TARS: Traffic-Aware Radar Scene Flow Estimation

Supplementary Material

A. Addntional Implementation Details

A.1. Further Model Details

In this section we provide more details about TARS.

When computing the point-level flow embeddings in Sec. 3.2.1, we use an MLP for positional encoding but omit it in those equations for simplicity. It is also applied in cross-attention in the TVF decoder (Sec. 3.4).

In the scene update stage in Sec. 3.3, we firstly apply CNN layers to the feature map $\chi^l_{\text{od}} \in \mathbb{R}^{H_l \times W_l \times D_l}$ to adapt the object features. We then apply pooling layers, because feature maps from the OD pyramid have varying resolutions (H_l, W_l) at each level, and we match them with the predefined shape of the coarse TVF.

In the flow painting stage in Sec. 3.3, we fuse the scene $\mathbf{X}^l_{\text{traffic}}$ and motion feature $\mathbf{X}^l_{\text{motion}}$ in a spatial attention style [21], specifically:

$$\mathbf{X}^l_{\text{fusion}} = \mathbf{w}^l \odot \mathbf{X}^l_{\text{traffic}} + (1 - \mathbf{w}^l) \odot \mathbf{X}^l_{\text{motion}}, \quad \text{with} \tag{9}$$

$$\mathbf{w}^l = \sigma(\text{Concat}(\mathbf{W}_1 * \mathbf{X}^l_{\text{traffic}}, \mathbf{W}_2 * \mathbf{X}^l_{\text{motion}})), \tag{10}$$

where $\mathbf{W}_1, \mathbf{W}_2$ are CNN kernels that generate pixel-wise attention scores, $\mathbf{w}^l$ is the pixel-wise attention weights.

In the flow painting stage in Sec. 3.3, we use axial attention to provide the global receptive field, specifically:

$$\text{TVF}^l_\tau = \text{TVF}^l_\tau\text{-H} + \text{TVF}^l_\tau\text{-W}, \quad \text{with} \tag{11}$$

$$\text{TVF}^l_\tau\text{-H} = \text{Self-A}(\text{Col}(\text{TVF}^l_{\tau-1})), \tag{12}$$

$$\text{TVF}^l_\tau\text{-W} = \text{Self-A}(\text{Row}(\text{TVF}^l_\tau\text{-H})), \tag{13}$$

$$\text{Self-A}(\cdot) = \text{Attention}(\cdot, \cdot, \cdot), \tag{14}$$

where $\tau = \{1, ..., \omega\}$, $\text{Col}(\cdot)$ collects W column vectors and $\text{Row}(\cdot)$ collects H row vectors from the 2D TVF.

On the proprietary dataset, we provide **all** models with ego-motion $\Omega \in \mathbb{R}^{4\times4}$ as known input. We apply ego-motion compensation before the PointGRU blocks (Sec. 3.5), aligning O and P, as well as P and Q into the same coordinate system.

A.2. Ego-Motion Head Details

On the VOD dataset, our TARS-ego adopts the same approach as RaFlow and CMFlow [6, 7] to train an ego-motion head. It predicts the ego transformation, which is treated as static flow and assigned to all static points. This assignment requires our model to distinguish between moving and static points, which is achieved by a motion-segmentation head.

Motion-segmentation head. The motion-segmentation head takes the final flow embeddings from the L-th level of TARS-ego as input and outputs a probability map $\mathbf{S}$ indicating for each point whether it is moving or static. We use a simple three-layer MLP followed by a sigmoid function to generate the probability map. A binary segmentation mask can be generated from $\mathbf{S}$ using a threshold of 0.5.

We apply the same training strategy in previous works [6, 7]. During training, we use ground truth (GT) segmentation pseudo-labels as the segmentation mask, in order to provide stable segmentation results for training the ego-motion head. During testing, we use the model output from the motion-segmentation head.

Ego-motion head. Since P_{warp} is obtained by warping P using the scene flow output F^L from the last L-th level, they can be used as natural correspondences to infer the ego motion. In the ego-motion head, we take P, P_{warp}, and $\mathbf{S}$ as input. Then we perform the differentiable weighted Kabsch algorithm [10], which infers rigid transformation between P and P_{warp}, with $(1-\mathbf{S})$ as the point weight to neglect moving points. Finally, we use the binary motion segmentation mask generated from $\mathbf{S}$ to identify static points and assign the inferred ego-motion to static points as their final scene flow F_{bg}. Since the ego-motion is inferred between P and P_{warp}, a more accurate F^L leads to improved ego-motion predictions and, ultimately, more accurate static flow F_{bg}. Results of the ego-motion estimation are shown in Tab. A.

Table A. Evaluation of the ego-motion head on VOD. RTE: relative translation error. RAE: relative angular error.

Model	RTE [m]	RAE [°]
CMFlow [7]	0.083	0.140
TARS-ego (ours)	**0.059**	**0.098**

A.3. TARS-ego and TARS-superego

On the VOD dataset, we evaluate our model using two setups that apply the ego-motion information differently: TARS-ego and TARS-superego, results shown in Tab. 3 of the main manuscript. TARS-ego uses GT ego-motion to train a ego-motion head and motion-segmentation head to refine the static flow F_{bg}; while TARS-superego uses ego-motion as input and applies ego-motion compensation, expecting the static flow F_{bg} to be zero vectors (without those additional heads). On the proprietary dataset, since ego-motion compensation is applied to **all** models, we do not specifically label our model as TARS-superego. In this section, we discuss the reason of leveraging ego-motion.

On one hand, ego-motion information from an odometer is simple, easy to obtain, yet highly effective. Utilizing this information enhances scene flow performance in both ways:

as supervision signal for an ego-motion head, or as known input for ego-motion compensation. Nevertheless, the latter aligns more closely with real-world autonomous driving scenarios, where a GPS/IMU sensor is often available. It helps address the challenges of radar scene flow.

On the other hand, performing ego-motion compensation maximizes the potential of our model TARS. By compensating point cloud P into the coordinate system of point cloud Q, the points in P and the OD feature map from Q share a common coordinate system, starting from the lowest level of TARS. This eliminates the need for the lower levels of TARS to firstly warp the points in P closer to the corresponding object features, ensuring alignment within the TVF throughout the process.

Algorithm 1 TARS-ego Forward Pass

Require: Consecutive point clouds P, Q, previous point cloud O and its low-level hidden state h_{t-2}
Ensure: Scene flow F^L, motion segmentation mask $\mathbf{S}$, predicted ego-motion $\hat{\Omega}$, hidden state h_{t-1}

▷ Collect multi-level OD features
$\{\chi^l_{\text{od}}\}_{l=1}^{L} \leftarrow \text{ODBranch}(Q)$
▷ 1. Point feature extraction
$(P, Q, \mathbf{p}, \mathbf{q}) \leftarrow \text{MLP}(P, Q)$
▷ 2. Low-level temporal module
$(P, Q, \mathbf{p}, \mathbf{q}, h_{t-1}) \leftarrow \text{PointGRU}(P, Q, \mathbf{p}, \mathbf{q}, O, h_{t-2})$
▷ 3. Point patch feature extraction
▷ when $l = L$: full point set
$\{P^l, Q^l, \mathbf{p}^l, \mathbf{q}^l\}_{l=1}^{L} \leftarrow \text{MultiScaleEncoder}(P, Q, \mathbf{p}, \mathbf{q})$
▷ 4. Flow & embeddings at $l = 1$
$P^1_{\text{warp}} \leftarrow P^1 + \mathbf{0}$
$\mathbf{e}^1_{\text{point}} \leftarrow \text{PointLevelEmb}(P^1_{\text{warp}}, Q^1, \mathbf{p}^1, \mathbf{q}^1)$
$(F^1, \mathbf{e}^1) \leftarrow \text{FlowHead}(\mathbf{e}^1_{\text{point}}, \varnothing, \varnothing)$
▷ 5. Coarse-to-fine prediction
$\text{TVF}^1 \leftarrow \varnothing$
for $l = 2$ up to L **do**
    $\text{TVF}^l \leftarrow \text{TVF\_Encoder}(P^{l-1}, \mathbf{p}^{l-1}, F^{l-1}, \mathbf{e}^{l-1}, \chi^l_{\text{od}}, \text{TVF}^{l-1})$
    $P^l_{\text{warp}} \leftarrow P^l + \text{Interp}(F^{l-1})$
    $\mathbf{e}^l_{\text{point}} \leftarrow \text{PointLevelEmb}(P^l_{\text{warp}}, Q^l, \mathbf{p}^l, \mathbf{q}^l)$
    $\hat{\mathbf{p}}^l = \text{Concat}(\text{Interp}(\mathbf{e}^{l-1}), \mathbf{p}^l)$
    $\mathbf{e}^l_{\text{traffic}} \leftarrow \text{TVF\_Decoder}(P^l_{\text{warp}}, \hat{\mathbf{p}}^l, \text{TVF}^l)$
    $(F^l, \mathbf{e}^l) \leftarrow \text{FlowHead}(\mathbf{e}^l_{\text{point}}, \text{Interp}(\mathbf{e}^{l-1}), \mathbf{e}^l_{\text{traffic}})$
end for
▷ 6. Static points correction
$(F^L, \mathbf{S}, \hat{\Omega}) \leftarrow \text{EgoHead}(F^L, \mathbf{e}^L, P^L)$
return $F^L, \mathbf{S}, \hat{\Omega}$

A.4. Pseudo-Code

In this section, we present the pseudo-code of TARS-ego (Algorithm 1) and detailed explanations of the TVF encoder and decoder modules (Algorithms 2 and 3).

The forward pass of TARS-ego takes P, Q, O, h_{t-2} and outputs $F^L, \mathbf{S}, \hat{\Omega}, h_{t-1}$. It first computes low-level point features, updates temporal context with PointGRU [9], and encodes point patch features using the MultiScaleEncoder. At the lowest level of TARS, the FlowHead initializes scene flow and embeddings using only point-level motion cues. A coarse-to-fine architecture then refines the flow: the TVF encoder builds the traffic-level motion understanding, the TVF decoder captures rigid motion in surrounding space, and the FlowHead produces refined flow from dual-level flow embeddings. Finally, the EgoHead corrects static points with predicted ego-motion.

Algorithm 2 TVF_Encoder

Require: Previous points P^{l-1}, features $\mathbf{p}^{l-1}$, flow F^{l-1}, flow embeddings $\mathbf{e}^{l-1}$, TVF^{l-1}, OD feature map χ^l_{od}
Ensure: Updated TVF^l

▷ 1. Scene update
if $\text{TVF}^{l-1} \neq \varnothing$ **then**
    $X_{\text{traffic}} \leftarrow \text{GRU}(\text{Conv}(\chi^l_{\text{od}}), \text{TVF}^{l-1})$ ▷ Eq. (4)
else
    $X_{\text{traffic}} \leftarrow \text{Conv}(\chi^l_{\text{od}})$
end if
▷ 2. Flow painting
$P^{l-1}_{\text{warp}} \leftarrow P^{l-1} + F^{l-1}$
$voxel \leftarrow \text{Voxelize2D}(P^{l-1}_{\text{warp}}, \text{Concat}(\mathbf{p}^{l-1}, \mathbf{e}^{l-1}))$
$X_{\text{motion}} \leftarrow \text{Point2Grid\_SelfAttn}(voxel)$
$X_{\text{fusion}} \leftarrow \text{SpatialAttnFusion}(X_{\text{traffic}}, X_{\text{motion}})$ ▷ Eq. (9)
▷ Global attention
$\text{TVF}^l \leftarrow \text{AxialAttn}(X_{\text{fusion}})$
return TVF^l

The Inputs to our TVF encoder (Algorithm 2) are previous level's P^{l-1}, $\mathbf{p}^{l-1}$, F^{l-1}, $\mathbf{e}^{l-1}$, TVF^{l-1}, χ^l_{od}, and the output is the updated TVF^l for the current level. In the scene update stage, a ConvGRU [1] integrates prior TVF with OD features to encode traffic context. Then, in the flow painting stage, warped points are voxelized and processed by self-attention to capture motion context. Finally, the spatial attention fuses traffic and motion streams, and the global attention enhances the combined representation.

Algorithm 3 TVF_Decoder

Require: Warped points P^l_{warp}, features $\hat{\mathbf{p}}^l$, TVF^l
Ensure: Traffic-level flow embeddings $\mathbf{e}^l_{\text{traffic}}$

$\mathcal{N}_{\text{TVF}} \leftarrow \text{KNN}(K, P^l_{\text{warp}}, \text{TVF}^l)$ ▷ Eq. (6)
$\mathbf{e}^l_{\text{traffic}} \leftarrow \text{Grid2Point\_CrossAttn}(\hat{\mathbf{p}}^l, \text{TVF}^l, \text{TVF}^l, \mathcal{N}_{\text{TVF}})$
return $\mathbf{e}^l_{\text{traffic}}$

The TVF decoder (Algorithm 3) takes warped points P^l_{warp}, combined features $\hat{\mathbf{p}}^l$ and the TVF^l. It finds the K

nearest grid cells for each point and applies a grid-to-point cross-attention. The output is the traffic-level flow embedding $\mathbf{e}^l_{\text{traffic}}$, which is then consumed by the FlowHead.

A.5. Reproduced LiDAR-Model Details

For a complete comparison with prior works, we include latest LiDAR scene flow models [12, 23] in the comparison on the VOD dataset, results shown in Tab. 3 in the main manuscript. In this section, we provide the details of our reproduction of DeFlow [23] and Flow4D [12].

DeFlow and Flow4D underperform compared to TARS and CMFlow [7]. This is because their fully-voxel representation, designed for the efficiency challenge in large-scale LiDAR point clouds, becomes unsuitable for sparse radar point clouds. In DeFlow [23], the GRU-based voxel-to-point refinement module becomes ineffective, because most pillars contain only a few radar point, leading to a loss of precise localization (although this effectively improves the efficiency). Similarly, Flow4D's [12] fine-grain 4D voxel representation is less appropriate for sparse radar data.

Originally implemented on the LiDAR dataset Argoverse 2 [20], DeFlow and Flow4D take two LiDAR point clouds $P \in \mathbb{R}^{N\times 3}$ and $Q \in \mathbb{R}^{M\times 3}$ as input. Note that, both of them take only x, y, z coordinates as input channels, without using the intensity information from LiDAR. For fair comparison on the VOD dataset, we extend their input to 5 dimensions, adding RRV (relative radial velocity) and RCS (radar cross-section). We also adapt their grid feature map shape according to VOD's point cloud range.

Flow4D supports multi-frame input, while its 2-frame setup also achieves SOTA performance on Argoverse 2. As labeled in Tab. 3, we test the Flow4D-2frame setup for a fair comparison with other models.

During training, we use the following losses: $\mathcal{L}_{\text{sc}}$, $\mathcal{L}_{\text{ss}}$, $\mathcal{L}_{\text{rd}}$, $\mathcal{L}_{\text{fg}}$, and $\mathcal{L}_{\text{bg}}$ (details see Appendix D.2). We keep their original implementations, and since these models lack ego-motion heads, we omit TARS-ego & CMFlow's additional losses ($\mathcal{L}_{\text{seg}}$, $\mathcal{L}_{\text{ego}}$, $\mathcal{L}_{\text{opt}}$). This setup is the same as the training of our TARS-no-ego model, where we also removed the ego-motion and motion-segmentation head (results shown in Tab. 6 Group No. 1, highlighted in blue). Certainly, ego-motion compensation is also not applied for TARS-no-ego. Under the same training loss and "no-ego" setup, DeFlow and Flow4D still have a large accuracy gap compared to TARS-no-ego (16.7% and 18.5% on AccS; 27.7% and 33.1% on AccR, respectively).

A.6. Dual-Task Network Training Strategy

TARS jointly performs object detection and radar scene flow, enhancing the scene flow accuracy and also enabling a comprehensive perception. Experiments in PillarFlowNet [8] show that joint training of these two tasks reduces the performance of each individual task. Our experimental results of joint training are consistent with theirs, shown in Tab. B. Therefore, we perform a staged training: first train an object detector, then freeze its parameters to stabilize feature maps for the scene flow branch. This maintains OD performance while also allowing flexibility to adopt different training strategies for the object detector.

Table B. Joint training experiment on the proprietary dataset.

Training Strategy			Scene Flow Branch				OD Branch AP [%]↑			
OD pre-trained	Joint	Staged	MEPE↓	AccS↑	AccR↑	SEPE↓	Car	Ped.	Cycl.	Truck
no	✓		0.094	60.4	77.0	**0.026**	65.3	51.7	55.4	52.3
yes, not frozen	✓		**0.067**	66.8	**87.1**	0.036	66.7	53.2	60.0	53.8
yes, frozen		✓	0.069	**69.8**	86.8	0.038	**67.5**	**54.5**	**62.2**	**55.9**

A.7. Object Detection Branch Details

We use an off-the-shelf object detector as the OD branch on both datasets and freeze its weights. In this way, we maintain the OD performance and stabilize the feature map. As discussed in Sec. 3.1, the object detection branch should have a grid-based backbone to provide bird's eye view (BEV) feature maps to the scene flow branch. This excludes point-based detectors, such as 3D-SSD [24], where features are preserved in point form rather than being structured into grids. In contrast, voxel-based detectors typically generate multi-scale feature maps, which aligns well with TARS's hierarchical architecture. Therefore, we utilize the OD feature map at each corresponding level within our hierarchical design.

We employ PointPillars [14] as the object detector on the VOD dataset (result shown in Tab. C). The OD branch result on the proprietary dataset is shown in Tab. D.

Table C. Performance of the object detection branch on the VoD dataset. The results are reported separately for the entire annotated area and the driving corridor area.

Dataset	AP in the entire annotated area [%]↑				AP in the driving corridor area [%]↑			
	Car	Pedestrian	Cyclist	mAP	Car	Pedestrian	Cyclist	mAP
VOD	30.59	30.21	61.95	40.92	64.19	41.61	85.04	63.61

Table D. Performance of the OD branch on the proprietary dataset.

Dataset	AP in the entire annotated area [%]↑			
	Car	Pedestrian	Cyclist	Truck
proprietary	67.46	54.50	62.15	55.86

B. Additional Ablation Studies

B.1. OD Branch Ablation Study

In Tab. E, we test the effect of the object detection branch's accuracy on the performance of the scene flow branch. By adjusting the feature channels of PointPillars, we create three models of varying sizes: PP-L, PP-M, and PP-S (from large to small).

However, we observed that even with massive reductions in feature channels and parameters, PP-M and PP-S still maintain OD accuracy comparable to PP-L, and their corresponding scene flow branch performance is also similar. To isolate the corrective effect of the ego-motion head on scene flow predictions, we further test the performance of TARS-no-ego under three PointPillars setups. The experiments show that using PP-L achieves the best performance in both TARS-ego and TARS-no-ego models. However, due to the similar accuracy of the OD branch, the performance differences in the scene flow branch are not significant.

Table E. Ablation study of the OD branch on the VOD dataset. PP-S, PP-M, PP-L refer to three different PointPillars [14] setups. TARS-no-ego: without ego-motion head and three supervision signals $\mathcal{L}_{\{\text{seg, ego, opt}\}}$.

Object Detection branch						Scene Flow Branch					
Model	#Params	AP in the entire annotated area				TARS-ego			TARS-no-ego		
		Car	Ped.	Cycl.	mAP	EPE	AccR	AccS	EPE	AccR	AccS
PP-S	**0.05M**	25.63	28.64	61.04	38.44	0.093	38.1	68.5	0.117	27.0	57.4
PP-M	0.33M	29.73	27.28	59.33	38.78	0.095	37.1	67.2	0.114	27.9	58.3
PP-L	9.30M	**30.59**	**30.21**	**61.95**	**40.92**	**0.092**	**39.0**	**69.1**	**0.111**	**28.5**	**59.3**

B.2. Class-wise Scene Flow Evaluation

Khatri et al. [11] proposed a novel scene flow evaluation method, computing class-aware EPE to analyze the failure cases in scene flow. We provide this analysis in Tab. F for the best three models in the VOD dataset, i.e. DeFlow [23], CMFlow [7] and our TARS-ego. Note that, Here we adapt the Bucket Normalized EPE [11] to MRNE. The reason is that, on the VOD dataset we compute MRNE for dynamic points, while EPE is an overall metric for all points. Therefore, to achieve class-aware evaluation, we compute per-class MRNE.

Table F. Scene flow evaluation on the VOD dataset, with per-class MRNE analysis. "Sup.": supervision.

		Overall				Moving	Static	Per-class MRNE↓		
Method	Sup.	EPE↓	AccS↑	AccR↑	RNE↓	MRNE↓	SRNE↓	Car	Ped.	Cycl.
DeFlow [23]	Cross	0.217	11.8	31.6	0.087	0.098	0.085	0.092	0.081	0.106
CMFlow [7]	Cross	0.130	22.8	53.9	0.052	0.072	0.049	0.064	0.062	0.086
TARS-ego (ours)	Cross	**0.092**	**39.0**	**69.1**	**0.037**	**0.061**	**0.034**	**0.051**	**0.052**	**0.073**

B.3. Comparison with Fully-supervised Methods

Ding et al. [7] compared weakly-supervised CMFlow with fully-supervised LiDAR models, *e.g.* PV-RAFT [19]. We continue this discussion in Tab. G. Note that the "fully-supervised" setup differs primarily in its annotations [7]: using human-annotated bboxes and tracking IDs to derive actual scene flow GT. In contrast, the aforementioned pseudo scene flow GT $\hat{F}_{\text{fg}}$ in the foreground loss is generated using an off-the-shelf, pre-trained LiDAR multi-object tracking model, requiring no additional annotation efforts.

Experiments in [7] demonstrated that the weakly-supervised CMFlow (with cross-modal losses) achieved 3% lower AccS and 4.8% lower AccR compared to the fully-supervised PV-RAFT. Nevertheless, CMFlow was able to achieve comparable EPE and AccR to the fully-supervised PV-RAFT by leveraging extra weakly-supervised training samples without costly annotations ($\sim 140\%$ more than the number of training samples used for PV-RAFT). In contrast, as shown in Tab. G, TARS-ego surpasses fully-supervised PV-RAFT with a 13.2% higher AccS and 10.4% higher AccR, without requiring any additional training samples.

Table G. Comparison with fully-supervised models on the VOD dataset. Fully-supervised model results are cited from [7]. "Sup." indicates the supervision signal, Full: training with actual scene flow ground truth, Self: training with only self-supervised losses, Cross: with additional cross-modal losses.

Method	Sup.	EPE [m]↓	AccS [%]↑	AccR [%]↑	RNE [m]↓
FlowStep3D [13]	Full	0.286	6.1	18.5	0.115
Bi-PointFlowNet [4]	Full	0.242	16.4	35.0	0.097
FlowNet3D [15]	Full	0.201	16.9	37.9	0.081
PointPWC-Net [22]	Full	0.196	17.7	39.7	0.079
PV-RAFT [19]	Full	**0.126**	**25.8**	**58.7**	**0.051**
PointPWC-Net [22]	Self	0.422	2.6	11.3	0.169
FlowStep3D [13]	Self	0.292	3.4	16.1	0.117
CMFlow [7]	Cross	**0.141**	**22.8**	**53.9**	**0.052**
TARS-ego (ours)	Cross	**0.092**	**39.0**	**69.1**	**0.037**

Table H. TARS under fully-supervised setup on VOD.

Method	Sup.	EPE↓	AccS↑	AccR↑	RNE↓	MRNE↓	SRNE↓
CMFlow [7]	Cross	0.130	22.8	53.9	0.052	0.072	0.049
PV-RAFT [19]	Full	0.126	25.8	58.7	0.051	N/A	N/A
TARS-ego (ours)	Cross	0.092	39.0	69.1	0.037	0.061	0.034
TARS-ego (ours)	Full	**0.087**	**42.2**	**72.7**	**0.035**	**0.059**	**0.031**

Training TARS under fully-supervised setup. We test the performance of TARS-ego under the aforesaid fully-supervised setup and disabled the self-supervised losses: $\mathcal{L}_{\text{sc}}, \mathcal{L}_{\text{ss}}, \mathcal{L}_{\text{rd}}, \mathcal{L}_{\text{opt}}$. As shown in Tab. H, the fully-supervised training improves the performance of TARS across all metrics, demonstrating TARS's potential with more precise GT.

B.4. Emergency-Scenario Analysis

In TARS, perceiving traffic is one of the core ideas. Incorporating traffic modeling into motion prediction may raise concerns about emergent situations, such as a pedestrian suddenly entering traffic. Nevertheless, this is addressed by the overall design of TARS: all traffic-related modules in TARS employ attentive layers, enabling the neural network to adaptively balance the reliance on point-level and traffic-level features.

This is particularly evident in the encoding and decoding processes of the TVF. In the TVF encoder, we treat the point property $\mathbf{p}^{l-1}$ (from PointNet) and motion features $\mathbf{e}^{l-1}$ (flow embeddings) as equally important by concatenating them and then applying point-to-grid self-attention. Similarly, in the TVF decoder, we form the query by concatenating $\mathbf{p}^{l}$ with the upsampled $\mathbf{e}^{l-1}$ for grid-to-point cross-

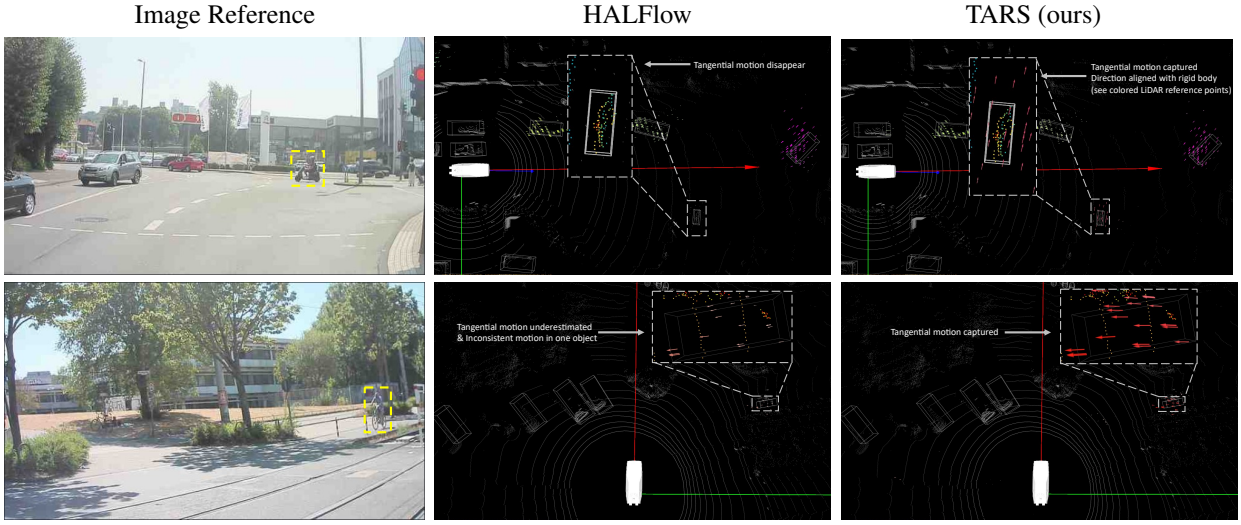

Figure 7. Qualitative results on the proprietary dataset, compared with HALFlow [18]. It illustrates two scenarios of vulnerable road users approaching the front area of the ego vehicle with tangential motion. TARS is able to capture these challenging tangential movements.

attention with the TVF.

More importantly, we leverage additional information from the OD branch, which provides object features of vulnerable road users. This helps the scene flow branch in perceiving the presence of vulnerable road users.

In Fig. 7, we show two scenarios where a person riding a bike approaches the front area of the ego vehicle from a tangential direction, which is particularly challenging to perceive by radar sensors. Our TARS successfully captures the tangential motion of the bike, while, in the outputs of the previous work, the tangential motion erroneously disappears or is underestimated.

These examples also highlights the importance of combining radar scene flow with object detection in autonomous driving. By integrating motion prediction with object bounding boxes, the system can select the optimal strategy based on the ego vehicle's speed and the direction and magnitude of target's motion.

C. Efficiency Analysis

We compare the efficiency of TARS with the previous best models on both datasets. All experiments are conducted on a single GeForce RTX 3090 GPU with a batch size of 1.

As shown in Tab. I, on the VOD dataset, our scene flow branch has 3.82M parameters, which is fewer than CMFlow's 4.23M. The total runtime per frame of TARS-ego is 84ms, 15ms higher than CMFlow's 69ms, with 10ms attributed to the scene flow branch and 5ms to the additional OD branch. Nevertheless, TARS can achieve real-time performance under a 10-Hz radar, functioning as a unified system for both object detection and scene flow estimation.

The results on the proprietary dataset are shown in Tab. J.

Table I. Efficiency analysis on the VOD dataset. The total runtime is a sum of scene flow branch (SF) and OD branch (if exist). #Param: the number of parameters in the scene flow branch.

Method	Performance (VOD)			Runtime [ms]↓			#Param ↓
	EPE [m]↓	AccS [%]↑	AccR [%]↑	Total	SF	OD	
CMFlow [7]	0.130	22.8	53.9	69	69	/	4.23M
TARS-ego (ours)	0.092	39.0	69.1	84	79	5	3.82M

Table J. Efficiency analysis on the proprietary dataset. The total runtime is a sum of scene flow branch (SF) and OD branch.

Method	Performance (proprietary)			Runtime [ms]↓			#Param ↓
	MEPE [m]↓	AccS [%]↑	AccR [%]↑	Total	SF	OD	
HALFlow [18]	0.170	50.9	63.8	77	77	/	1.00M
TARS (ours)	0.069	69.8	86.8	102	96	6	1.60M

Compared to the previous best model, i.e., HALFlow, TARS adds 0.6M additional parameters. The total runtime of TARS is 102ms per frame, 25ms higher than HALFlow's 77ms, with 19ms from the scene flow branch and 6ms from the additional OD branch. On the proprietary dataset, TARS operates at the edge of the real-time criterion and could achieve a real-time dual-task system with slight runtime optimization (*e.g.* using FP16).

TARS has a higher runtime on the proprietary dataset compared to the VOD dataset. This results from the significantly larger number of radar points per frame (∼6K vs. 256 points), and scene flow estimation operates on two input point clouds at the same time. On the VOD dataset, the increase in parameters is mainly due to the CNN layers for adapting the OD features in the TVF encoder.

D. Evaluation Metrics & Loss Functions

D.1. Evaluation Metrics

In this section, we introduce details of the evaluation metrics used in the experiments. Our input consists of two point clouds P and Q. Let $\hat{\mathbf{f}}_i$ and $\mathbf{f}_i$ represent the GT and predicted scene flow for a point p_i, respectively. N_{fg} and N_{bg} denote the number of moving points and static points. The subscripts fg or bg indicate points belonging to moving or static point set.

On the VOD dataset, we follow the evaluation metrics used in CMFlow [7]:

- EPE $= \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{f}_i - \hat{\mathbf{f}}_i\|_2$,
- AccS $= \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(\text{EPE}_i < 0.05)$,
- AccR $= \frac{1}{N}\sum_{i=1}^{N} \mathbb{I}(\text{EPE}_i < 0.1)$,
- RNE $= \frac{1}{N}\sum_{i=1}^{N} \text{EPE}_i / \frac{r_{R_i}}{r_{L_i}}$, RNE normalizes EPE by the resolution ratio between radar and LiDAR at each point location (independent of model predictions), in order to accommodate low-resolution radar. The average resolution ratio on VOD is 2.5. For high-resolution radar datasets, we directly report EPE & MEPE.
- MRNE $= \frac{1}{N_{\text{fg}}}\sum_{i\in P_{\text{fg}}} \text{RNE}_i$,
- SRNE $= \frac{1}{N_{\text{bg}}}\sum_{i\in P_{\text{bg}}} \text{RNE}_i$,

where $\mathbb{I}(\cdot)$ is the indicator function, which evaluates to one if and only if the condition is true, $\frac{r_R}{r_L}$ is the resolution ratio between radar and LiDAR.

On the proprietary dataset, high-resolution radars eliminate the need for RNE, and we focus on the evaluation of moving points since ego-motion compensation is applied:

- MEPE $= \frac{1}{N_{\text{fg}}}\sum_{i\in P_{\text{fg}}} \|\mathbf{f}_i - \hat{\mathbf{f}}_i\|_2$,
- MagE $= \frac{1}{N_{\text{fg}}}\sum_{i\in P_{\text{fg}}} \left| \|\mathbf{f}_i\|_2 - \|\hat{\mathbf{f}}_i\|_2 \right|$,
- DirE $= \frac{1}{N_{\text{fg}}}\sum_{i\in P_{\text{fg}}} \arccos\left(\frac{\mathbf{f}_i^T \cdot \hat{\mathbf{f}}_i}{\|\mathbf{f}_i\|_2 \|\hat{\mathbf{f}}_i\|_2}\right)$,
- AccS $= \frac{1}{N_{\text{fg}}}\sum_{i\in P_{\text{fg}}} \mathbb{I}(\text{MEPE}_i < 0.05)$,
- AccR $= \frac{1}{N_{\text{fg}}}\sum_{i\in P_{\text{fg}}} \mathbb{I}(\text{MEPE}_i < 0.1)$,
- SEPE $= \frac{1}{N_{\text{bg}}}\sum_{i\in P_{\text{bg}}} \|\mathbf{f}_i - \hat{\mathbf{f}}_i\|_2$,
- AvgEPE $= \frac{1}{2}(\text{MEPE} + \text{SEPE})$,

where AccS and AccR are computed only on moving points. MagE and DirE have not appeared in prior works. They reflect the two aspects of EPE: magnitude error and directional error. They can help reveal the sources of MEPE, *e.g.* high-speed underestimation or direction error.

D.2. Weakly-Supervised Loss Functions

In our weakly-supervised training, we employ the three self-supervised losses proposed in [6]: the soft Chamfer loss $\mathcal{L}_{\text{sc}}$, spatial smoothness loss $\mathcal{L}_{\text{ss}}$, and radial displacement loss $\mathcal{L}_{\text{rd}}$. We also use the foreground loss $\mathcal{L}_{\text{fg}}$ [7] and our background loss $\mathcal{L}_{\text{bg}}$.

On the VOD dataset, we train TARS-ego with additional cross-modal losses from CMFlow [7]: the motion segmentation loss $\mathcal{L}_{\text{seg}}$, ego-motion loss $\mathcal{L}_{\text{ego}}$, optical flow loss $\mathcal{L}_{\text{opt}}$. TARS-superego and TARS-no-ego didn't use these losses.

Let $P' = P_{\text{warp}}$ denote the resulting point cloud that warped by the predicted scene flow.

- The soft Chamfer loss $\mathcal{L}_{\text{sc}}$ minimizes distances between nearest points between P_{warp} and Q while handling outliers using probabilistic matching, formulated as:

$$\mathcal{L}_{\text{sc}} = \sum_{p'_i \in P_{\text{warp}}} \mathbb{I}(\nu(p'_i) > \delta) \left[\min_{q_j \in Q} \|p'_i - q_j\|_2^2 - \epsilon \right]_+ + \sum_{q_i \in Q} \mathbb{I}(\nu(q_i) > \delta) \left[\min_{p'_j \in P_{\text{warp}}} \|q_i - p'_j\|_2^2 - \epsilon \right]_+, \tag{15}$$

where $\nu(p)$ represents the per-point Gaussian density factor estimated using kernel density estimation, points with $\nu(p)$ below threshold δ are discarded as outliers, and the application of $[\cdot]_+ = \max(0, \cdot)$ in Eq. (15) ensures that small matching discrepancies below ϵ are ignored. For further details, we refer to RaFlow [6].

- The spatial smoothness loss $\mathcal{L}_{\text{ss}}$ enforces neighboring points to have similar flow vectors, weighted by distance to ensure spatial smoothness, formulated as:

$$\mathcal{L}_{\text{ss}} = \sum_{p_i \in P} \sum_{p_j \in \mathcal{N}_P(p_i)} k(p_i, p_j) \|\mathbf{f}_i - \mathbf{f}_j\|_2^2, \tag{16}$$

where $k(p_i, p_j) = \exp\left(-\frac{\|p_i - p_j\|_2^2}{\alpha}\right)$ is a radial basis function (RBF) kernel that weighs each neighbor point $p_j \in \mathcal{N}_P(p_i)$ based on its Euclidean distance to p_i, with α controlling the impact of the distance. All kernel weight values are normalized together using a softmax function.

- The radial displacement loss $\mathcal{L}_{\text{rd}}$ constrains the radial projection of predicted flow vectors using RRV measurements, formulated as:

$$\mathcal{L}_{\text{rd}} = \sum_{p_i \in P} \left| \frac{\mathbf{f}_i^T \cdot p_i}{\|p_i\|} - \text{RRV}_i \Delta t \right|, \tag{17}$$

where p_i denotes the 3D point coordinate, RRV_i is the RRV measurement of a point, and Δt is the time interval between two radar scans.

- The foreground loss $\mathcal{L}_{\text{fg}}^l$ uses pseudo scene flow GT $\hat{F}_{\text{fg}}$ derived from the object bounding box and tracking ID generated by an off-the-shelf LiDAR multi-object tracking model. This loss is applied for each level and formulated as:

$$\mathcal{L}_{\text{fg}}^l = \frac{1}{N_{\text{fg}}^l} \sum_{i=1}^{N_{\text{fg}}^l} \left\| \hat{\mathbf{f}}_{\text{fg}_i}^l - \mathbf{f}_{\text{fg}_i}^l \right\|_2, \tag{18}$$

where $\hat{\mathbf{f}}_{\text{fg}_i}^l$ is the pseudo GT for i-th moving point.

Table K. SOTA object detection (OD) and tracking accuracy on the nuScenes and VOD dataset.

Sensor	nuScenes SOTA OD [%]↑		VOD SOTA OD [%]↑		SOTA Tracking [%]↑		
	Method	mAP	Method	mAP	Method	Dataset	AMOTA
LiDAR	LION (NeurIPS24) [16]	**69.8**	CMFA (ICRA24) [5]	**69.6**	VoxelNeXt (CVPR23) [3]	nuScenes	**71.0**
Radar	RadarDistill (CVPR24) [2]	20.5	CMFA (ICRA24) [5]	41.8	RaTrack (ICRA24) [17]	VOD	31.5

• The background loss $\mathcal{L}_{\text{bg}}^{l}$ uses pseudo GT $\hat{F}_{\text{bg}}$ derived from ego-motion or using zero vectors if ego-motion compensated, applied for level l and formulated as:

$$\mathcal{L}_{\text{bg}}^{l} = \frac{1}{N_{\text{bg}}^{l}} \sum\nolimits_{i=1}^{N_{\text{bg}}^{l}} \left\| \hat{\mathbf{f}}_{\text{bg}_i}^{l} - \mathbf{f}_{\text{bg}_i}^{l} \right\|_2, \tag{19}$$

where $\hat{\mathbf{f}}_{\text{bg}_i}^{l}$ is the pseudo GT for i-th static point.

• The overall loss $\mathcal{L}_{\text{all}}$ is a sum of the above losses:

$$\mathcal{L}_{\text{all}} = \mathcal{L}_{\text{sc}} + \mathcal{L}_{\text{ss}} + \mathcal{L}_{\text{rd}} + \sum\nolimits_{l=1}^{L} (\mathcal{L}_{\text{fg}}^{l} + \lambda_{\text{bg}} \mathcal{L}_{\text{bg}}^{l}). \tag{20}$$

On the VOD dataset, we use additional cross-modal losses [7]: the motion segmentation loss $\mathcal{L}_{\text{seg}}$, ego-motion loss $\mathcal{L}_{\text{ego}}$, optical flow loss $\mathcal{L}_{\text{opt}}$.

• The motion segmentation loss $\mathcal{L}_{\text{seg}}$ uses the pseudo segmentation GT derived from the odometer and RRV to train a motion-segmentation head, formulated as:

$$\mathcal{L}_{\text{seg}} = \frac{1}{2} \sum_{i=1}^{N} \hat{s}_i \log(s_i) + (1 - \hat{s}_i) \log(1 - s_i), \tag{21}$$

where $\hat{s}_i \in \{0, 1\}$ is the pseudo motion segmentation GT derived from odometer and RRV, and $s_i \in [0, 1]$ is the predicted moving probability.

• The ego-motion loss $\mathcal{L}_{\text{ego}}$ uses the GT ego-motion from the odometer to train an ego-motion head (Appendix A.2), formulated as:

$$\mathcal{L}_{\text{ego}} = \frac{1}{N} \sum_{i=1}^{N} \left\| (\Omega - \Omega_{\text{pred}}) \begin{bmatrix} p_i \\ 1 \end{bmatrix} \right\|_2, \tag{22}$$

where Ω_{pred} is the predicted rigid transformation, Ω is the GT ego-motion derived from odometry.

• The optical flow loss $\mathcal{L}_{\text{opt}}$ projecting the scene flow onto an image and training with pseudo optical flow labels from an off-the-shelf optical flow model as additional supervision signal, formulated as:

$$\mathcal{L}_{\text{opt}} = \frac{1}{N_{\text{fg}}} \sum_{i=1}^{N_{\text{fg}}} D_{\text{Point-Ray}} \left(p_i + \mathbf{f}_i, \text{Ray}(p_i^{\text{proj}} + \widehat{\mathbf{f}_i^{\text{opt}}}), \theta \right), \tag{23}$$

where p_i^{proj} is the projection of point p_i on the image plane, $\widehat{\mathbf{f}_i^{\text{opt}}}$ is pseudo optical flow GT generated by a off-the-shelf image optical flow model, and $D_{\text{Point-Ray}}(\cdot)$ calculates the point-line distance between the warped 3D point and the corresponding $\text{Ray}(\cdot)$ traced from the optical flow-warped pixel. The parameter θ denotes the sensor calibration parameters. This loss is computed only for moving points. For further details, we refer to [7].

• The overall loss on the VOD dataset is formulated as:

$$\begin{aligned} \mathcal{L}_{\text{all}} =& \mathcal{L}_{\text{sc}} + \mathcal{L}_{\text{ss}} + \mathcal{L}_{\text{rd}} + \sum\nolimits_{l=1}^{L} (\mathcal{L}_{\text{fg}}^{l} + \lambda_{\text{bg}} \mathcal{L}_{\text{bg}}^{l}) \\ &+ \mathcal{L}_{\text{seg}} + \mathcal{L}_{\text{ego}} + \lambda_{\text{opt}} \mathcal{L}_{\text{opt}}, \end{aligned} \tag{24}$$

where λ_{opt} is set to 0.1 as CMFlow [7].

E. Discussion on the Joint Network

TrackFlow [11] directly derives LiDAR scene flow from object detection and tracking results. This approach is intuitive, especially as it directly uses the object-detection bounding boxes to ensure motion rigidity. However, this method is unsuitable for radar, where sparsity and noise cause a 30-50 mAP drop in object detection (OD) and a 40 AMOTA gap in tracking (see Tab. K). This leads to missing motion predictions for false negatives and frequent tracking failures, which severely limit its ability to infer radar scene flow. By contrast, our point-wise radar scene flow predicts motion per point, ensuring stable motion signals. Even if objects are undetected or noisily tracked, some moving points can still be identified, providing auxiliary information for decision-making in autonomous driving. In our design, TARS perceives traffic context by leveraging OD feature maps instead of bounding boxes, thereby reducing reliance on high OD recall.

References

[1] Nicolas Ballas, Li Yao, Chris Pal, and Aaron Courville. Delving deeper into convolutional networks for learning video representations. *arXiv preprint arXiv:1511.06432*, 2015. 4, 5, 2

[2] Geonho Bang, Kwangjin Choi, Jisong Kim, Dongsuk Kum, and Jun Won Choi. Radardistill: Boosting radar-based object detection performance via knowledge distillation from lidar features. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15491–15500, 2024. 7

[3] Yukang Chen, Jianhui Liu, Xiangyu Zhang, Xiaojuan Qi, and Jiaya Jia. Voxelnext: Fully sparse voxelnet for 3d object detection and tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21674–21683, 2023. 7

[4] Wencan Cheng and Jong Hwan Ko. Bi-pointflownet: Bidirectional learning for point cloud based scene flow estimation. In *European Conference on Computer Vision*, pages 108–124. Springer, 2022. 2, 4

[5] Jianning Deng, Gabriel Chan, Hantao Zhong, and Chris Xiaoxuan Lu. Robust 3d object detection from lidar-radar point clouds via cross-modal feature augmentation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6585–6591. IEEE, 2024. 7

[6] Fangqiang Ding, Zhijun Pan, Yimin Deng, Jianning Deng, and Chris Xiaoxuan Lu. Self-supervised scene flow estimation with 4-d automotive radar. *IEEE Robotics and Automation Letters*, 7(3):8233–8240, 2022. 2, 6, 7, 1

[7] Fangqiang Ding, Andras Palffy, Dariu M Gavrila, and Chris Xiaoxuan Lu. Hidden gems: 4d radar scene flow learning using cross-modal supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9340–9349, 2023. 2, 5, 6, 7, 1, 3, 4

[8] Fabian Duffhauss and Stefan A Baur. Pillarflownet: A real-time deep multitask network for lidar-based 3d object detection and scene flow estimation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10734–10741. IEEE, 2020. 2, 3

[9] Hehe Fan and Yi Yang. Pointrnn: Point recurrent neural network for moving point cloud processing. *arXiv preprint arXiv:1910.08287*, 2019. 5, 2

[10] Wolfgang Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, 32(5):922–923, 1976. 2, 1

[11] Ishan Khatri, Kyle Vedder, Neehar Peri, Deva Ramanan, and James Hays. I can't believe it's not scene flow! In *European Conference on Computer Vision*, pages 242–257. Springer, 2025. 3, 4, 7

[12] 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*, 2025. 2, 7, 3

[13] Yair Kittenplon, Yonina C Eldar, and Dan Raviv. Flowstep3d: Model unrolling for self-supervised scene flow estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4114–4123, 2021. 2, 6, 7, 4

[14] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12697–12705, 2019. 3, 4

[15] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. Flownet3d: Learning scene flow in 3d point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 529–537, 2019. 1, 2, 4

[16] Zhe Liu, Jinghua Hou, Xinyu Wang, Xiaoqing Ye, Jingdong Wang, Hengshuang Zhao, and Xiang Bai. Lion: Linear group rnn for 3d object detection in point clouds. *Advances in Neural Information Processing Systems*, 37:13601–13626, 2024. 7

[17] Zhijun Pan, Fangqiang Ding, Hantao Zhong, and Chris Xiaoxuan Lu. Ratrack: moving object detection and tracking with 4d radar point cloud. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4480–4487. IEEE, 2024. 7

[18] Guangming Wang, Xinrui Wu, Zhe Liu, and Hesheng Wang. Hierarchical attention learning of scene flow in 3d point clouds. *IEEE Transactions on Image Processing*, 30:5168–5181, 2021. 2, 3, 4, 7, 8, 5

[19] Yi Wei, Ziyi Wang, Yongming Rao, Jiwen Lu, and Jie Zhou. Pv-raft: Point-voxel correlation fields for scene flow estimation of point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6954–6963, 2021. 2, 4

[20] Benjamin Wilson, William Qi, Tanmay Agarwal, John Lambert, Jagjeet Singh, Siddhesh Khandelwal, Bowen Pan, Ratnesh Kumar, Andrew Hartnett, Jhony Kaesemodel Pontes, et al. Argoverse 2: Next generation datasets for self-driving perception and forecasting. *arXiv preprint arXiv:2301.00493*, 2023. 3

[21] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. Cbam: Convolutional block attention module. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018. 5, 1

[22] Wenxuan Wu, Zhi Yuan Wang, Zhuwen Li, Wei Liu, and Li Fuxin. Pointpwc-net: Cost volume on point clouds for (self-) supervised scene flow estimation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, pages 88–107. Springer, 2020. 1, 2, 3, 4, 7

[23] Qingwen Zhang, Yi Yang, Heng Fang, Ruoyu Geng, and Patric Jensfelt. DeFlow: Decoder of scene flow network in autonomous driving. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2105–2111, 2024. 2, 7, 3, 4

[24] Zaiwei Zhang, Bo Sun, Haitao Yang, and Qixing Huang. H3dnet: 3d object detection using hybrid geometric primitives. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XII 16*, pages 311–329. Springer, 2020. 3