DIO: Decomposable Implicit 4D Occupancy-Flow World Model

Supplementary Material

This supplementary material includes an extended discussion of related work in Sec. 1. It also provides additional model and implementation details, including architecture, training, baselines, and metrics, as described in Sec. 2. In Sec. 3 we report additional quantitative and qualitative results. We present visualizations of our model, prompted with different source points and comparisons against ablations. Furthermore, we visually compare the scene occupancy predictions with the state-of-the-art, and provide visualizations over multiple frames of a sequence. Please also refer to the video of the supplementary material for additional visualizations. Lastly, Sec. 4 discusses the limitations of our approach and outlines potential directions for future work.

1 Extended Related Work:

Tab. 1 compares DIO's features against those of ImplicitO [1], 4D-Occ [1] and UnO [2]. DIO is the only method using prompts with source points (SP), enabling prediction of scene occupancy, instance occupancy and flow, flexible. Moreoever, it uniquely combines LiDAR and bounding box supervision for instance occupancy, and introduces a new sparse architecture.

DIO is also related to methods that predict scene flow. [3] examines the generalization capabilities of neural scene flow priors (NFSPs) and proposes a self-supervised approach for neural scene flow prediction using multi-frame inputs. Similarly, [4] also goes beyond the two input frame assumption, temporally fuses multiple point clouds after encoding 3D intra-voxel features, and introduces a new 4D architecture for scene flow estimation. [5] proposes a multi-body regularization for NSFPs improving predicted physically implausible motion vectors of prior methods. TrackFlow [6] demonstrates that state-of-the-art scene flow methods struggle to capture the motion of small objects and that existing evaluation protocols fail to account for this limitation. However, none of the mentioned flow methods predicts 4D occupancy, uses an occupancy based flow-consistency loss, or employ prompts as our method.

Method	Scene Occupancy	Background Occupancy	Instance Occupancy	SP	Flow	Supervision	Network
ImplicitO	✓	-	-	-	✓	Box	Dense
4D-Occ	✓	✓	-	-	-	LiDAR	Dense
UnO	✓	✓	-	-	-	LiDAR	Dense
DIO	✓	✓	✓	✓	✓	LiDAR + Box	Sparse

Table 1. Tabular comparison of DIOs features against those of ImplicitO, Occ4D and UnO.

2 Implementation Details

2.1. Architecture

Encoder: We omit the batch dimension in the following to enhance clarity. A LiDAR sweep consists of a set of points with four features, (x, y, z, t). This means that each LiDAR sweep can be represented as a $\mathbf{L}_{\text{sweep}} \in \mathbb{R}^{N\times 4}$ tensor where N is the number of LiDAR points in the given sweep. We leverage the sparse SECOND backbone [7] to encode the point clouds, but modify it to output feature maps at multiple resolutions. This results in *sparse* feature volumes: $\mathbf{V}_{2x} \in \mathbb{R}^{16\times 1280\times 1280\times 151}$, $\mathbf{V}_{4x} \in \mathbb{R}^{32\times 640\times 640\times 76}$, $\mathbf{V}_{8x} \in \mathbb{R}^{64\times 320\times 320\times 38}$, and $\mathbf{V}_{16x} \in \mathbb{R}^{128\times 160\times 160\times 19}$. Note that aside from the feature dimension (the first), these shapes merely reflect the artificial bounds of the sparse tensor, and do not reflect the actual memory requirements. The sparse feature volume $\mathbf{V}_{16x}$ is then densified and then flattened along the z dimension and outputs a dense feature map $\mathbf{M'}_{16x} \in \mathbb{R}^{160\times 160\times 1152}$. The densification process involves filling in the voxels that do have sparse features, and filling in all remaining voxels with zeros. This feature map is passed to a deformable spatial attention neck (DSAN). DSAN then passes these dense feature map $\mathbf{M'}_{16x}$ through a deformable convolutional layer, specifically the pytorch implementation[1] [8]. Getting into the specifics, we first use a linear layer to project the feature size down to 256, then pass it into 4 layers of spatial deformable attention. Each layer has $H = 8$ attention heads each with 4 offsets, a dropout of 0.1 and a feed forward dimension of 1024. After being passed through these layers, which serve to spread the sparse features out, we pass this dense feature map into a series of convolution and transpose convolutional layers. More specifically, we first pass it through 5 convolutional layers with a stride of 2, while retaining the feature dimension of 256, and using ReLU activations. Then we apply another convolution with a stride of 1 with the same activation and output size, and finally put the features through a transpose convolution with a stride size of 2. This finally provides us with our dense feature map $\mathbf{M}_{16x} \in \mathbb{R}^{160\times 160\times 512}$. The output of the encoder are the concatenated scene features at multiple resolutions $\mathcal{Z} = [\mathbf{V}_{2x}, \mathbf{V}_{4x}, \mathbf{V}_{8x}, \mathbf{M}_{16x}]$, which are then processed by the decoder described next.

Decoder: DIO first encodes a query point $\mathbf{q}$ with a sinusiodal embedding giving $\mathbf{e}_q$ of size 64 (16x4) and then uses $\mathbf{q}$ to perform a tri-linear interpolation at x, y, z in the

[1] https://github.com/chengdazhi/Deformable-Convolution-V2-PyTorch/tree/pytorch_1.0.0

sparse feature volumes $\mathbf{V}_{2x}, \mathbf{V}_{4x}, \mathbf{V}_{8x}$ and a bi-linear interpolation in $\mathbf{M}_{16x}$. The concatentation of the interpolation results gives a vector $\mathbf{z}_q$. We perform the same operations for the corresponding source point $\mathbf{s}$ resulting in $\mathbf{z}_s$ with the same size. We use a learnable embedding $\mathbf{e}_s$ with size 64 in case we prompt with an empty source point for scene occupancy. Moreover, we encode both $\mathbf{e}_q$ and $\mathbf{e}_s$ with a single linear layer followed by a ReLU activation resulting in $\mathbf{z}_{q,\text{lin}}$ and $\mathbf{z}_{s,\text{lin}}$. $\mathbf{z}_q, \mathbf{z}_s, \mathbf{s}_{q,\text{lin}}, \mathbf{z}_{q,\text{lin}}$ are vectors with a size of 624, whereas feature vectors are then added with $\mathbf{z}_{q,s} = \mathbf{z}_q + \mathbf{z}_s + \mathbf{s}_{q,\text{lin}} + \mathbf{z}_{q,\text{lin}}$. We pass $\mathbf{z}_{q,s}$ through a MLP to predict attention offsets $\Delta\mathbf{q}$, which are together with $\mathcal{Z}$ processed by a 3D deformable attention module [8] followed by a MLP resulting in $\mathbf{z}_{\text{att}}$ with size 128. We use 4 different attention heads for each feature map in $\mathcal{Z}$ each with 2 deformable attention offsets, resultig in 32 total deformable attention offsets. The final occupancy o and flow $\mathbf{f}$ are obtained by: $o, \mathbf{f} = \text{MLP}(\mathbf{z}_{\text{att}} + \mathbf{z}_{q,\text{lin}} + \mathbf{z}_{s,\text{lin}} + \mathbf{z}_{q,s})$.

2.2. Training

We next describe hyperparameters used during training. During training, DIO uses a combination of occupancy loss $\mathcal{L}_{\text{occ}}$ and flow-consistency loss $\mathcal{L}_{\text{flow}}$ with $L = \mathcal{L}_{\text{occ}} + \lambda_{\text{flow}}\mathcal{L}_{\text{flow}}$ and $\lambda_{\text{flow}} = 0.02$. We train across 16 GPUs with 50,000 iterations each and a batch size of 1 on each GPU. For the instance occupancy-based negative prompts we choose a scaled bounding box $\mathbf{b}'_i$ with length $100\,\text{m}$, width $30\,\text{m}$ and height equal to the height of the original bounding box $\mathbf{b}$, centered around the original box. For the gaussian ball of the free-space based negative prompts, the standard deviation for the 3D positions is $\sigma_x = 20\,\text{m}, \sigma_y = 20\,\text{m}, \sigma_z = 2\,\text{m}$ (Equ.(3)). Here, per sample, we select $K = 10$ negative source points associated with $M = 800$ query points each. The goal of the described source point sampling is to ensure that the model generalizes as well as possible to any possible continuous source point "source" that may be used during inference. In summary we sample $|\mathcal{P}^+_{\text{scene}}| + |\mathcal{P}^-_{\text{scene}}| + |\mathcal{P}^+_{\text{obj}}| + |\mathcal{P}^-_{\text{obj}}| + |\mathcal{P}^-_{\text{box}}| + |\mathcal{P}^-_{\text{rand}}| = 88,000$ prompts during training.

2.3. LiDAR Renderer

For training the downstream task of LiDAR point cloud prediction, we utilize a ray depth prediction head as described in [2]. The ray depth prediction head for scene models ([2] and DIO-$\varnothing$) is implemented as described in the original publication. During training, the L1 loss $\mathcal{L}_{\text{raydepth}}$ is computed between the predicted depth and the ground truth depth for each ray, averaged over all rays.

DIO —when prompted with non-empty source points (DIO-L, DIO-NL, DIO-D) — and ImplicitO-4D [1] only predict occupancy for instances and not for the background. Therefore, we apply the L1 loss only to rays that intersect with the ground truth bounding boxes of instances. To ensure a fair comparison during evaluation, we still predict the depth of all rays. However, since these models are not trained on rays that don't hit instances, this may result in unreasonable predictions due to out-of-distribution errors. Hence, for these models, we extend the ray depth prediction head by adding an additional output that predicts the probability that a ray hits an instance p_{rayhit}. During training, we employ an additional cross-entropy loss, $\mathcal{L}_{\text{rayhit}}$, based on the predicted logit. Consequently, the final loss for the LiDAR renderer in instance-based occupancy models is given by $\mathcal{L} = \mathcal{L}_{\text{raydepth}} + \mathcal{L}_{\text{rayhit}}$.

During the evaluation of instance-based models, we filter out all rays whose probability of hitting an instance is smaller than 0.5 to get the final predicted point cloud. Filtering the rays based on a learned model output ensures a fair comparison in contrast to alternatives that would filter the rays based on the ground truth bounding box, which would leak instance size information during evaluation.

2.4. Experimental Details

Detector-based Source Points: In our experiments, source points from a detector are generated using HEDNet [9]. We use the original implementation[2] and the model processes four LiDAR point cloud frames (the current frame and three past frames) captured at 10 Hz to produce bounding box proposals. These proposals include the 3D centroid, width, length, height, heading, and confidence score. Final detections are obtained by applying a score threshold of 0.1 and performing non-maximum suppression (NMS) with an IoU threshold of 0.1. For the source points used to prompt DIO, we extract only the 3D centroids from the final detections.

Baselines: We use the official implementation[3] and trained model of 4D-Occ [10] as our baseline. For a fair comparison, we re-implement ImplicitO [1] —which originally predicts BEV occupancy over time (2D+1=3D)— and extend it to a 4D version that predicts 3D occupancy over time. All other implementation details follow the original work. Similarly, we implement UnO as described in its original publication [2].

Metrics: Due to the evaluation setting, the number of inference prompts during testing depends on the LiDAR rays and the number of objects per scene and, hence, can vary per frame. For more information about theoretical and experimental inference efficiency in an autonomy stack please refer to Sec. 3.

In line with [2, 10], we evaluate our model performance using depth L1 error (L1), and depth relative L1 error (AbsRel), Chamfer Distance (CD), Near Field Chamfer Dis-

[2] https://github.com/zhanggang001/HEDNet

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

tance (NFCD). The NFCD is computed within a specified region of interest (ROI) around the ego vehicle, set as $[-70, 70]$ m in both x-axis and y-axis, as well as the ROI in z-direction is $[-4.5, 4.5]$ m. We follow the implementation of the metrics as described in [2].

Instance-based Point Cloud Prediction. In the object-based evaluation the ground truth point cloud consists of points inside the current instance of interest. Hence, we only compute the L1 and AbsRel for rays, whose LiDAR points are located inside the bounding box. For models that predict only occupancy for instances (ImplicitO-4D, DIO-NL) we filter the rays based on the predicted ray hit probability as described above. If the probability is larger than 0.5, we consider the ray and its corresponding predicted depth as valid. Hence, for the NFCD calculation in the hit rate metrics $\text{HR}_{\text{NFCD},\tau}$ and $\text{HR}_{\text{Asy. NFCD},\tau}$ of these models we consider the *point cloud of the valid rays as the predicted point cloud*. We use the same ROI in x and y direction as in the leaderboard evaluation. The ROI in z-direction is $[-1.5, 3.5]$.

For the scene occupancy baselines [2, 10], we do not employ $\text{HR}_{\text{NFCD},\tau}$ because this metric favors models that predict only instance-based occupancy. This preference arises because the symmetric Chamfer distance, when applied to scene occupancy, also accounts for the shortest distances between distant background points and ground truth (GT) points within the bounding box. In contrast, the NFCD calculation for $\text{HR}_{\text{Asy. NFCD}}$ considers only the shortest distances originating from the GT points within the bounding box. Notably, the latter metric could even favor the scene occupancy baselines, as these models predict more points compared to DIO-NL or [1], which only predict point clouds of instances.

3 Additional Results

Leaderboard Results: Tab. 2 reports all methods of the leaderboard of the Argoverse 2: 4D Occupancy Forecasting Challenge[4] comparing to the results of DIO. We observe that DIO outperforms all baselines in terms of L1 (ranking metric), AbsRel, and NFCD. While DIO underperforms UnO in the CD metrics, that is likely because it was trained on a ROI smaller than the one stated in [2] due to memory constraints. The better performance in NFCD which evaluates with a smaller ROI supports this hypothesis.

Computational Complexity: In theory, DIO's runtime is linear w.r.t the prompts (i.e. query-source point pairs). In practice, all prompts run *in parallel* (constant runtime regime) until the GPU saturates (linear runtime regime). In Fig. 1, we visualize the inference time on a NVIDIA RTX 2080 Ti as a function of the number of prompts (query and source point pairs). For qualitative results, we query the entire voxel grid around the SDV for each source point, which results in a high prompt count. However, this is not required for autonomy, as shown in QuaD [11], which smartly queries only around candidate SDV future trajectories. For a similar number of prompts as used in QuAD, the runtime of DIO is similar to UnO. Assume a scenario with ten other relevant vehicles, and that the planner considers 2,000 candidate trajectories with 10 timesteps each, and 10 points to approximate the SDV's contour [1]. This results in 200,000 query points $|Q|$. Constructing scene and instance occupancy for ten vehicles would require $|P| = |Q| + |S||Q| = 11|Q| = 2,200,000$ prompts, but as shown in QuAD, query quantization can reduce $|Q|$ by *two orders of magnitude* without sacrificing performance, which sets $|P| \sim 22,000$, well within the real-time regime. Thanks to DIO's flexibility, even if there is a larger number of relevant objects in the scene, we could select a small subset of relevant agents for which to run instance occupancy (i.e., with non-empty source point), and handle the rest with scene occupancy (i.e., empty source point) to help with runtime scaling. In addition to query quantization, future works could further reduce the query points by using reachable sets of objects. We also found *sparse convolutions* crucial for the 3D backbone, as dense ones exceeded memory limits during training, while inference times remained similar (15.08 ms sparse vs. 13.62 ms dense).

	L1 (m) ↓	AbsRel (%) ↓	NFCD (m^2) ↓	CD (m^2) ↓
RayTracing [10]	2.50	35.00	3.62	17.03
occformer_ep15	3.57	22.00	3.35	91.61
4D-Occ [10]	3.22	19.00	2.45	72.74
occformer	2.97	17.00	2.03	71.33
Progressive ARM	2.32	13.00	1.81	71.41
UnO [2]	2.24	12.00	0.86	**8.10**
DIO-∅	**2.11**	**11.00**	**0.85**	13.96

Table 2. Full results of the Argoverse 2 leaderboard. For methods that are published we report the corresponding reference.

Parameter Scaling: In Fig. 2 we show experiments down-scaling the width of DIO's neck and header layers by a factor of 2,4,8, respectively. Even our most down-scaled model still outperforms UnO. We intentionally do not downsample the backbone as it is directly taken from the previous work SECOND [7] and it is therefore beyond our scope to evaluate it's efficacy when shrunk.

Effect of Box-based Negative Queries: Fig. 3 shows a comparison between our base model and an ablation that does not use additional negative queries. As evidenced by the results, neglecting the addition of negative queries leads to false-positive predicted occupancy in the bounding boxes of other agents, which underscores the importance of this design choice.

[4] https://eval.ai/web/challenges/challenge-page/1977/leaderboard/4662

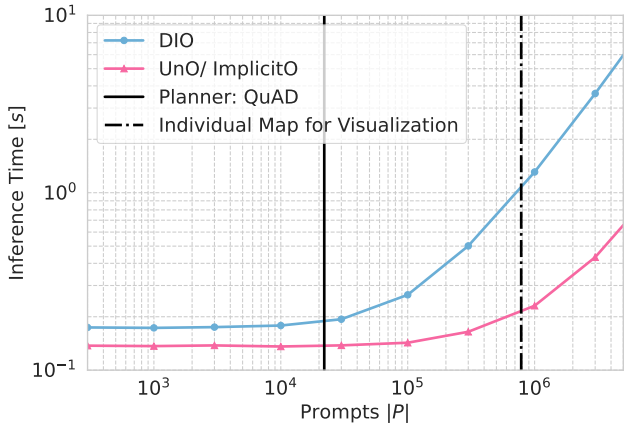

Figure 1. Computational complexity of DIOand baselines for different operation points.

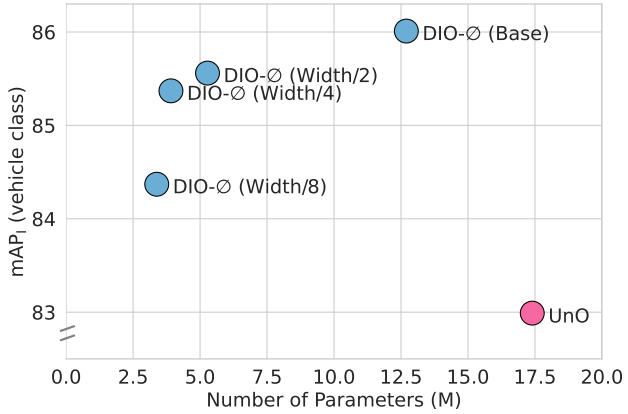

Figure 2. Down-scaling DIO's neck and header layers by a factor of 2, 4 and 8, respectively. Even our smallest model, with over 5 $\times$ fewer parameters than UnO, still outperforms it.

Effect of Flow-Consistency Loss: Fig. 4 ablates the influence of the flow-consistency loss. Not using this loss leads to a time-boundary effect where the present-time occupancy at query points $\mathbf{q}_{t=t_0}$ contains many gaps. This is because the model is trained to predict occupancy immediately behind the LiDAR return, so mimicking the input point cloud from the input is an effective way to minimize this loss when $\mathbf{q}_{t=t_0}$. However, since this is not effective to minimize the loss at $\mathbf{q}_{t>t_0}$ as the model does not have the corresponding LiDAR points as input, the flow consistency loss solves the issue. Beyond enhancing the scene completion capabilities of our model, incorporating flow as an additional output provides valuable information for the decision-making process of SDVs. Motion planners can leverage flow, alongside predicted occupancy, in its cost functions such as those related to headway [12].

Prompting with Different Source Points: Fig. 5 visualizes the predicted instance occupancy when DIO is prompted with source points at different locations. We observe that, when prompted with a source point lying within the bounding box of an object, the model correctly predicts the shape of the corresponding object. In contrast, when queried with a source point that lies in free space, the model does not predict any occupancy.

Multi-Modal Forecasting: Multi-modal forecasts, such as those in Fig. 7 - Example 2 (3 s), show different predicted maneuvers (lane-following vs. lane-changing). Moreover, in Fig. 6 (black circle) we illustrate turning-left vs. going-straight predictions. Sampling separate modes of the distribution is interesting for future work and could be done using generative models (e.g., diffusion).

Additional Visualizations Fig. 7 visualizes DIO's scene occupancy, instance occupancy, and scene flow outputs in additional examples. Fig. 8 provides visualizations of DIO's occupancy completions at different time steps of the same sequence. Lastly, Fig. 9 shows a comparison of DIO against the state-of-the-art 4D occupancy model UnO [2] in different scenes. For a more detailed description of the figures please refer to the individual caption. During all visualizations we use the centroids of the label bounding box to generate source points for instance occupancy. Furthermore, during visualization for both scene and object queries, we query the model on a full uniform voxel grid.

4 Limitations and Future Work

First, in our experiments, the detector uses a separate backbone to encode the LiDAR information, which induces additional computational overhead. Future work could train a joint model using a shared backbone with both a detection and an occupancy head. One advantage here could be that detections of varying quality could be used for source point generation during training. For example, one could generate additional negative queries based on false-positive detections, making the approach even more robust against the source point type. Moreover, further advantages of multi-task architectures have been demonstrated in [13].

Second, the separation of the occupancy of different instances is less and less clear as the future horizon grows, which can be partially observed in Fig. 7 (e.g., overlapping futures). This is expected as the model is trained to predict the *marginal* probability that an instance occupies a 4D point, but this depends on the *joint* distribution of behaviors of the different agents in the scene. Although the current behavior is perhaps desired for some applications, there might be others where modeling this joint distribution via a generative model [14–18] could be more adequate, which would allow to separate different futures.

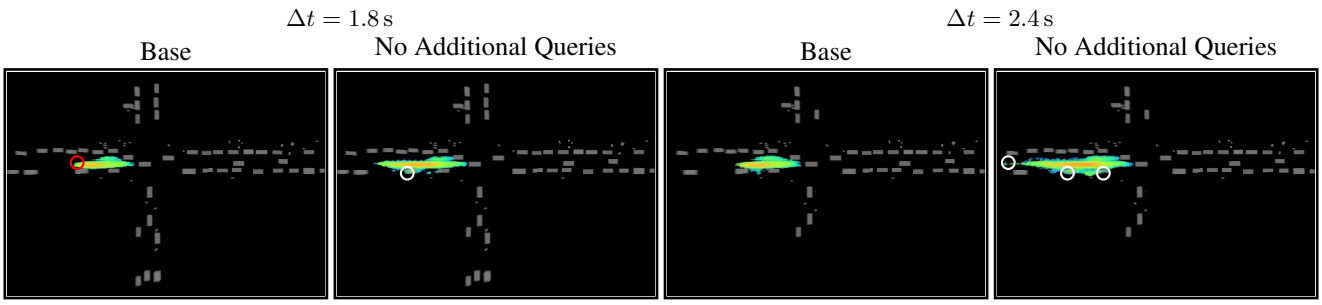

Figure 3. Ablation study on the impact of the additional instance-based negative query points. The figures show a scene with a one-sided street for two forecasted time steps. The source point lies inside the agent highlighted by the red circle. Omitting the negative queries results in false-positive occupancy predictions in other objects. For instance, the ablation without additional negative queries predicts occupancy of the queried agent inside the parking vehicles and also an unreasonable backward movement (white circles). In contrast, the base model predicts a multi-modal future indicating possible acceleration, deceleration and lane changes maneuvers. The ground truth bounding boxes are visualized in grey. The occupancy forecasts are colored by height and occupancy probability value.

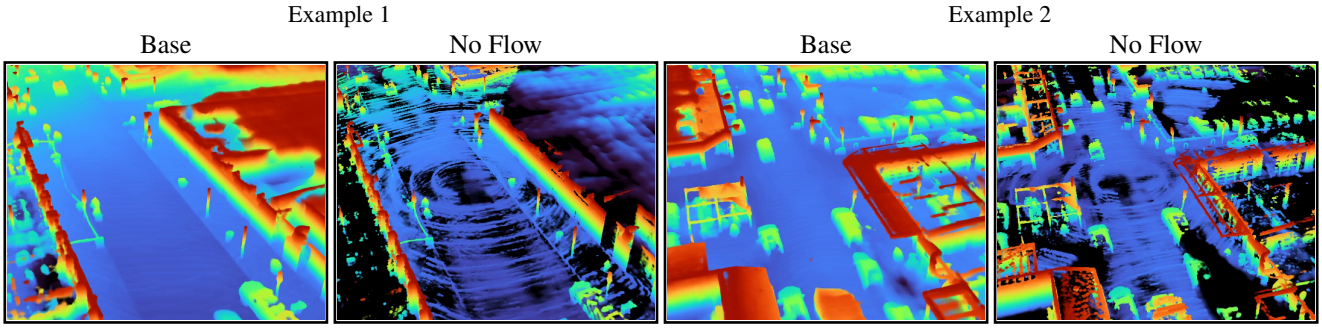

Figure 4. Ablation study on the impact of the flow-consistency loss $\mathcal{L}_{\text{flow}}$. Omitting this loss results in a voxelization effect at the present time step and false-negative predicted occupancy (black color).

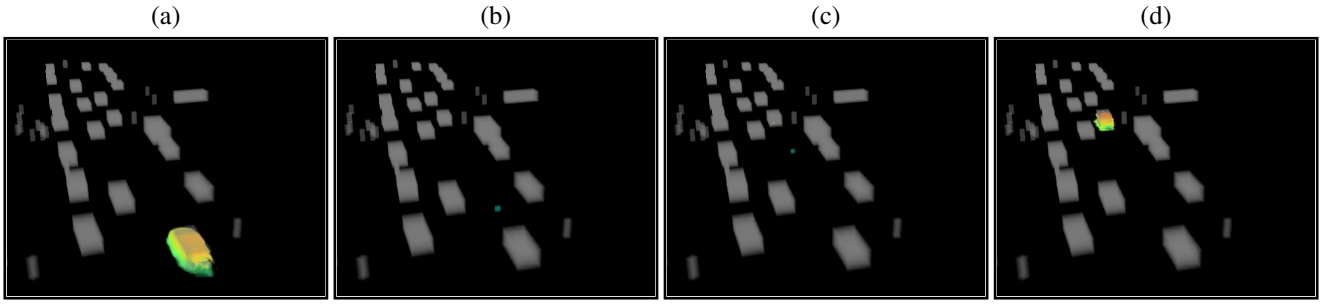

Figure 5. Prompting DIO with source points (cyan dot) moving on a straight line from object 1 (a), over free space (b, c) to object 2 (d). Grey boxes illustrate the label bounding boxes for reference. The model is queried in a voxel grid spanning the whole ROI.

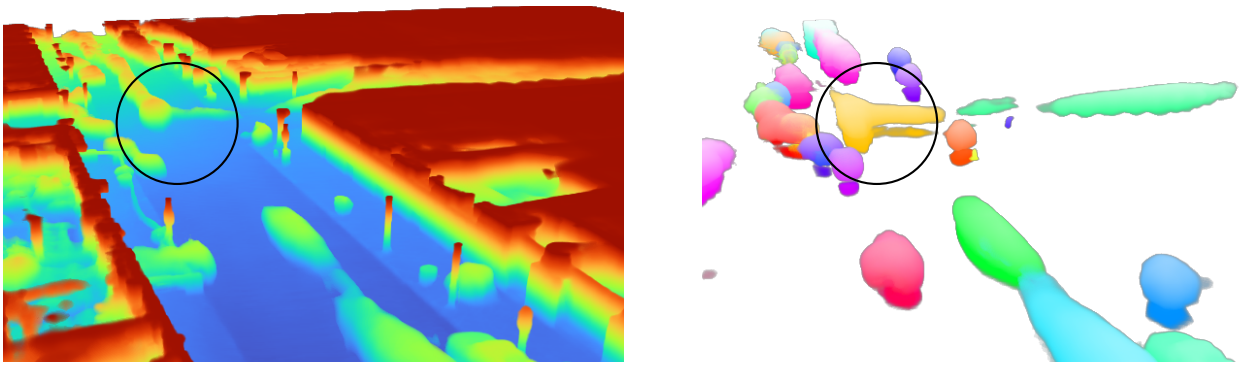

Figure 6. Multi-modal forecasts (turning-left vs. going-straight) indicated by the black circle.

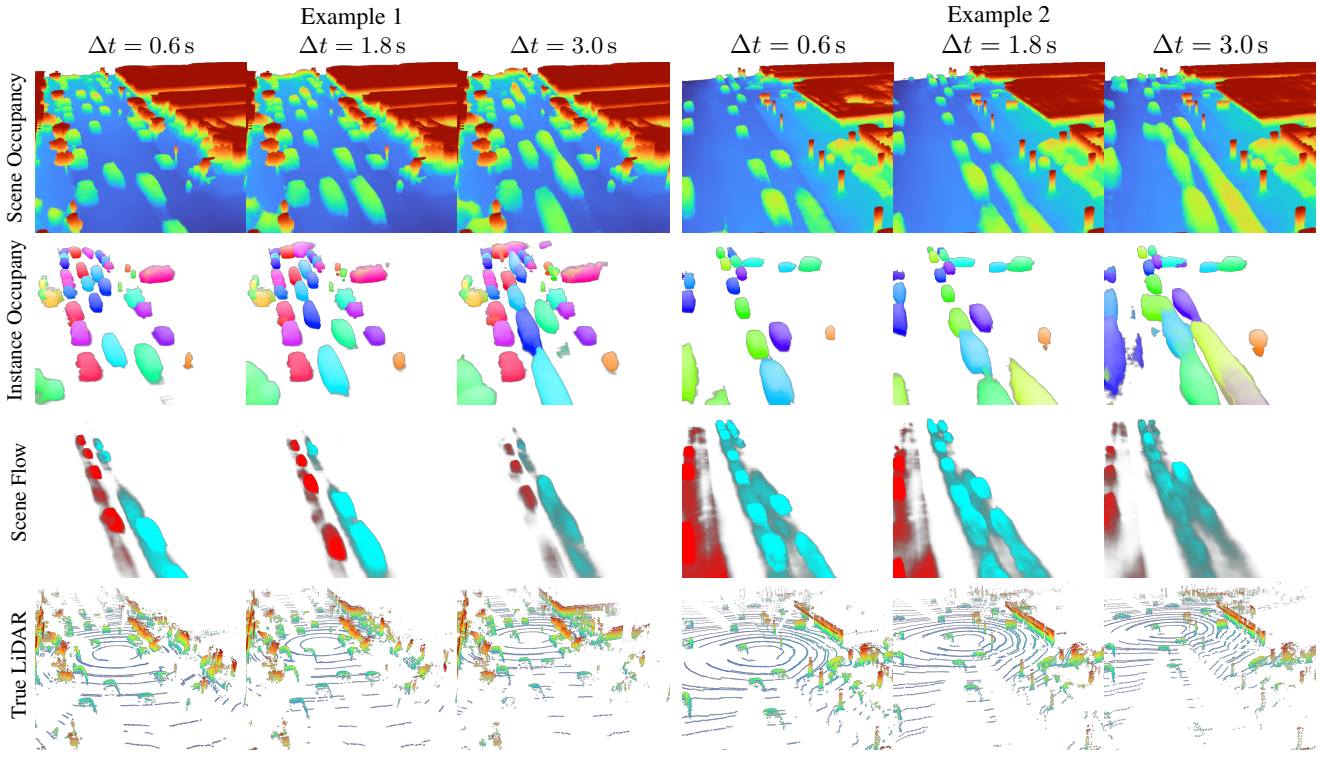

Figure 7. DIO forecasts visualized on two different examples. In example 1, we observe accurate shape predictions of pedestrians (e.g., orange instance) and decomposition of the crowded scene with flow predictions in both traffic directions. Example 2 shows multi-modal future occupancy predictions with reasonable uncertainty due to possible acceleration and deceleration as well as lane change and turning maneuvers. Notably, the decomposition into instance occupancy allows to gain insight from which instance this uncertainty originates which can lead to a safer decision-making process.

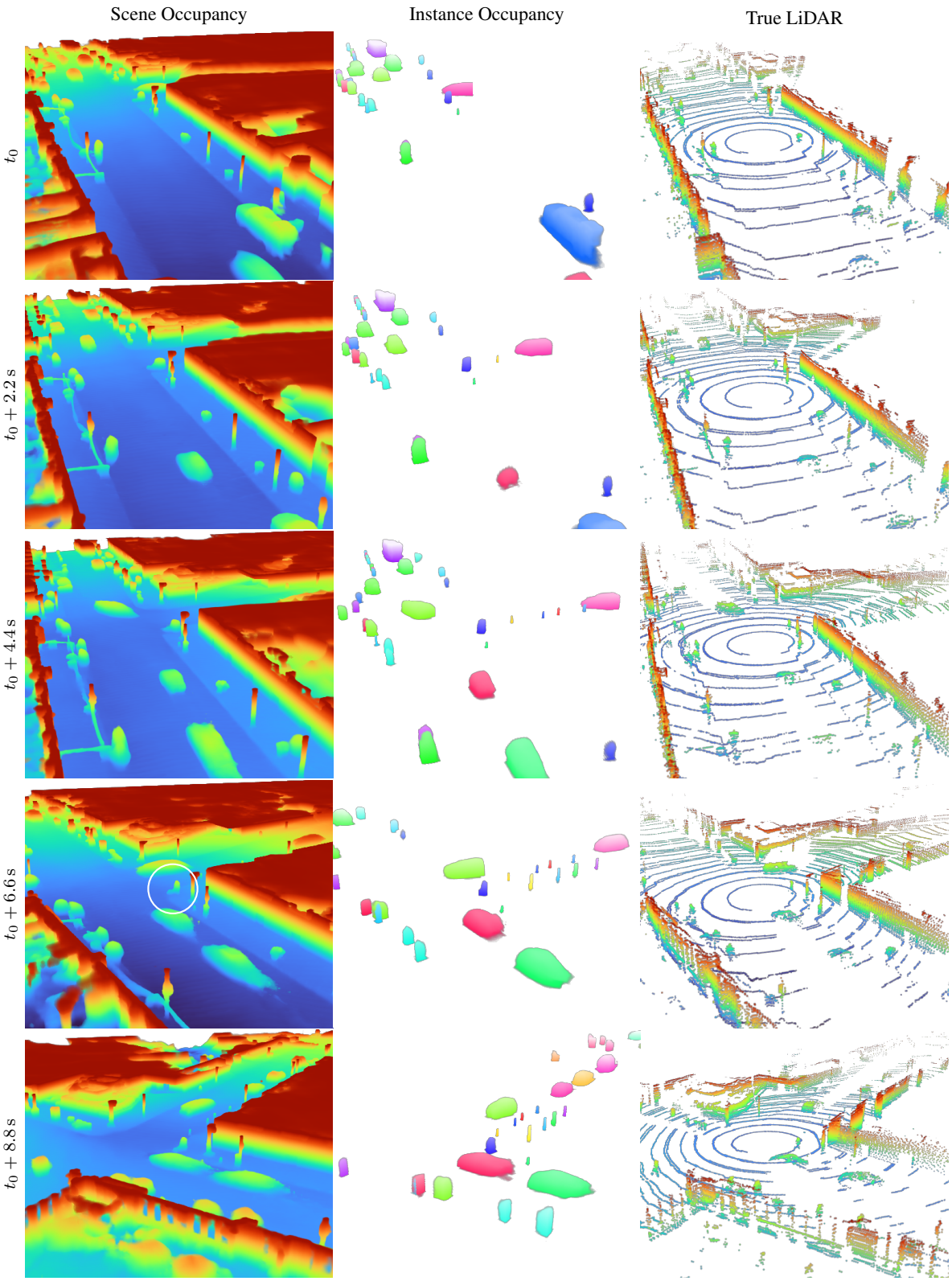

Figure 8. Visualizing DIO's scene and instance occupancy completion over multiple time steps of the same sequence. Our model predicts highly detailed static structures like a construction site barrier and signs, moving vehicles (e.g., green, pink and purple instances), pedestrians and even one with a stroller (white circle) waiting at the intersection.

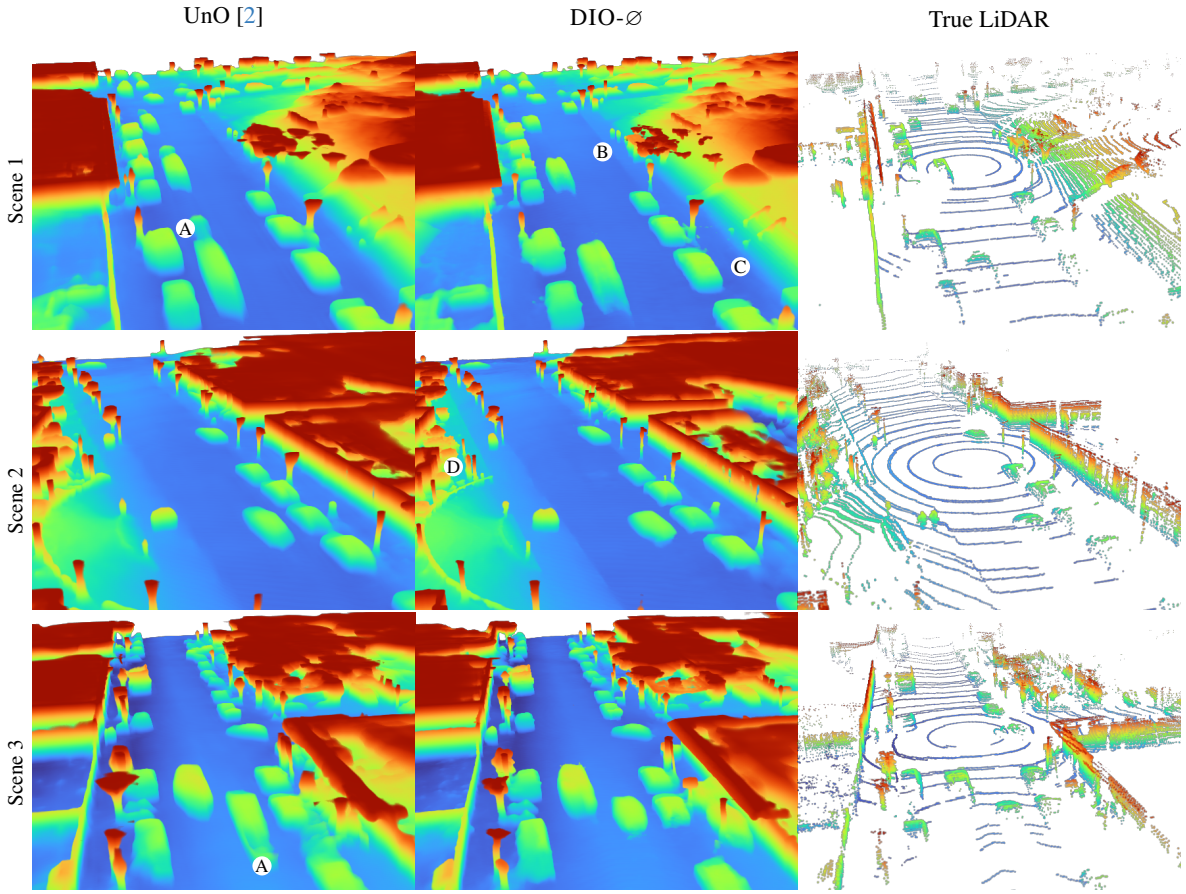

Figure 9. Comparing UnO's [2] and DIO-∅'s scene occupancy in multiple scenes where our model predicts more fine-grained structures. For instance, vehicle shapes of DIO's predictions often include the mirrors and wheels and the shape of signs is sharper. Furthermore, we observe the following among others: **(A)** UnO's output shows false-positive occupancy behind moving vehicles. **(B)** DIO predicts the pose of pedestrians waiting at a bus stop and a sharp border of a sidewalk. **(C)** DIO predicts individual stairs. **(D)** DIO predicts trees and a guard-rail in higher resolution.

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. 1, 2, 3

[2] Ben Agro, Quinlan Sykora, Sergio Casas, Thomas Gilles, and Raquel Urtasun. Uno: Unsupervised occupancy fields for perception and forecasting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14487–14496, June 2024. 1, 2, 3, 4, 8

[3] Dongrui Liu, Daqi Liu, Xueqian Li, Sihao Lin, Hongwei xie, Bing Wang, Xiaojun Chang, and Lei Chu. Self-supervised multi-frame neural scene flow, 2024. 1

[4] Jaeyeul Kim, Jungwan Woo, Ukcheol Shin, Jean Oh, and Sunghoon Im. Flow4d: Leveraging 4d voxel network for lidar scene flow estimation. *IEEE Robotics and Automation Letters*, pages 1–8, 2025. 1

[5] Kavisha Vidanapathirana, Shin-Fang Chng, Xueqian Li, and Simon Lucey. Multi-body neural scene flow. In *2024 International Conference on 3D Vision (3DV)*. IEEE, 2024. 1

[6] Ishan Khatri, Kyle Vedder, Neehar Peri, Deva Ramanan, and James Hays. I can't believe it's not scene flow! In *ECCV*, 2024. 1

[7] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10), 2018. 1, 3

[8] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results, 2018. 1, 2

[9] Gang Zhang, Junnan Chen, Guohuan Gao, Jianmin Li, and Xiaolin Hu. HEDNet: A hierarchical encoder-decoder network for 3d object detection in point clouds. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023. 2

[10] 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. 2, 3

[11] Sourav Biswas, Sergio Casas, Quinlan Sykora, Ben Agro, Abbas Sadat, and Raquel Urtasun. Quad: Query-based interpretable neural motion planning for autonomous driving. *arXiv preprint arXiv:2404.01486*, 2024. 3

[12] 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. 4

[13] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, Lewei Lu, Xiaosong Jia, Qiang Liu, Jifeng Dai, Yu Qiao, and Hongyang Li. Planning-oriented autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 17853–17862, June 2023. 4

[14] Sergio Casas, Cole Gulino, Simon Suo, Katie Luo, Renjie Liao, and Raquel Urtasun. Implicit latent variable model for scene-consistent motion forecasting. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII 16*, pages 624–641. Springer, 2020. 4

[15] Alexander Cui, Sergio Casas, Abbas Sadat, Renjie Liao, and Raquel Urtasun. Lookout: Diverse multi-future prediction and planning for self-driving. In *ICCV*, 2021.

[16] Chiyu Jiang, Andre Cornman, Cheolho Park, Benjamin Sapp, Yin Zhou, Dragomir Anguelov, et al. Motiondiffuser: Controllable multi-agent motion prediction using diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9644–9653, 2023.

[17] Ari Seff, Brian Cera, Dian Chen, Mason Ng, Aurick Zhou, Nigamaa Nayakanti, Khaled S Refaat, Rami Al-Rfou, and Benjamin Sapp. Motionlm: Multi-agent motion forecasting as language modeling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8579–8590, 2023.

[18] Christopher Diehl, Timo Sebastian Sievernich, Martin Krüger, Frank Hoffmann, and Torsten Bertram. Uncertainty-aware model-based offline reinforcement learning for automated driving. *IEEE Robotics and Automation Letters*, pages 1167–1174, 2023. 4