detection, using point clouds helps maintain simplicity in training and model usage. Unlike other data representations such as voxels and meshes, point clouds eliminate the need for complex and time-consuming preprocessing while preserving all the original information. Among existing models that directly use point clouds, one of the most popular is VoteNet [7].

1.1. VoteNet

VoteNet is also an end-to-end model that combines Hough voting with deep learning. It uses an encoder-decoder architecture. In the encoder stage, it learns point cloud features through a backbone network, a PointNet++ network [6]. In the decoder stage, it generates votes for object centroids using deep Hough voting. These votes are then clustered through sampling and grouping to form object proposals using a shared PointNet [5]. Finally, the model outputs 3D bounding boxes and classifies each object in the 3D point cloud.
Even though VoteNet is an end-to-end model, it consists of three main blocks, making it more challenging to adjust hyperparameters. Moreover, it requires careful manual design, such as selecting appropriate hand-encoded inductive biases, tuning radii, designing specialized 3D operators, and defining suitable loss functions. When the data changes, the model must be redesigned to accommodate variations in size and sparsity.

1.2. Transformer and DETR

The Transformer uses an attention mechanism. It is permutation-invariant and can leverage long-range contextual information to learn features, making it well-suited for unordered point cloud data. Since the model needs to learn the relationships between each point and the entire point cloud. Therefore, using a Transformer could be a powerful and simple approach for 3D point cloud data.
For object detection tasks, DETR [1] is an end-to-end model that eliminates the need for many hand-designed components. It relies on a backbone network to extract image features, followed by an encoder-decoder Transformer to generate bounding boxes and classify objects. The encoder learns the context of the image, while the decoder learns object queries instead of relying on human-designed anchors to generate object bounding boxes. DETR uses Hungarian loss, eliminating the need for Non-Maximum Suppression (NMS), which requires manually selecting a threshold hyperparameter.
However, it was originally designed for 2D images. The authors attempted to adapt it for 3D point cloud data with minimal modifications.

2. 3DETR

The 3DETR model maintains the end-to-end Transformer architecture. Furthermore, it does not require a convolutional backbone to preprocess the data, allowing it to be trained from scratch. It utilizes non-parametric queries to eliminate the need for users to define the queries manually. To enhance performance, the model incorporates Fourier positional embeddings [2], and it offers a variant, 3DETR-m using hand-designed inductive biases.

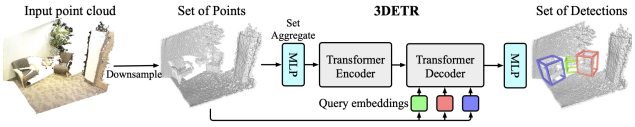


Figure 1. Pipeline of 3DETR

2.1. Pipeline

3DETR takes the XYZ coordinates of an unordered 3D point cloud as input and predicts 3D bounding boxes and object classes for the scene as showed in Figure 1.

2.1.1 Downsampling data

3D point cloud data typically contains a large number of points, so downsampling is necessary. Instead of uniform sampling, 3DETR uses the set-aggregation downsampling operation from PointNet++ to preserve the data's features while reducing the number of points.
First, Farthest Point Sampling (FPS) is used to ensure that the sampled points cover the entire point cloud. Next, the points are grouped using Ball Query or K-Nearest Neighbors. Then, an MLP is used to learn the features of each point group. Finally, to keep the feature size small, max pooling is applied to reduce the feature number.

2.1.2 Encoder-Decoder Transformer

The encoder combines multiple layers of self-attention and non-linear projections, providing output features of the same dimension. One key difference from DETR is that it doesn't require positional embeddings, as it directly uses the XYZ coordinates, which already contain the spatial information of the data.
Next, the decoder is applied. Like DETR, it predicts the bounding boxes simultaneously, allowing for parallelized predictions. The decoder takes the output of the encoder, query embeddings, and positional embeddings as input. It then generates a set of features, which are used by the prediction MLPs to predict the bounding boxes and classes for each object.

2.1.3 Fourier positional embeddings

Since the decoder uses features instead of XYZ coordinates like the encoder, it requires additional positional information. The authors tested different positional embeddings and found that Fourier positional embedding provides the best performance in Fig2.

2.1.4 Non-parametric query embeddings

To generate the final bounding boxes, the model uses query embeddings to help determine the number of objects and

#	Method	Positional Embedding		Query Type	ScanNetV2	
		Encoder	Decoder		AP_{25}	AP_{50}
1	3DETR	-	Fourier	np + Fourier	62.7	37.5
2		Fourier	Fourier	np + Fourier	61.8	37.0
3		Sine	Sine	np + Sine	55.8	30.9
4		-	-	np + Sine	31.3	10.8
5	DETR [4]†	Sine	Sine	parametric [4]	15.4	5.3

np: non-parametric query (§ 3.2)

Figure 2. Decoder Query Type and Positional Embedding (table 5 in article)

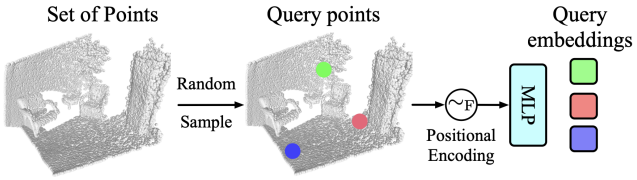


Figure 3. Query embeddings

localize the results. These query embeddings are learnable and do not have any fixed parameters, making the approach much simpler than using predefined anchors.
To create the query embeddings, the model first uses Farthest Point Sampling (FPS) to sample points from the downsampled point cloud. Then, it applies the Fourier positional encoding method to encode the XYZ coordinates of these points, generating higher-dimensional features. Finally, an MLP is applied to compress these features, making them more lightweight. An overview is shown in Figure 3.

2.1.5 Bounding box parametrization and prediction

The final MLPs predict a 3D bounding box characterized by four attributes:

- **Location c** – The coordinates of the center of the bounding box.
- **Size d** – The 3D dimensions of the bounding box.
- **Class s** – The category of the object inside the bounding box, represented as a one-hot vector.
- **Orientation a** – The direction of the bounding box, represented using a bin-based classification a_c with a residual a_r for fine-grained precision. It is not mandatory. However, for some datasets, such as the SUN RGB-D dataset, the orientation of the bounding box is required.

To determine the location, the model reuses the query's coordinates and adds an offset predicted by the MLPs to represent the center of the bounding box.
The size prediction corresponds to the 3D dimensions of the bounding box which is a 3D rectangle.
The class is represented as a one-hot vector. If there is no

object around the query coordinates, the class label will be set to "background" or "not an object."
For the orientation, instead of predicting an angle directly, the model first predicts the class corresponding to one of 12 discrete angle bins within the range $[0, 2\pi)$. Then, it predicts a residual value to refine the angle for higher precision. This approach is used because angles are periodic and not continuous, making direct learning more challenging. For example, an angle of 1° and 365° are essentially close, but a direct numerical difference does not capture their relationship. By using discrete bins, the model can better represent the relationship between neighboring angles, making learning more stable and effective.
Finally, for the prediction of model, each predicted box is presented by $\hat{b} = [\hat{c}, \hat{d}, \hat{a}, \hat{s}]$.

$$\hat{c}, \hat{d} \in [0,1]^3, \quad \hat{s} \in [0,1]^{K+1}, \quad \hat{a} = [\hat{a}_c, \hat{a}_r]$$

K is the number of object classes.

2.1.6 Set matching

During training, there are ground truth bounding boxes b for each point cloud. It uses the Hungarian algorithm to compute the optimal bipartite matching between all the predicted boxes $\hat{b}$ and ground truth boxes b, as in DETR. However, with 3D data, it formulates a matching cost.
A matching cost is defined to facilitate the box matching. The cost consists of a geometric part, which incorporates the overlap of the boxes and the distance between the centers of the boxes, and a semantic part, which uses the class data $\hat{s}$ and s. It not only focuses on the predicted class but also utilizes the distribution of $\hat{s}$ and s. It then measures the likelihood of the ground truth class and the likelihood of the box features belonging to a foreground class.

$$\text{matching cost}(\hat{b}, b) = -\lambda_1 \text{GIoU}(\hat{b}, b) + \lambda_2 \|\hat{c} - c\|_1])$$
$$-\lambda_3 \hat{s}[\text{s}_{\text{gt}}] + \lambda_4 (1 - \hat{s}[\text{s}_{\text{bg}}])$$

For hyperparameters λ, the authors set λ_1,λ_2,λ_3,λ_4 as 2,1,0,0 for ScanNetV2 and 3,5,1,5 for SUN RGB-D.

2.1.7 Loss function

After matching the predicted boxes with the ground truth boxes, we can calculate the loss between each pair of boxes.

$$L_{3DETR} = \lambda_c \|\hat{c} - c\|_1 + \lambda_d \|\hat{d} - d\|_1$$
$$+\lambda_{ar} \|\hat{a}_r - a_r\|_{\text{huber}}$$
$$-\lambda_{\text{ac}} a_c^\top \log \hat{a}_c - \lambda_s s_c^\top \log \hat{s}_c$$

The center and dimensions of the boxes are evaluated using l_1 regression losses and normalized within the range [0, 1] for scale invariance.
The angular residuals of the boxes use the Huber regression loss.
The angular class of bins and the object class are evaluated using entropy losses.
For the weights of λ, the authors set λ_c, λ_d, λ_{ar}, λ_{ac}, λ_s to 5, 1, 1, 0.1, 5 respectively.
There are then two special cases for calculating the loss. One case occurs when the predicted box is considered as background. In this case, only the cross-entropy loss of the semantic class term is calculated. The weight for the cross-entropy loss is 0.2 for the 'background' class and 0.8 for the K object classes.
The other case arises when the bounding boxes of the dataset are axis-aligned. In this case, a loss is directly applied to the Generalized Intersection over Union (GIoU).

2.1.8 3DETR-m

3DETR-m uses a masked self-attention encoder. Due to the flexible structure of 3DETR, the model can be easily modified. In this case, the encoder is adjusted to incorporate inductive biases for 3D data. Since neighboring points are generally more important than distant points for learning object features, each layer of the encoder applies a binary mask to the self-attention mechanism. This mask ensures that each point focuses only on its neighbors. The model uses an l_2 radius to define the neighbor's radius for each point, with radius values set to [0.16, 0.64, 1.44].
Although 3DETR-m introduces additional hyperparameters, it is still simpler to train compared to PointNet++. It only adds a mask and does not require multi-layer 3D feature aggregation or upsampling.

3. Experiments

During our work, we first used a pretrained model to verify the results reported in the article. Then, we conducted experiments to explore potential improvements to the model. Due to the lack of CUDA support, we were unable to run the code on our local computers. Instead, we used Google Cloud and Lightning.ai to execute the code, which limited our access to computational resources.

3.1. Preparation SUN RGB-D dataset

Due to limited memory, we chose to use the SUN RGB-D dataset for our work, as it is significantly smaller than the ScanNet dataset. In the article, they choose the SUN RGB-D-v1 dataset which has 5K single-view RGB-D training samples with oriented bounding box annotations for 37 object categories. But as mentioned in the article, the evaluation for the SUN RGB-D dataset is conducted only on the 10 most frequent categories.
To adapt to the model's usage, the authors reuse the code from VoteNet to extract point clouds and annotations and

generating a well-structured dataset for 3DETR.
Here is an example before preprocessing the data: for the 2D image token in SUN RGB-D dataset 4, it can generate a point cloud 5a representation along with a 3D bounding box5b.

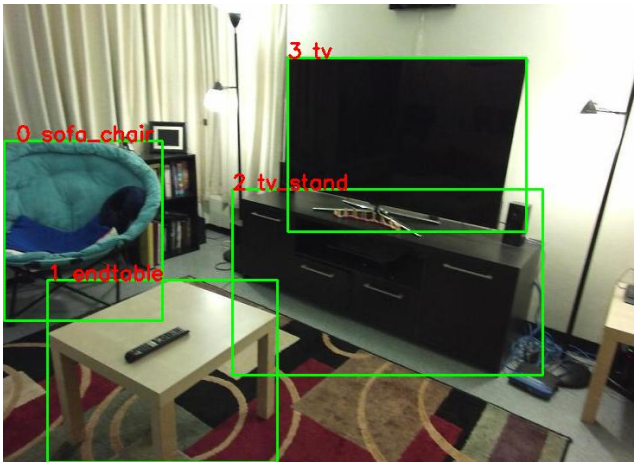
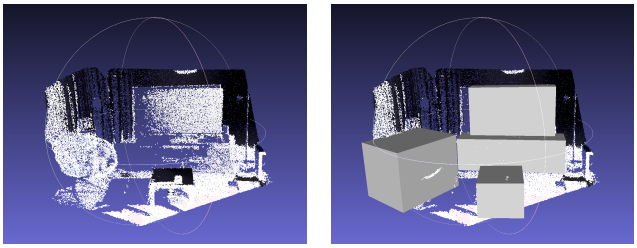


Figure 4. Example of an image with 2D bounding boxes from the SUN RGB-D dataset



(a) point cloud (b) with bounding box

Figure 5. The point cloud and 3D bounding box created for the image

To reduce the dataset size for training a new model, we created a smaller dataset, ensuring that at least 30 objects for each class are represented in the training set and 10 objects for each class are represented in the test set. Finally, we obtained 195 point clouds for the training set and 67 point clouds for the testing set.

3.2. Verify the results of pretrained model

We evaluate the 3DETR-m pretrained model given by the authors of [4] on SUN RGB-D validation set and recover similar results as in the original paper, which are shown in Table 1. We use 128 queries at test time, as recommended in [4] for a good performance/computation time ratio. The model has been also trained with 128 queries.
In general, the results of our test are very close to the public score. The differences in test performance are probably due to the floating-point precision of the GPU used.

Metric	[4] Results	Our Results
Mean AP_{25}	59.1	58.43
Mean AP_{50}	32.7	31.41
Bed AP_{25}	84.6	84.12
Table AP_{25}	52.6	48.89
Sofa AP_{25}	65.3	64.86
Chair AP_{25}	72.4	72.48
Toilet AP_{25}	91.0	90.68
Desk AP_{25}	34.3	31.13
Dresser AP_{25}	29.6	29.50
Nightstand AP_{25}	61.4	61.56
Bookshelf AP_{25}	28.5	28.14
Bathtub AP_{25}	69.8	72.89

Table 1. [4] reported mean AP_{25}, mean AP_{50} and per-class AP_{25} for SUN RGB-D vs. results found by testing 3DETR-m locally.

3.3. Verify impact of number of queries

Then, we verified the impact on the test time number of queries on the mean average precision. We tested the number of queries ranging from 16 to 512. The result is shown on Figure 6. Using more than 128 queries at test time improves the performance very little, while it increases computation time linearly with the number of queries.

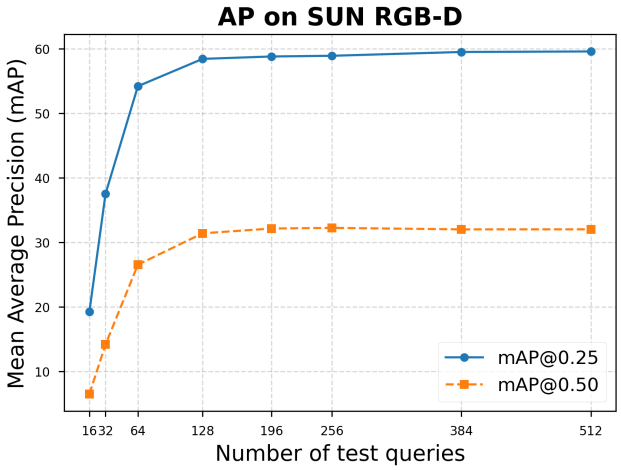


Figure 6. Evolution of mean AP_{25} and mean AP_{50} with the number of queries at test time.

For 10 classes in SUN RGB-D dataset, the per-class evolution (Figure 7) of average precision with the number of queries shows a similar pattern across all categories. Therefore, 128 queries is a good choice for the SUN RGB-D dataset.
Qualitatively, the predicted bounding boxes are close to the ground truth, as we can see in Figure 8, which is coherent

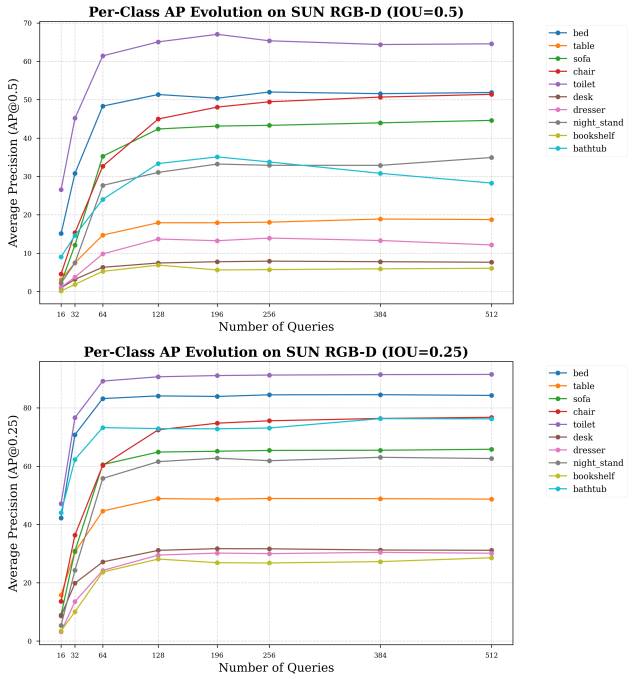


Figure 7. Evolution of AP_{25} (bottom) and AP_{50} (top) with the number of queries at test time for each category.

with the model's performance.

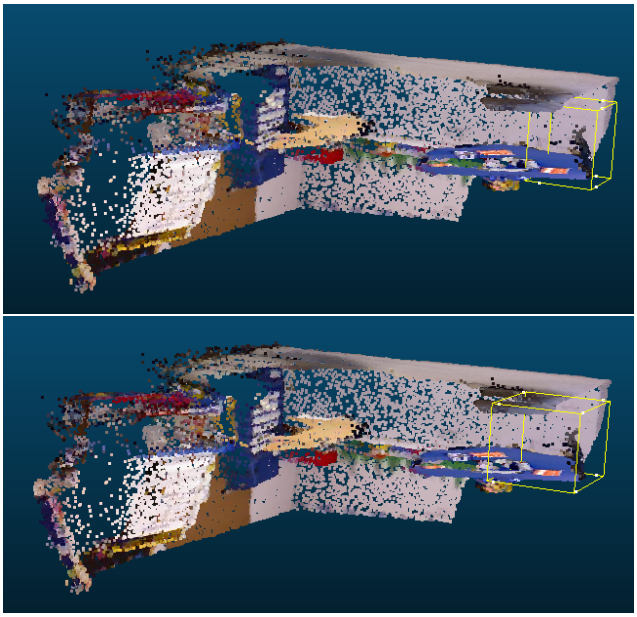


Figure 8. Visualization of a ground truth bounding box (top) and a bounding box predicted by the pretrained 3DETR-m model, overlaid on the input point cloud (colors are for visualization only, they were not given to the model) using CloudCompare.

3.4. Our own experiments using RGB channels

In this experiment, we study the benefit of using RGB channels in addition to points coordinates. To this end, we train two different models on the smaller dataset discussed in Section 3.1: one 3DETR-m only with points coordinates, and one 3DETR-m with points coordinates and RGB channels.

We trained each model for 350 epochs, using 128 queries, a batch size of 8, a weight decay of 0.01 (original 3DETR-m uses a weight decay of 0.1), a base learning rate of 1×10^{-4}, and gradient clipping to 1 (original 3DETR-m clips gradient to 0.1). We evaluate both on the same smaller test set as discussed in Section 3.1, using 128 queries.

The results are reported in Table 2. Using RGB seems to severly impact performance, but a larger scale experiment would be needed to conclude this confidently. It is likely that the optimal number of queries at training time is different when not using RGB and when using it. Unfortunately, training 3DTER-m with RGB on a NVIDIA V100 GPU on the full SUN RGB-D dataset requires approximately 5 days (130 hours), which we cannot afford. It took about 6 hours to train and evaluate each model on the smaller dataset.

Metric	3DTER-m	3DETR-m RGB
Mean AP_{25}	11.59	3.31
Mean AP_{50}	1.08	0.06
Bed AP_{25}	20.60	5.04
Table AP_{25}	0.85	1.29
Sofa AP_{25}	20.35	5.22
Chair AP_{25}	0.18	0.02
Toilet AP_{25}	22.85	5.66
Desk AP_{25}	4.54	2.44
Dresser AP_{25}	1.24	1.94
Nightstand AP_{25}	0.80	0.50
Bookshelf AP_{25}	9.88	7.33
Bathtub AP_{25}	34.66	3.63

Table 2. [4] Comparison of mean AP_{25}, mean AP_{50} and per-class AP_{25} for smaller SUN RGB-D, with 3DETR-m and 3DETR-m RGB

Due to its simplicity, the only hyperparameter that is interesting to study at test time is the number of queries, but any other relevant experiment would require training a new model, limiting a lot the possibility of experiments here. Also, the provided code from [4] runs only on a specific version of CUDA, and *only* on CUDA, which means testing on CPU is not even an option. Since we already spent almost 35€ on Google Cloud to perform 10 evaluations and train two new models, we cannot go any further. This is the main limitation of our work.

4. Limitations and Improvements Proposal

3DETR does not achieve state-of-the-art precision. There are still several ways to further improve performance.

For input data, while the point cloud has several advantages, its sparsity and uneven distribution make obtaining suitable object queries quite challenging. Additionally, the computation of attention is computationally complex. There are potential improvements, such as SEED [3], which uses sparse attention, reduces the number of decoder layers, and shares parameters to accelerate the model. It also employs better query initialization, performs multiple subsampling steps to create multi-scale features, and utilizes a more effective matching algorithm, dynamic label assignment, to improve the accuracy.

For hyperparameters, instead of using constant value, adaptive computation for the number of queries is also a promising direction, and research into efficient self-attention mechanisms could provide further benefits.

For component of the model, there are newer techniques to replace, such as improved Fourier positional embeddings, which can offer better positional encoding performance [2]. Additionally, better encoder designs based on the decoder and loss functions could further enhance performance.

For prediction, similar innovations used to enhance VoteNet could be integrated into our model in the future, such as predicting a hybrid set of geometric primitives, including point centers, edge centers, and face centers, or using a hierarchical graph network in Transformers to dynamically adjust the radius for 3DETR-m.

It would also be compelling to evaluate 3DETR on emerging LiDAR-centric datasets like KITTI. While achieving superior performance with pure RGB data remains challenging, integrating complementary modalities such as depth or LiDAR could unlock new insights. For instance, combining sparse LiDAR point clouds with dense RGB features through cross-modal attention might enhance robustness in complex 3D scenarios. Unlike RGB-based methods that struggle with illumination variations, LiDAR provides geometry-first representations. A hybrid approach leveraging 3DETR's attention mechanism to fuse LiDAR's geometric consistency and RGB's texture richness could address challenges like occlusions in autonomous driving benchmarks.

Beyond object detection, 3DETR could also be extended to other 3D tasks, such as 3D instance segmentation, 3D object tracking, 3D shape completion, 3D scene understanding, and reconstruction, among others.

5. Discussion

3DETR utilizes the Transformer architecture from DETR with minimal modifications to the vanilla Transformer block, allowing it to process 3D point cloud data directly. The use of non-parametric queries reduces the need for hand-tuned hyperparameters compared to other 3D-specific models, while Fourier positional embeddings contribute to competitive performance. 3DETR effectively learns point cloud features, allowing it to detect objects even from partial scans. It can also predict amodal bounding boxes and handle missing annotations in dataset.

This end-to-end model is simple, easy to implement, and offers a flexible pipeline, making it straightforward to exchange components. Additionally, the model can adapt to different inference times without requiring retraining, as adjusting the number of layers is sufficient.

While the model currently relies on manual hyperparameter tuning and has not yet reached state-of-the-art precision levels, it serves as a promising baseline framework for future enhancements.

6. Contributions

Killian ran 3DETR on Google Cloud to verify the results of the pretrained model and to train a new model using additional RGB data. In the end, he completed half of the report. Fei didn't have enough memory to run the entire SUN RGB-D dataset, so she created a smaller dataset and tested a new 3D DETR model SEED [3] for point clouds using this dataset on lightning.ai platform. Due to policy restrictions, there was no public pretrained model available. She was unable to successfully train the model, as the code was specifically designed for the Waymo dataset, making it difficult to adapt to a custom dataset. In the end, she also completed half of the report.

References

[1] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020. 1

[2] Ermo Hua, Che Jiang, Xingtai Lv, Kaiyan Zhang, Ning Ding, Youbang Sun, Biqing Qi, Yuchen Fan, Xue Kai Zhu, and Bowen Zhou. Fourier position embedding: Enhancing attention's periodic extension for length generalization. *arXiv preprint arXiv:2412.17739*, 2024. 1, 6

[3] Zhe Liu, Jinghua Hou, Xiaoqing Ye, Tong Wang, Jingdong Wang, and Xiang Bai. Seed: A simple and effective 3d detr in point clouds. 2024. 6

[4] Ishan Misra, Rohit Girdhar, and Armand Joulin. An end-to-end transformer model for 3d object detection, 2021. 1, 4, 5

[5] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2017. 1

[6] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017. 1

[7] Charles R. Qi, Or Litany, Kaiming He, and Leonidas J. Guibas. Deep hough voting for 3d object detection in point clouds, 2019. 1