[Supplementary] UnO: Unsupervised Occupancy Perception and Forecasting

Ben Agro*, Quinlan Sykora*, Sergio Casas*, Thomas Gilles, Raquel Urtasun
Waabi, University of Toronto
{bagro, qsykora, sergio, tgilles, urtasun}@waabi.ai

In this appendix, we describe implementation details relevant to UNO including architecture and training, implementation details for any baselines and ablations, and further details on metrics and evaluation procedures. Finally, we showcase several additional results:

- Visualizations comparing the 4D occupancy and point cloud forecasts of UNO against various baselines including the SOTA 4D-OCC.
- Visualizations of UNO's occupancy in the context of the camera data provided by Argoverse 2.
- Comparing the 4D occupancy recall of UNO, UNO fine-tuned with bounding box data, and UNO trained with bounding box data without pre-training for 4D occupancy.

1. UNO Implementation Details

1.1. LiDAR Encoder

The LiDAR encoder takes H past LiDAR sweeps to produce a Birds Eye View (BEV) feature map $\mathbf{Z}$ used by the implicit decoder of UNO. Below we describe the architecture of the LiDAR encoder, illustrated in Fig. 1, where we omit the batch dimension for simplicity.

Voxelization: Each LiDAR sweep is a set of points with four features (x, y, z, t). A small MLP is used encode these points as feature vectors of size $F = 128$. The LiDAR points are placed in a 2D BEV grid of shape (L, W) based on their (x, y) coordinates. We use an ROI of $[-100, 150]$ m on the x dimension, $[-100, 100]$ m on the y dimension with a voxel size of $0.156\,25\,\mathrm{m}$ in x, y, resulting in $L = \frac{200}{0.15265} = 1280, W = \frac{250}{0.1526} = 1600$. A 2D feature map of shape (F, L, W) is then generated from this BEV grid, where each grid cell has a feature vector that is the sum of the features from all points in that grid cell.

Backbone: This 2D feature map is processed by three convolution layers interleaved with batch-normalization and ReLU layers, the first convolutional layer having a stride of 2, resulting in a feature map of shape $(F, L/2, W/2)$.

Then, this feature map is processed by a series of ten residual layers, labelled *ResBlock* in Fig. 1, which each employ a sequence of dynamic convolution [3], batch-normalization, ReLU, dynamic convolution, batch-normalization, squeeze-and-excitation [4], and dropout operations. Each residual layer produces a feature map, and layers 0, 2, and 4 down-sample their output by a factor of 2 (using convolutions with stride 2).

We extract three multi-level feature maps from the output of layers 1, 3, and 9 with shapes $(F, L/4, W/4)$, $(F, L/8, W/8)$, and $(F, L/16, W/16)$ respectively, across which information is fused with multiscale deformable attention [9] to produce three feature maps with the same shapes as the input.

Feature Pyramid Network These three multi-level feature maps are fused into a single feature map $\mathbf{Z}$ of shape $(F, L/4, W/4)$ by a lightweight Feature Pyramid Network [7]. $\mathbf{Z}$ is used by the implicit decoder, described below, to predict 4D geometric occupancy.

*Denotes equal contribution

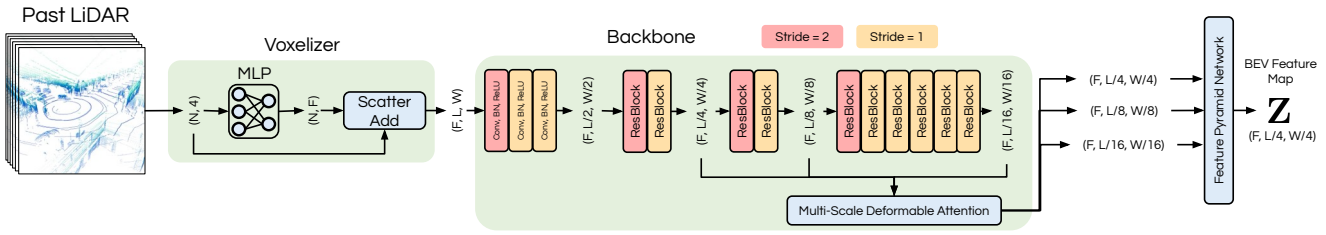

Figure 1. The architecture of the LiDAR Encoder. The batch dimension is omitted from the tensor shapes.

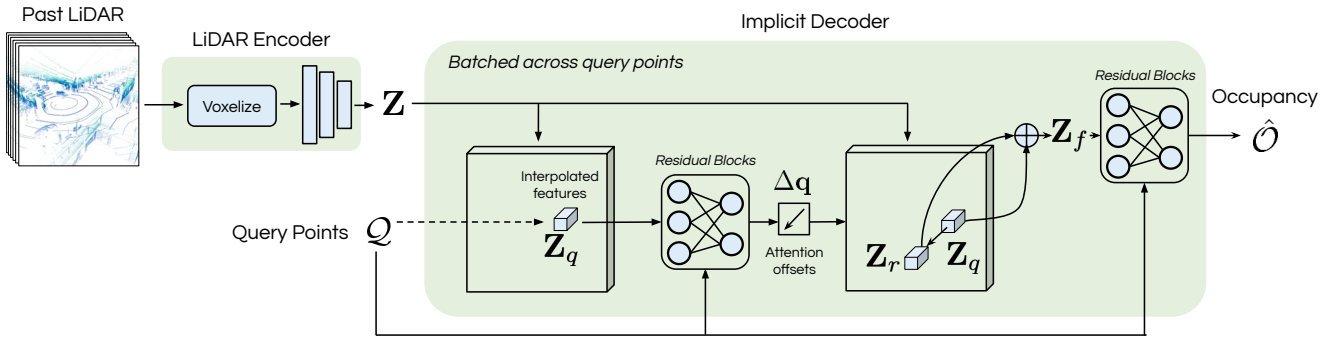

Figure 2. UNO's architecture zoomed in on the implicit decoder.

1.2. Implicit Occupancy Decoder

The inputs to the implicit occupancy decoder are the feature map $\mathbf{Z} \in \mathbb{R}^{F \times \frac{L}{4} \times \frac{W}{4}}$ from the lidar encoder, and a set of 4-dimensional query points $\mathcal{Q} \in \mathbb{R}^{|\mathcal{Q}| \times 4}$ with features (x, y, z, t). Our decoder follows closely that of ImplicitO [1], but we describe it below. See Fig. 2 for an architecture diagram. The decoder consists of three main parts: offset prediction, feature aggregation, and occupancy prediction.

Offset Prediction: The offset prediction module begins by interpolating $\mathbf{Z}$ at the (x, y) locations specified by the spatial components of the query points $\mathbf{q}_{x,y} \in \mathcal{Q}_{x,y}$ to produce interpolated feature vectors $\mathbf{z}_q$ that contain information about the scene around each query point. Next, linear projections of $\mathbf{Z}_q$ and $\mathbf{q}$ are added and passed through a residual layer consisting of two fully connected linear layers and a residual connection. We use a dimensionality of 16 for these linear projections and the hidden size of the linear layers in the residual layers. Finally, a linear layer is used to produce an offset per query point $\Delta\mathbf{q}$. These offsets are added to the query points to find the offset sample locations $\mathbf{r} = \mathbf{q}_{x,y} + \Delta\mathbf{q}_{x,y}$, which is the input to the next module.

Feature Aggregation: The offset sample points $\mathbf{r}$ are meant to store the (x', y') locations in $\mathbf{Z}$ that contain important information for occupancy prediction at query point (x, y, t). As such, the feature aggregation module begins by interpolating $\mathbf{Z}$ at $\mathbf{r}$ to obtain a feature vector $\mathbf{Z}_r$, for each query point. This feature vector is concatenated with $\mathbf{z}_q$ to obtain the summary features for the query point, denoted $\mathbf{Z}_f$, of dimensionality $2F = 256$.

Occupancy Prediction: This module uses the aggregated feature vector $\mathbf{Z}_f$ and the query points $\mathbf{q}$ to predict occupancy logits, $\hat{\mathcal{O}}$, for all query points. $\mathbf{Z}_f$ and $\mathbf{q} = (x, y, z, t)$ are passed through three residual blocks with an architecture inspired by Convolutional Occupancy Networks [8]. The residual block takes as input a linear projection of $\mathbf{q} = (x, y, z, t)$. Each block adds its input to a linear projection of $\mathbf{Z}_f$, and then passes this through a residual layer using a hidden size 16. The input to the subsequent block is the output of the previous block.

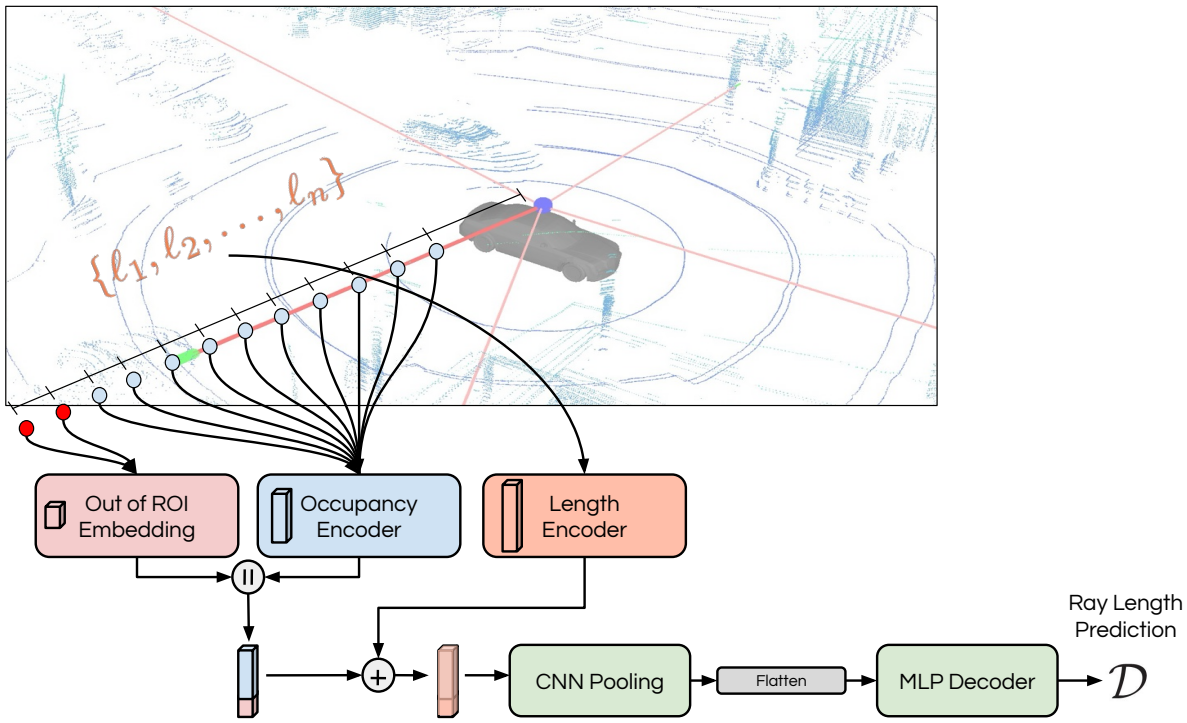

Figure 3. An overview of our the learned render header. The predicted occupancy at each point along the ray is combined with the distance of the point from the sensor, after which the features are reduced using a 1D CNN and then passed through an MLP to produce a single scalar prediction. || denotes lengthwise concatenation, and + denotes addition.

1.3. Rendering Header

The rendering header is the small network that uses the occupancy predictions of UNO along a lidar ray and estimates the depth of the lidar point along the ray. Here we provide more details on the architecture and training of this header. A diagram is in Fig. 3. Given a query lidar ray, we generate $N_r = 2000$ query points along the ray every $\epsilon_r = 0.1\,\text{m}$. The query points inside UNO's ROI are fed to UNO, which predicts occupancy logits in a tensor of shape $N_{r_{in}} \times 1$ where $N_{r_{in}}$ is the number of points inside the ROI. The occupancy logits are encoded by a linear layer ("Occupancy Encoder" in Fig. 3) to produce a tensor of shape $N_{r_{in}} \times 256$. We maintain a learned embedding of shape $N_r \times 256$, and for the $N_{r_{out}} = N_r - N_{r_{in}}$ query points outside the ROI, we index this embedding to create an "Out-of-ROI-Embedding" of shape $N_{r_{out}} \times 256$. The encoded occupancy and out-of-ROI-embedding are concatenated along the length dimension resulting in a tensor of shape $N_r \times 256$. An encoding of the depth of each query point along the ray is added to this tensor, which is processed by a sequence of CNN layers to combine information across the points. Each CNN layer uses a kernel size of 4 and a stride of 2, and they have intermediate dimensions of $64, 32, 16, 16, 16, 8$. ReLU activations are used between the CNN layers. These 1D features are flattened (resulting in a feature vector per ray of dimensionality 232) and passed through an MLP using the ReLU activation function with intermediate dimensions $232, 64, 32, 16, 1$ where the final dimension is the scalar prediction for the depth of the ray $\mathcal{D}$. All the lidar rays are used for training. We randomly sample a batch of 450 lidar rays (90 for each future lidar sweep) at each training step.

1.4. Training Details

We train UNO for a total of 800,000 iterations across 16 GPUs (50,000 iterations each) with a batch size of 1 on each GPU.

Optimizer: We warmup the learning rate at a constant rate for the first 1000 iterations beginning from 8.0×10^{-5} and ending at 8.0×10^{-4}. Then we use cosine schedule configured to bring the learning rate to 0 by 50,000 iterations. We use the AdamW optimizer [6] with a weight decay of 1.0×10^{-4}.

Initialization: We follow [1] in their initialization procedure for the linear layer that predicts the attention offsets in the implicit decoder: we initialize the weights with a small standard deviation of 0.01 and a bias of 0. This ensures that the initial offset prediction is small enough such that $\mathbf{r} = \mathbf{q}_{x,y} + \Delta\mathbf{q}_{x,y}$ still represents a point within the feature map $\mathbf{Z}$, but large enough such that the predicted offsets do not get stuck at $\Delta\mathbf{q}_{x,y} = \mathbf{0}$.

Training Data: For point cloud forecasting, we present results of UNO on Argoverse 2, NuScenes, and KITTI. For each evaluation setting, we train a version of UNO on the corresponding training dataset split (for Argoverse 2 we use the Sensor dataset training split).

For BEV semantic occupancy forecasting, we train all models on the vehicle labels provided by the Argoverse 2 Sensor dataset. We define the "vehicle class" to be the union of the Argoverse classes `REGULAR_VEHICLE`, `LARGE_VEHICLE`, `BUS`, `BOX_TRUCK`, `TRUCK`, `VEHICULAR_TRAILER`, `TRUCK_CAB`, `SCHOOL_BUS`, `ARTICULATED_BUS`, `RAILED_VEHICLE`, and `MESSAGE_BOARD_TRAILER`.

2. Evaluation Details

2.1. Point Cloud Forecasting

For point-cloud forecasting, we follow the evaluation procedure of 4D-OCC [5]. The code is available online[1], but we explain it in this section.

Setting: On Argoverse 2 and KITTI, $H = 5$ past LiDAR sweeps are used as input at an interval of $0.6\,\text{s}$, and the goal is to forecast point clouds at 5 future timesteps $\{0.6, 1.2, \ldots, 3.0\}$ s. On NuScenes we use $H = 6$ sweeps at an interval of $0.5\,\text{s}$ as input, and forecast point clouds at 6 future timesteps $\{0.5, 1.0, \ldots, 3.0\}$ s. Evaluation for Table 1. in the main paper are on the following dataset splits:

- AV2: Argoverse 2 LiDAR dataset test split
- NuScenes: validation set[2]
- KITTI: test set

We note that the Argoverse 2 point cloud forecasting challenge and the point-cloud forecasting results in Table. 2 of the main paper use the Argoverse 2 Sensor dataset test split.

Metrics: Point cloud evaluation has four metrics: Chamfer Distance (CD), Near Field Chamfer Distance (NFCD), depth L1 error, and depth relative L1 error (AbsRel), each of which we explain below.

CD is computed as:

$$CD = \frac{1}{2N}\sum_{\mathbf{x}\in\mathbf{X}}\min_{\hat{\mathbf{x}}\in\hat{\mathbf{X}}}||\mathbf{x}-\hat{\mathbf{x}}||_2^2 + \frac{1}{2M}\sum_{\hat{\mathbf{x}}\in\hat{\mathbf{X}}}\min_{\mathbf{x}\in\mathbf{X}}||\mathbf{x}-\hat{\mathbf{x}}||_2^2, \tag{1}$$

where $\mathbf{X} \in \mathbb{R}^{N\times 3}$, $\hat{\mathbf{X}} \in \mathbb{R}^{M\times 3}$ represent the ground-truth and predicted point clouds, respectively. NFCD is CD computed only on (ground truth and predicted) points inside the ROI of $[-70, 70]$ m in both the x and y axes, and $[-4.5, 4.5]$ m in the z axis around the ego vehicle.

L1 and AbsRel require the ground truth point-cloud $\mathbf{X}$ and the predicted point cloud $\hat{\mathbf{X}}$ to have a 1-1 correspondence of ray directions. While for UNO this is always the case because we predict depth along the rays from the future ground truth point clouds, for some point-cloud forecasting methods this condition might not be met. Thus, for fair evaluation, [5] first fits a surface to the predicted point cloud, and the intersection of the ground truth ray and that surface is computed to find the predicted ray depth. In practice, this is done by computing the projection of $\mathbf{X}$ and $\hat{\mathbf{X}}$ on the unit sphere, for each projected ground truth point finding the nearest projected forecasted point, and then setting the depth along that ground truth ray equal to the depth of its nearest forecasted point. The result of this is a vector of forecasted depths along each LiDAR ray $\hat{\mathbf{D}} \in \mathbb{R}^N$ and ground truth depths $\mathbf{D} \in \mathbb{R}^N$. Then L1 is calculated as

$$\text{L1} = \texttt{mean}(\texttt{abs}(\mathbf{D} - \hat{\mathbf{D}})) \tag{2}$$

[1] https://github.com/tarashakhurana/4d-occ-forecasting

[2] See https://github.com/tarashakhurana/4d-occ-forecasting/issues/8 for an explaination as to why the test set is not used.

and

$$\text{AbsRel} = \texttt{mean}(\texttt{abs}((\mathbf{D} - \hat{\mathbf{D}})/\mathbf{D})), \tag{3}$$

where the vector division is element-wise.

2.2. BEV Semantic Occupancy Forecasting

Setting: We evaluate BEV semantic occupancy on the Argoverse 2 Sensor validation dataset. The LiDAR input is $H = 5$ past LiDAR sweeps at an interval of $0.6\,\text{s}$ to match the unsupervised pre-training of UNO, and the BEV semantic occupancy predictions are evaluated on 2D grids of points of size $140\,\text{m}$ by $140\,\text{m}$ with a spatial resolution of $0.2\,\text{m}$ centered on the ego at future timesteps $\{0, 0.6, 1.2, \ldots, 3.0\}$ s.

Metric: Mean Average Prediction (mAP) is computed on (a) the predicted occupancy values from all the grid points across all evaluation examples, and (b) the corresponding ground truth occupancy values at those points. For thresholds $[0, 0.01, 0.02, \ldots, 0.99, 1.0]$, the number of true positives (TP), false positives (FP), and false negatives (FN), are calculated. At each threshold value, we obtain a precision and recall

$$\text{precision} = \frac{TP}{TP + FP}, \text{ recall} = \frac{TP}{TP + FN}, \tag{4}$$

and mAP is then computed as the area under the curve formed by precision (y-axis) and recall (x-axis) across all thresholds.

3. Baseline Implementation Details

FREE-SPACE-RENDERING: We use same model architecture as UNO, described in Sec. 1. Mirroring the training of the learned renderer (see Sec. 1.3), we sample 450 future lidar rays within the $3\,\text{s}$ time horizon. Considering a single lidar ray, we generate $N_r = 2000$ evenly spaced query points along the lidar ray at a step size of $\epsilon_r = 0.1\,\text{m}$ from the emitting sensor. The model is queried at these points to produce occupancy predictions $\hat{\mathbf{o}} = [\hat{o}_1, \ldots, \hat{o}_{N_r}] \in [0, 1]^{N_r}$ along each the ray. Following [5], we complement the cumulative maximum of the occupancy along the LiDAR ray to obtain a prediction of the free-space:

$$\hat{\mathbf{v}} = 1 - \texttt{cummax}(\hat{\mathbf{o}}). \tag{5}$$

Using the ground-truth future lidar point we can generate a free-space label $\mathbf{v} = [v_1, \ldots, v_{N_r}] \in \{0, 1\}^{N_r}$ which is 1 for all points along the ray prior to the LiDAR point and 0 for all points along the ray after the LiDAR point. Then the model is trained with cross entropy loss between the free-space label and the free-space prediction.

DEPTH-RENDERING: We use the same model architecture as UNO, described in Sec. 1. Similarly to FREE-SPACE-RENDERING, we sample future lidar rays and query the model along each lidar ray to produce occupancy predictions $\hat{\mathbf{o}} = [\hat{o}_1, \ldots, \hat{o}_{N_r}] \in [0, 1]^{N_r}$. The expeceted depth along the ray is computed as

$$\mathbb{E}[\ell] = \sum_{i=0}^{N_r} (\epsilon_r * i) \left(\left(\prod_{j=0}^{i-1} (1 - \hat{o}_j) \right) \hat{o}_i \right), \tag{6}$$

and the loss against the ground truth LiDAR point depth ℓ_{gt} is computed as $|\ell_{gt} - \mathbb{E}[\ell]|$, averaged across all rays in the batch.

MP3 We use the same encoder as UNO for a fair comparison (see Sec. 1). MP3 decodes initial occupancy and forward flow over time using a fully convolutional header, and the future occupancy is obtained by warping the initial occupancy with the temporal flow as described in the original paper [2].

ImplicitO We use the same model architecture as UNO (which is very similar to the orignal paper [1], see Sec. 1), but the query point feature size is 3 (x, y, t) because they are in BEV instead of 3D.

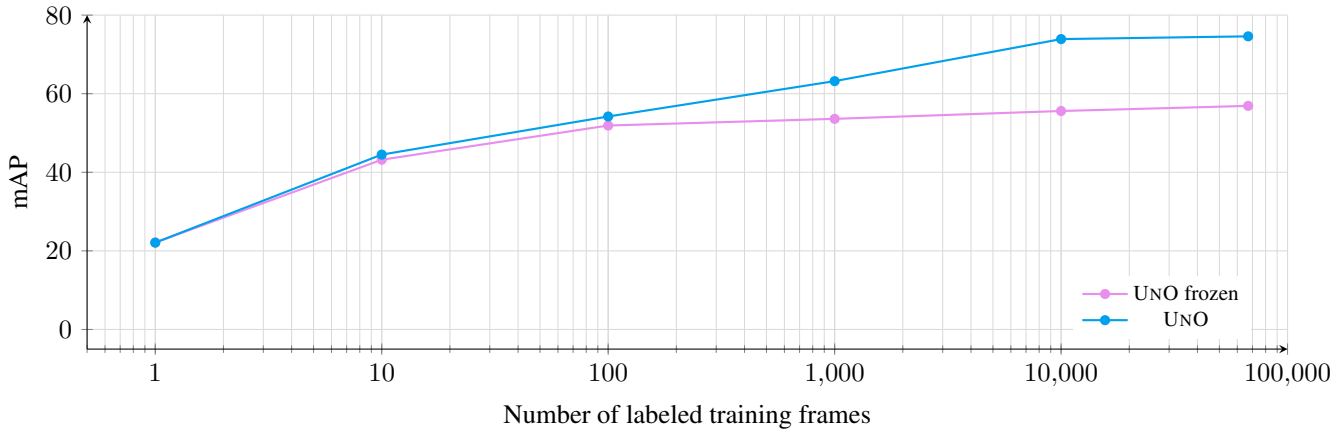

Figure 4. BEV semantic occupancy results comparing UNO Frozen, which fine-tunes only the occupancy decoder, to UNO, which fine-tunes all parameters.

Rendering	NFCD (m^2) ↓	CD (m^2) ↓	AbsRel (%) ↓	L1 (m^2) ↓
Threshold 0.5	3.09	22.75	13.34	3.80
Threshold 0.8	2.09	18.49	11.33	3.01
Threshold 0.9	1.63	18.16	11.16	2.67
Threshold 0.95	1.48	23.02	14.90	2.91
Learned	**0.83**	**8.10**	**10.09**	**2.03**

Table 1. Comparing point cloud prediction performance of UNO with different rendering methods (thresholding occupancy and learned rendering) on the Argoverse 2 Sensor dataset.

4. Additional Quantitative Results

4.1. Effect of Freezing the Encoder During Fine-tuning

In this experiment, we want to investigate how fine-tuning UNO on BEV semantic occupancy is affected by amount of model parameters that are updated. To do this, we fine-tune UNO on the task of BEV semantic occupancy forecasting, but freeze all the parameters in the LiDAR encoder (see Sec. 1), which we call UNO Frozen. Fig. 4 presents the results across multiple levels of supervision, against UNO with all parameters fine-tuned. We notice that for up to 100 labelled frames, the performance of UNO Frozen is close to UNO, but their performance diverges as the amount of labelled data increases. This is likely because fine-tuning only the occupancy decoder, which has very few parameters (0.06M out of a total of 17.4M), has limited representational power in changing the output fully from geometric occupancy (which it was pre-trained for) to BEV semantic occupancy.

4.2. Effect of Learned Renderer:

In the main paper we described that generating point cloud forecasts from occupancy is a non-trivial task, motivating our use of the learned renderer. In this experiment, we compare it to a simple baseline that thresholds the forecasted occupancy values. Specifically, to find the depth of a ray, we walk along the ray direction starting from the sensor location, and stop when the occupancy value exceeds a given threshold. Tab. 1 presents the results of this comparison on the Argoverse 2 Sensor dataset, where we see large improvements from using the learned rendering method over the thresholding method with any threshold. This is expected since the learned renderer can better contextualize the scene by taking into account a set of occupancy values predicted along the ray.

4.3. Supervised vs Unsupervised Occupancy

In this section we compare the unsupervised 4D geometric occupancy predictions to supervised 4D semantic occupancy predictions (trained using bounding box labels). To do this fairly across geometric and semantic occupancy predictions, we employ our recall metric described in the main paper, which ignores background regions (which geometric occupancy includes, but semantic occupancy does not) and focuses on foreground actors and free-space (which both forms of occupancy

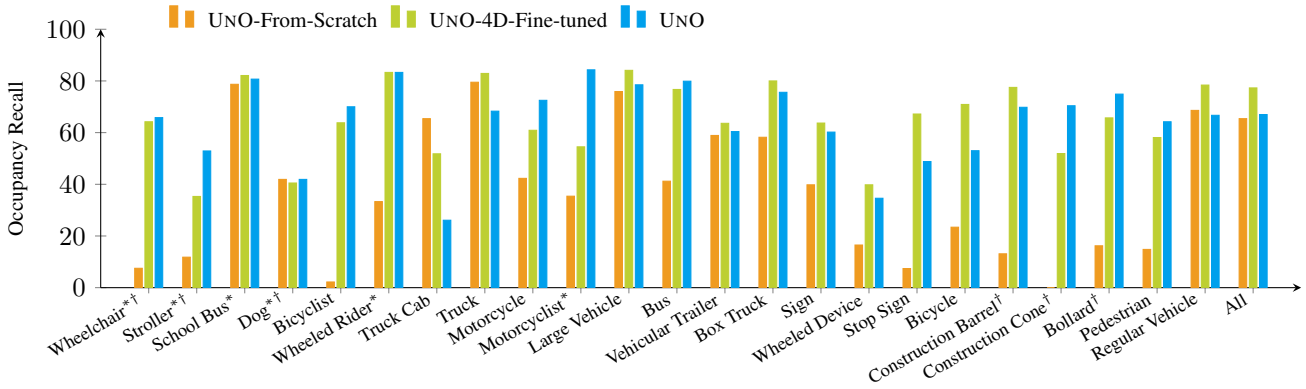

Figure 5. Recall of UNO compared to other supervised and semi-supervised baselines on the Argoverse 2 Sensor dataset, averaged across the prediction horizon. Recall was computed at a target precision of 0.7. * denotes the rarest 25% of classes, and † denotes the smallest (by average bounding box volume) 25% of classes.

	Point Cloud Forecasting				**BEV Semantic Occupancy**		**4D Occupancy**
δ (m)	NFCD (m^2) ↓	CD (m^2) ↓	AbsRel (%) ↓	L1 (m^2) ↓	mAP ↑	Soft-IoU ↑	Recall @ 0.7
1.00	0.85	8.22	11.5	2.08	50.3	21.1	65.7
0.01	**0.82**	**8.06**	10.8	2.04	52.0	22.2	65.8
0.10	0.83	8.10	**10.1**	**2.03**	**52.3**	**22.3**	**67.0**

Figure 6. Ablation of the δ parameter used in the training of UNO.

should understand).

In Fig. 5, we evaluate the multi-class 3D occupancy recall of:

1. UNO: Unsupervised 4D geometric occupancy pre-training only (described in main paper).
2. UNO-From-Scratch: Using the UNO architecture, but with no unsupervised pre-training. Instead, this model is directly trained only on labeled bounding box data to forecast 4D semantic occupancy. We group all actor classes into a single class.
3. UNO-4D-Fine-Tuned: UNO with 4D geometric occupancy pre-training and additional fine-tuning to forecast 4D semantic occupancy (grouping all actors into a single class).

For this fine-tuning process, we train both the supervised and the fine-tuned models with all available data.

Remarkably, UNO (which uses no labeled data) achieves overall better recall than UNO-From-Scratch. We hypothesize this is because the supervised model has to learn to classify between foreground and background objects, which can make it hard to learn the occupancy of rare classes, small classes, or classes that are hard to distinguish from the background. This is highly relevant to safe autonomous driving for detecting potentially unknown or rarely seen road agents.

UNO-4D-Fine-Tuned slightly improves recall (see the "All" class) over UNO and greatly improves over UNO-From-Scratch. This highlights the expressive representations learned during the UNO pre-training procedure.

4.4. Ablation of the δ Parameter

$\delta = 0.1\,\mathrm{m}$ was used for all three datasets, however, our pre-training procedure is quite robust to the choice of δ. Below we ablate the choice of δ on all tasks. First, we note that all the possible values below result in SOTA performance. Second, it shows that increasing δ from $0.1\,\mathrm{m}$ to $1.0\,\mathrm{m}$ indeed degrades performance. Qualitatively, we notice very small differences in occupancy forecasts, with a slight enlargement of shapes with $\delta = 1.0\,\mathrm{m}$ that is apparent in the tree trunks.

5. Additional Qualitative Results

5.1. Occupancy and Point Cloud Forecasting

Figs. 7 to 9 show the geometric occupancy forecasts and point cloud forecasts of UNO and various baselines from the main paper. A few overarching themes are:

- 4D-OCC struggles to abstract point clouds into occupancy; it mainly predicts occupancy only where there are observed points.
- 4D-OCC and DEPTH-RENDERING don't model motion and object extent.
- FREE-SPACE-RENDERING understands extent, but does not model motion very accurately.
- UNBALANCED UNO is under-confident (even on static background areas we see relatively low confidence). It has reasonable motion predictions but without much multi-modality, and it is worse at understanding of extent than UNO.
- UNO models motion, extent, and multi-modality.

We use the following numbers to highlight interesting things in Figs. 7 to 9.

1. Struggling to abstract point clouds into occupancy.
2. Inaccurate predictions of extent.
3. Predicting a moving actor stays static.
4. Disappearing occupancy on an actor.
5. Inaccurate motion predictions.
6. Under-confidence on background areas.
7. Under-confidence on actors.
8. Multi-modal predictions.
9. Accurate motion predictions for non-linear actors (e.g., turning, lane-changing).

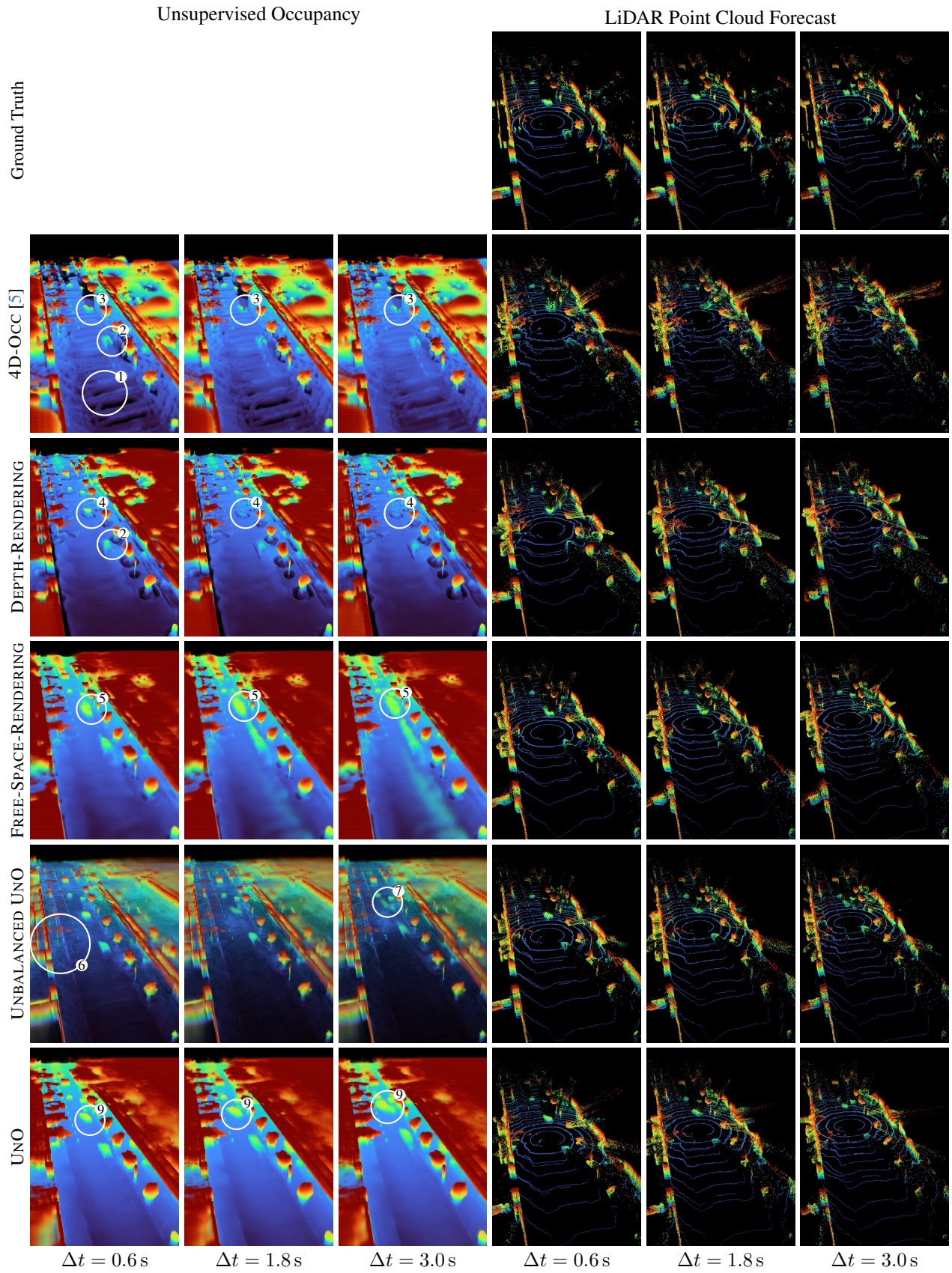

Figure 7. An additional qualitative showing 4D occupancy and point cloud forecasts on Argoverse 2. The numbered labels refer to the themes listed in Sec. 5.1.

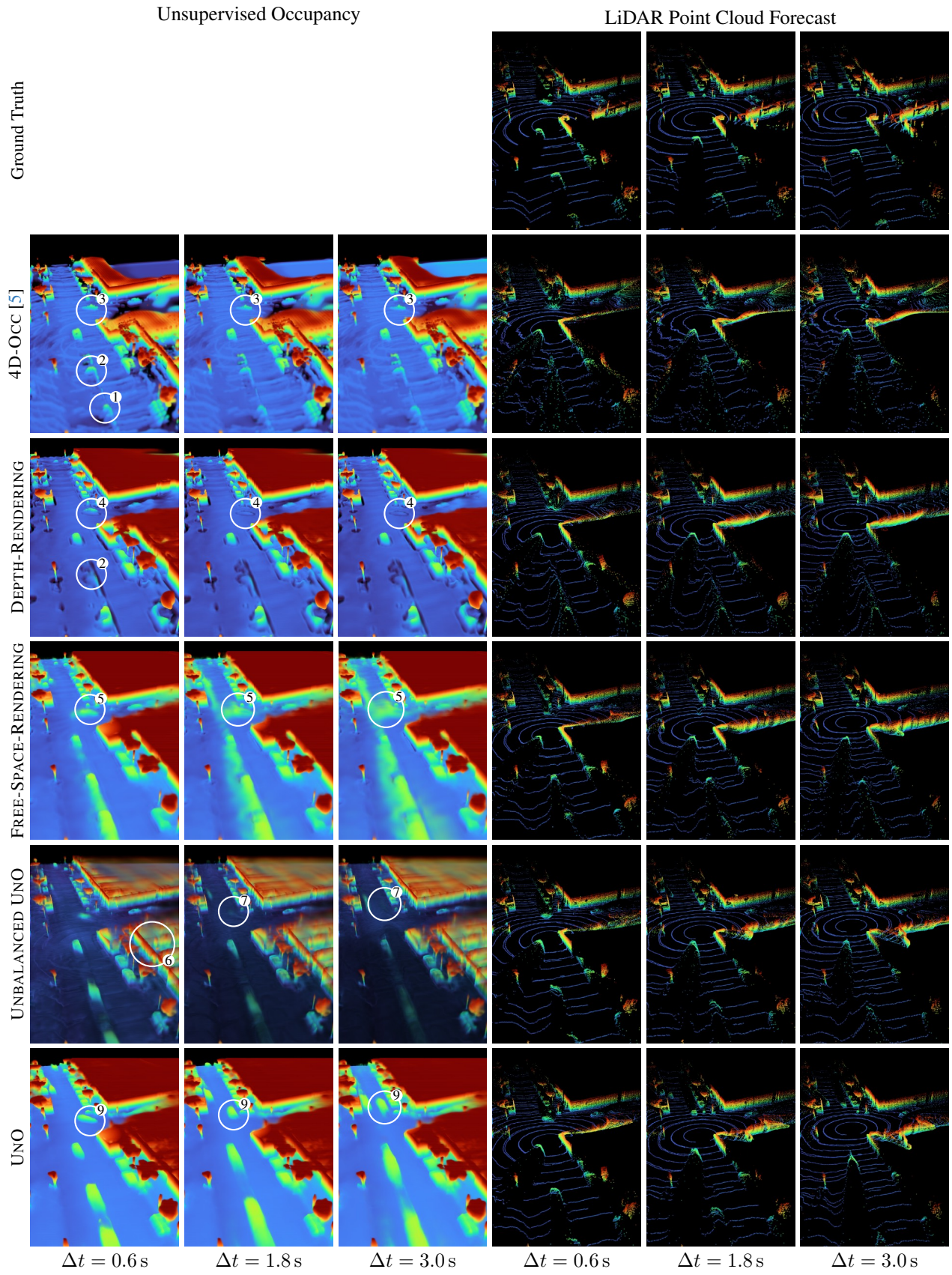

Figure 8. An additional qualitative showing 4D occupancy and point cloud forecasts on Argoverse 2. The numbered labels refer to the themes listed in Sec. 5.1.

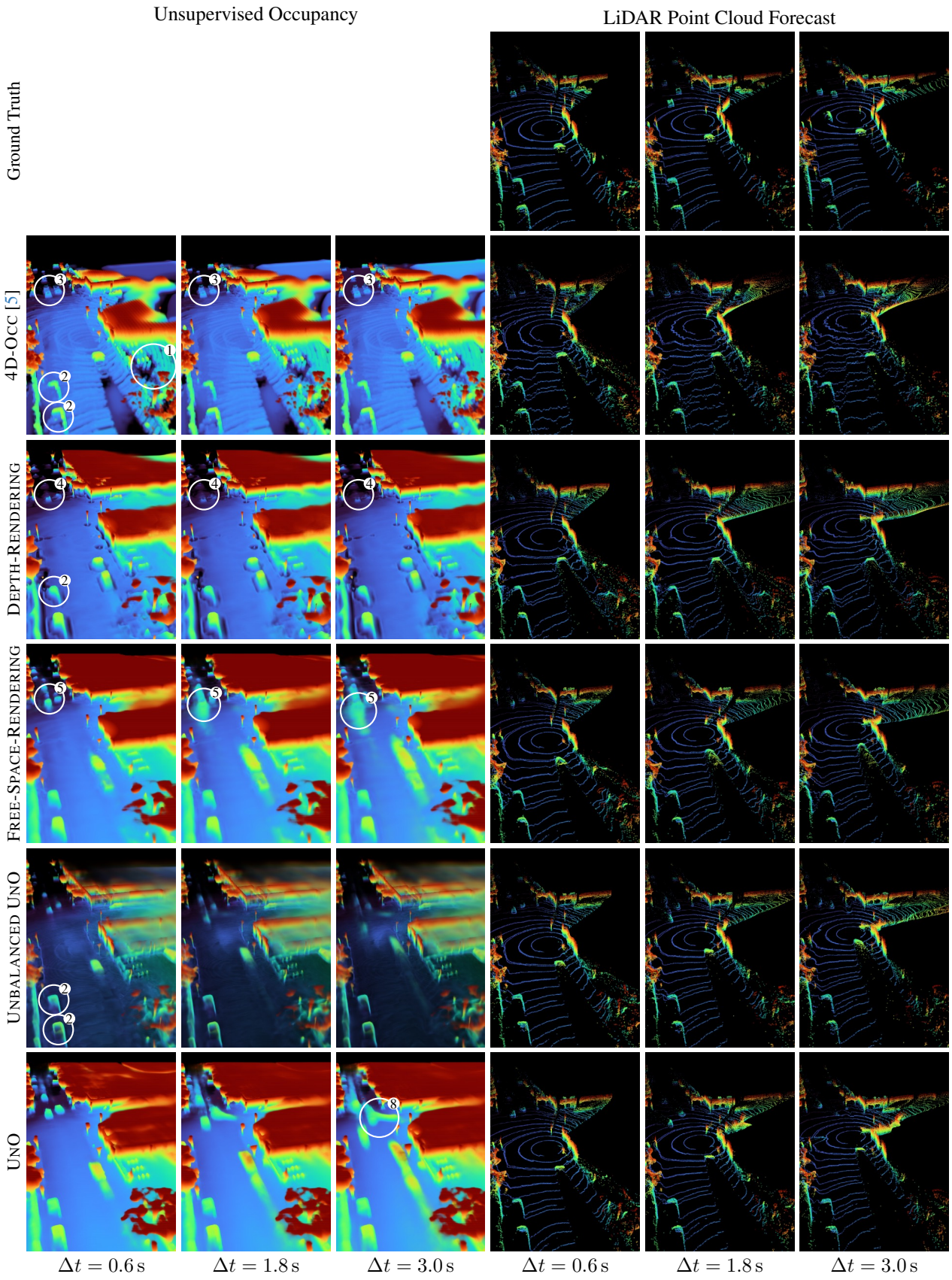

Figure 9. An additional qualitative showing 4D occupancy and point cloud forecasts on Argoverse 2. The numbered labels refer to the themes listed in Sec. 5.1.

5.1.1 Occupancy Perception

To contextualize UNO's occupancy predictions, we visualize them along side camera images provided in Argoverse 2 in Fig. 10. We observe that UNO is able to perceive all objects in the scene, including relatively rare occurrences like construction, over a large region of interest.

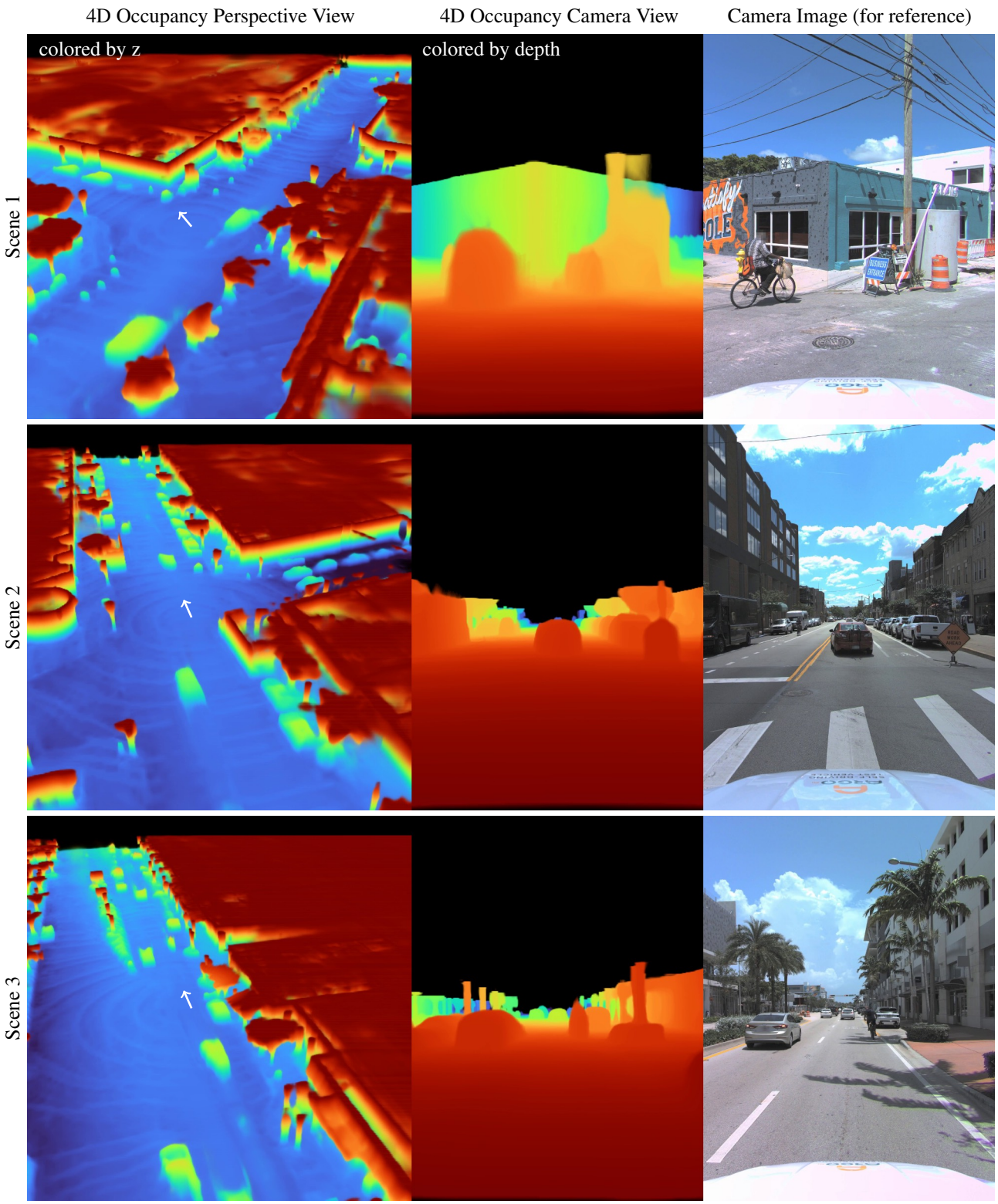

Figure 10. A visualization of UNO's occupancy fields at the present time from both perspective and camera view, next to camera data for reference. The white arrows in perspective view represent the camera viewpoint. **Scene 1**: UNO perceives construction elements and cyclist cross the road. **Scene 2**: UNO perceives a distant cyclist, the construction signage, and many vehicles. **Scene 3**: UNO perceives the traffic island median and the cyclist in the ego lane.

5.1.2 Failure Cases

The main failure cases of UNO are due to the limited range and noise of LiDAR sensors, resulting in limited supervision. Figure 11 shows how this can manifest in uncertain occupancy far from the ego (a), and also at points high above the ego (b) since the LiDAR rays never return from the sky. Figure 10 also shows that there is room for improvement in capturing fine details like the shape of the cyclist in Scene 1. Finally, we note that the long occupancy "tails" behind certain vehicles are not necessarily a failure case. These are expected when predicting marginalized occupancy probabilities on multi-modal scenarios where a vehicle could stay still or accelerate.

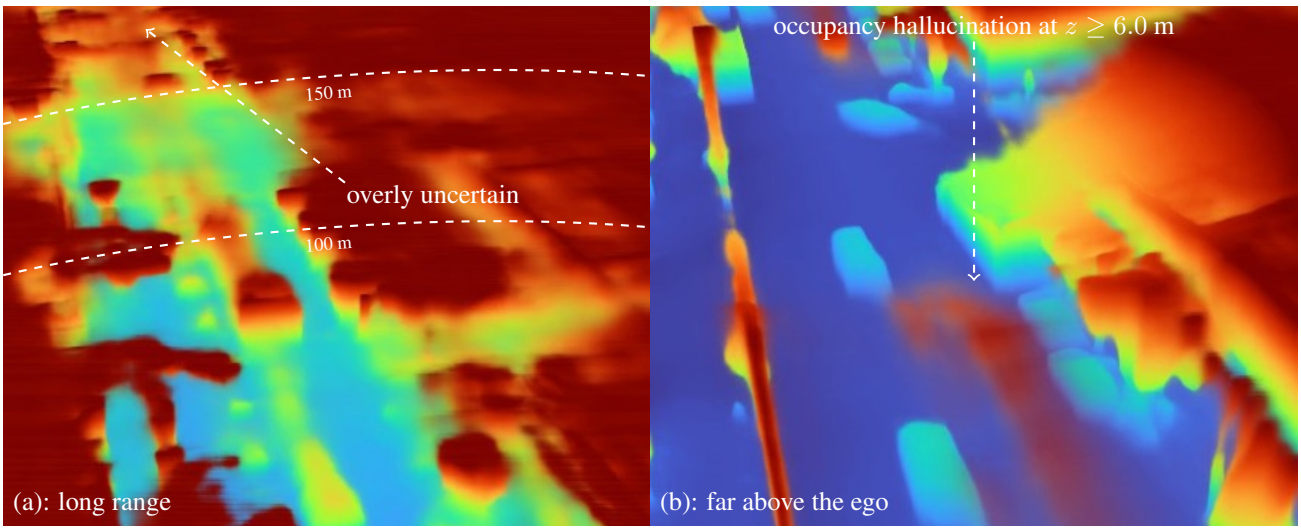

Figure 11. Failure cases UNO.

References

[1] Ben Agro, Quinlan Sykora, Sergio Casas, and Raquel Urtasun. Implicit occupancy flow fields for perception and prediction in self-driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1379–1388, 2023. 2, 4, 5

[2] Sergio Casas, Abbas Sadat, and Raquel Urtasun. Mp3: A unified model to map, perceive, predict and plan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14403–14412, 2021. 5

[3] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels, 2020. 1

[4] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 1

[5] Tarasha Khurana, Peiyun Hu, David Held, and Deva Ramanan. Point cloud forecasting as a proxy for 4d occupancy forecasting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1116–1124, 2023. 4, 5, 9, 10, 11

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3

[7] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 1

[8] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 523–540. Springer, 2020. 2

[9] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv preprint arXiv:2010.04159*, 2020. 1