Supplemental Material

A. Implementation Details

A.1. Architecture

We describe the 3DETR architecture in detail.

Architecture. We follow the dataset preprocessing from [42] and obtain $N = 20,000$ points and $N = 40,000$ points respectively for each sample in SUN RGB-D and ScanNetV2 datasets. The $N \times 3$ matrix of point coordinates is then passed through one layer of the downsampling and set aggregation operation [45] which uses Farthest-Point-Sampling to sample 2048 points randomly from the scene. Each point is projected to a 256 dimensional feature followed by the set-aggregation operation that aggregates features within a ℓ_2 distance of 0.2. The output is a 2048×256 dimensional matrix of features for the $N' = 2048$ points which is input to the encoder. We now describe the encoder and decoder architectures (illustrated in Fig 6).

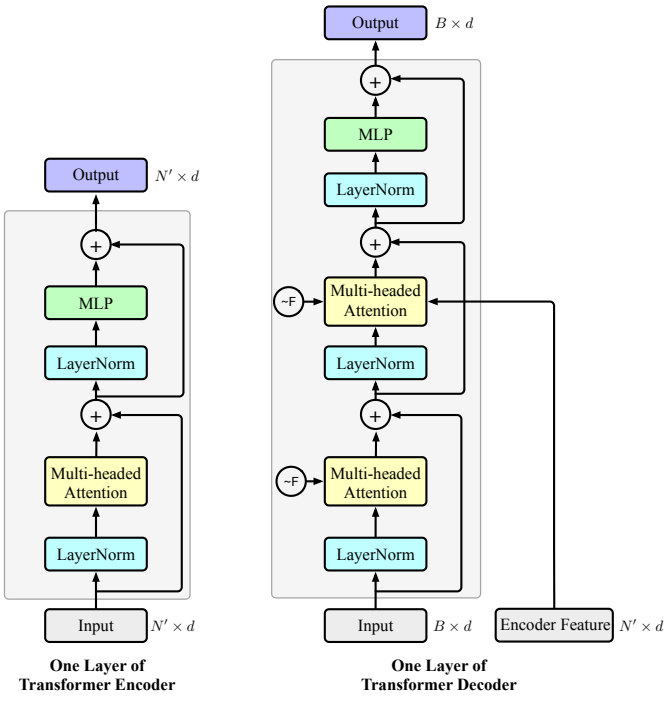

Figure 6: Architecture of Encoder and Decoder. We present the architecture for one layer of the 3DETR encoder and decoder. The encoder layer takes as input $N' \times d$ features for N' points and outputs $N' \times d$ features too. It performs self-attention followed by a MLP. The decoder takes as input $B \times d$ features (the query embeddings or the prior decoder layer), $N' \times d$ point features from the encoder to output $B \times d$ features for B boxes. The decoder performs self-attention between the B query/box features and cross-attention between the B query/box features and the N' point features. We denote by $\sim_F$ the Fourier positional encodings [64] used in the decoder. All 3DETR models use $d = 256$.

Encoder. The encoder has three layers of self-attention followed by an MLP. The self-attention operation uses multi-headed attention with four heads. The self-attention produces a 2048×2048 attention matrix which is used to attend to the features to produce a 256 dimensional output. The MLPs in each layer have a hidden dimension with 128. All the layers use LayerNorm [2] and the ReLU non-linearity.

3DETR-m Encoder. The masked 3DETR-m encoder has three layers of self-attention followed by an MLP. At each layer the self-attention matrix of size `#points`$\times$`#points` is multiplied with a binary mask M of the same size. The binary mask entry M_{ij} is 1 if the point coordinates for points i and j are within a radius r of each other. We use radius values of $[0.4, 0.8, 1.2]$ for the three layers. The first layer operates on 2048 points and is followed by a downsample + set aggregation operator that downsamples to 1024 points using a radius of 0.4, similar to PointNet++. The encoder layers follow the same structure as the vanilla Encoder described above, *i.e.*, MLPs with hidden dimension of 128, multi-headed attention with four heads *etc*. The encoder produces 256 dimensional features for 1024 points.

Decoder. The decoder operates on the $N' \times 256$ encoder features and $B \times 256$ location query embeddings. It produces a $B \times 256$ matrix of box features as output. The decoder has eight layers and uses cross-attention between the location query embeddings (Sec 3.2 main paper) and the encoder features, and self-attention between the box features. Each layer has the self-attention operation followed by a cross-attention operation (implemented exactly as self-attention) and an MLP with a hidden dimension of 256. All the layers use LayerNorm [2], ReLU non-linearity and a dropout of 0.3.

Bounding box prediction MLPs. The box prediction MLPs operate on the $B \times 256$ box features from the decoder. We use separate MLPs for the following five predictions - 1) center location offset $\Delta\mathbf{q} \in [0, 1]^3$; 2) angle quantization class; 3) angle quantization residual $\in \mathbb{R}$; 4) box size $\mathbf{s} \in [0, 1]^3$; 5) semantic class of the object. Each MLP has 256 hidden dimensions and uses the ReLU non-linearity. The center location and size prediction MLP outputs are followed by a sigmoid function to convert them into a $[0, 1]$ range.

Inference speed. 3DETR has very few 3D-specific tweaks and uses standard PyTorch. VoteNet relies on custom GPU CUDA kernels for 3D operations. We measured the inference time of 3DETR (256 queries) and VoteNet (256 boxes) on a V100 GPU with a batchsize of 8 samples. Both models downsample the pointcloud to 2048 points. 3DETR needs 170 ms while VoteNet needs 132 ms. As research into efficient self-attention becomes more mature (several recent works show promise), it will benefit the runtime and memory efficiency of our model.

Encoder Layers	Decoder Layers	Inference time
3	3	153
3	6	164
3	8	170
3	10	180
6	6	193
6	8	213
8	8	219

Table 7: Inference Speed and Memory. We provide inference speed (in milliseconds) for different number of encoder and decoder layers in the 3DETR model. All timings are measured on a single V100 GPU with a batchsize of 8 and using 256 queries.

A.2. Set Loss

The set matching cost is defined as:

$$C_{\text{match}}(\hat{\mathbf{b}}, \mathbf{b}) = \underbrace{-\lambda_1 \text{GIoU}(\hat{\mathbf{b}}, \mathbf{b}) + \lambda_2 \|\hat{\mathbf{c}} - \mathbf{c}\|_1}_{\text{geometric}} \underbrace{- \lambda_3 \hat{\mathbf{s}}[s_{gt}]}_{\text{semantic}}$$

For B predicted boxes and G ground truth boxes, we compute a $B \times G$ matrix of costs by using the above pairwise cost term. We then compute an optimal assignment between each ground truth box and predicted box using the Hungarian algorithm. Since the number of predicted boxes is larger than the number of ground truth boxes, the remainder $B-G$ boxes are considered to match to background. We set $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ as $2, 1, 0, 0$ for ScanNetV2 and $3, 5, 1, 5$ for SUN RGB-D.

For each predicted box that is matched to a ground truth box, our loss function is:

$$\begin{aligned}\mathcal{L}_{\text{3DETR}} = 5 * \|\hat{\mathbf{c}} - \mathbf{c}\|_1 + \|\hat{\mathbf{d}} - \mathbf{d}\|_1 + \|\hat{\mathbf{a}}_r - \mathbf{a}_r\|_{\text{huber}} \\ - 0.1 * \mathbf{a}_c^\intercal \log \hat{\mathbf{a}}_c - 5 * \mathbf{s}_c^\intercal \log \hat{\mathbf{s}}_c\end{aligned}$$

For each unmatched box that is considered background, we compute only the semantic loss term. The semantic loss is implemented as a weighted cross entropy loss with the weight of the 'background' class as 0.2 and a weight of 0.8 for the K object classes.

B. Experiments

We provide additional experimental details and hyperparameter settings.

B.1. Improved baselines

We improve the VoteNet and BoxNet baselines by doing a grid search and improving the optimization hyperparameters. We train the baseline models for 360 epochs using the Adam optimizer [20] with a learning rate of 1×10^{-3} decayed by a factor of 10 after $160, 240, 320$ epochs and a weight decay of 0. We found that using a cosine learning rate schedule, even longer training than 360 epochs or the AdamW optimizer did not make a significant difference in performance for the baselines. These improvements to the baseline lead to an increase in performance, summarized in Table 8.

Method	ScanNetV2		SUN RGB-D	
	AP_{25}	AP_{50}	AP_{25}	AP_{50}
BoxNet [42]	45.4	-	53.0	-
BoxNet† [42]	49.0	21.1	52.4	25.1
VoteNet [42]	58.6	33.5	57.7	-
VoteNet† [42]	60.4	35.5	58.3	33.4

Table 8: Improved baseline. We denote by † our improved implementation of the baseline methods and report the numbers from the original paper [42]. Our improvements ensure that the comparisons in the main paper are fair.

B.2. Per-class Results

We provide the per-class mAP results for ScanNetV2 in Table 10 and SUN RGB-D in Table 9. The overall results for these models were reported in the main paper (Table 1).

B.3. Detailed state-of-the-art comparison

We provide a detailed comparison to state-of-the-art detection methods in Table 11. Most state-of-the-art methods build upon VoteNet. H3DNet [89] uses 3D primitives with VoteNet for better localization. HGNet [5] improves VoteNet by using a hierarchical graph network with higher resolution output from its PointNet++ backbone. 3D-MPA [11] uses clustering based geometric aggregation and graph convolutions on top of the VoteNet method. 3DETR does not use Voting and has fewer 3D specific decisions compared to all other methods. 3DETR performs favorably compared to these methods and outperforms VoteNet. This suggests that, like VoteNet, 3DETR can be used as a building block for future 3D detection methods.

B.4. 3DETR-m with Vote loss

We tuned the VoteNet loss with the 3DETR-m encoder and our best tuned model gave 60.7% and 56.1% mAP on ScanNetV2 and SUN RGB-D respectively (settings from Table 3 of the main paper). The VoteNet loss performs better with 3DETR-m compared to the vanilla 3DETR encoder (gain of 6% and 3%), confirming that the VoteNet loss is dependent on the inductive biases/design of the encoder. Using our set loss is still better than using the VoteNet loss for 3DETR-m (Table 1 *vs*. results stated in this paragraph). Thus, our set loss design decisions are more broadly applicable than that of VoteNet.

Model	bed	table	sofa	chair	toilet	desk	dresser	nightstand	bookshelf	bathtub
3DETR	81.8	50.0	58.3	68.0	90.3	28.7	28.6	56.6	27.5	77.6
3DETR-m	84.6	52.6	65.3	72.4	91.0	34.3	29.6	61.4	28.5	69.8

Table 9: Per-class AP_{25} for SUN RGB-D.

Model	cabinet	bed	chair	sofa	table	door	window	bookshelf	picture	counter	desk	curtain	refrigerator	showercurtrain	toilet	sink	bathtub	garbagebin
3DETR	50.2	87.0	86.0	87.1	61.6	46.6	40.1	54.5	9.1	62.8	69.5	48.4	50.9	68.4	97.9	67.6	85.9	45.8
3DETR-m	49.4	83.6	90.9	89.8	67.6	52.4	39.6	56.4	15.2	55.9	79.2	58.3	57.6	67.6	97.2	70.6	92.2	53.0

Table 10: Per-class AP_{25} for ScanNetV2.

Method	**Arch.**	**ScanNetV2**		**SUN RGB-D**	
		AP_{25}	AP_{50}	AP_{25}	AP_{50}
BoxNet† [42]	BoxNet	49.0	21.1	52.4	25.1
3DETR	Tx.	62.7	37.5	56.8	30.1
VoteNet† [42]	VoteNet	60.4	37.5	58.3	33.4
3DETR-m	Tx.	65.0	47.0	59.0	32.7
H3DNet [89]	VoteNet + 3D primitives	67.2	48.1	60.1	39.0
HGNet [5]	VoteNet + GraphConv	61.3	34.4	61.6	34.4
3D-MPA [11]	VoteNet + GraphConv	64.2	49.2	-	-

Table 11: Detailed state-of-the-art comparison on 3D detection.

B.5. Adapt queries at test time

We provide additional results for Section 5.1 of the main paper. We change the number of queries used at test time for the same 3DETR model. We show these results in Fig 7 for two different 3DETR models trained with 64 and 256 queries respectively. We observe that the model trained with 64 queries is more robust to changing queries at test-time, but at its most optimal setting achieves worse detection performance than the model trained with 256 queries. In the main paper, we show results of changing queries at test time for a model trained with 128 queries that achieves a good balance between overall performance and robustness to change at test-time.

B.6. Visualizing the encoder attention

We visualize the encoder attention for a 3DETR model trained on the SUN RGB-D dataset in Fig 8. The encoder focuses on parts of objects.

B.7. Shape Classification setup

Dataset and Metrics. We use the processed point clouds with normals from [45], and sample 8192 points as input for both training and testing our models. Following prior work [90], we report two metrics to evaluate shape classification performance: 1) Overall Accuracy (OA) evaluates how many point clouds we classify correctly; and 2) Class-Mean Accuracy (mAcc) evaluates the accuracy for each class independently, followed by an average over the per-class accuracy. This metric ensures tail classes contribute equally to the final performance.

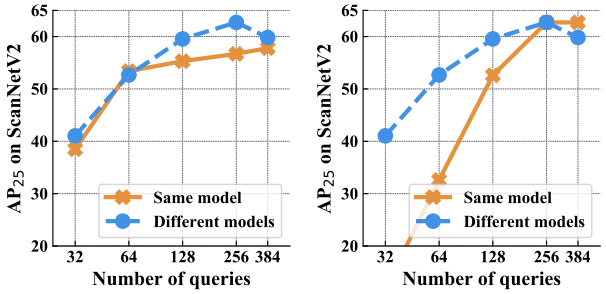

Figure 7: Adapt queries at test time. Similar to Figure 5 of the main paper, we change the number of queries at test time for a 3DETR model and compare it to different models trained with a varying number of queries. We plot the results for the same 3DETR model trained with 64 queries (left) or with 256 queries (right).

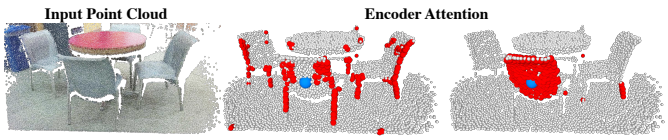

Figure 8: Encoder attention. We visualize the encoder attention for two different heads. We compute the self-attention from the reference point (blue dot) to all the points in the scene and display the points with the highest attention values in red. The encoder groups together different geometric parts (legs of multiple chairs) or focuses on single parts of an instance (backrest of a chair).

Architecture Details. We use the base 3DETR and 3DETR-m encoder architectures, followed by a 2-layer MLP with batch norm and a 0.5 dropout to transform the final features into a distribution over the 40 predefined shape classes. Differently from object detection experiments, our point features include the 3D position information concatenated with 3D normal information at each point, and hence the first linear layer is correspondingly larger, though the rest of the network follows the same architecture as the encoder used for detection. For the experiments with 3DETR, we prepend a `[CLS]` token, output of which is used as input to the classification MLP. For the experiments with 3DETR-m that involve masked transformers, we simply max pool

the final layer features, which are then passed into the classifier.

Training Details. All models are trained for 250 epochs with a learning rate (LR) of 4×10^{-4} and a weight decay of 0.1, using the AdamW optimizer. We use a linear warmup of LR from 4×10^{-7} to the initial LR over 20 epochs, and then decay to 4×10^{-5} over the remaining 230 epochs. The models are trained on 4 GPUs with a batch size of 2 per GPU.