Supplementary Materials:
Towards Unsupervised Object Detection from LiDAR Point Clouds

Lunjun Zhang **Anqi Joyce Yang** **Yuwen Xiong** **Sergio Casas**
Bin Yang† **Mengye Ren**† **Raquel Urtasun**
Waabi, University of Toronto
{lzhang, jyang, yxiong, sergio, urtasun}@waabi.ai

In this document, we first describe additional implementation details in Sec. 1. We next showcase additional quantitative results in Sec. 2 and additional qualitative results in Sec. 3. For videos of our qualitative results, visit the project website: https://waabi.ai/research/oyster.

1. Implementation Details

Detector Architecture: We use a detector similar to single-stage PIXOR detector [9]. The inputs to the detector are 3D voxelized binary LiDAR images from Bird-Eye View (BEV), with front-range region of interest (ROI) of $[0, 80]$ meters longitudinally and $[-40, 40]$ meters laterally with respect to the traveling direction of the ego vehicle. The step size of the voxel in BEV is 0.15625 meters, producing an input feature resolution of 512×512. In the z-axis of the voxels, we clip the range to be $[-1.5, 5.5]$ meters, and use a step size of 0.2 meters. We also concatenate 4 previous voxelized frames to the current frame along the channel dimension, after ego motion compensation has been applied to the LiDAR points of those 4 previous frames (assuming access to the poses of ego vehicle in the world coordinate per frame). As a result, the channel dimension for the input feature maps is $35 \times 5 = 175$. Unlike PIXOR, we do not use the point-wise reflectance values.

The detector architecture is similar to ResNet [3] with Feature Pyramid Network (FPN) [6]. The ResNet backbone has: initial stem layers which downsample the feature resolution by $2\times$, and then 3 residual stages, each of which downsamples the feature resolution by another $2\times$. The downsampling in the stem layers is done via strided `conv3x3` in the first layer of stem; the downsampling in 3 residual stages is done only inside the first residual block of each stage, via strided `conv1x1` on the residual inputs and strided `conv3x3` in the middle of a `Bottleneck` block. The 3 residual stages have $(6, 6, 4)$ blocks respectively. We use Sync BatchNorm (`SyncBN`) [4] for all normalization layers, and the weights and biases of the final `SyncBN` inside each `Bottleneck` block are initialized to be zero for better initialization. We use ReLU activations as all non-linearities. The stem layers reduce the channel dimension of feature volumes from 175 in the BEV LiDAR inputs to 64, and the `Bottleneck` layers in 3 residual stages subsequently use $(48, 64, 96)$ as the bottleneck dimensions and expand those dimensions by $4\times$ as the outputs of the ResNet backbone and the inputs to the FPN neck. Therefore, FPN inputs have ($4\times$, $8\times$, $16\times$) downsampled resolutions with channels $(192, 256, 384)$, which are combined to produce a feature map with $4\times$ downsampled resolution and 128 channels.

Detection is performed on the $4\times$ downsampled resolution from BEV. After FPN, we use separate headers for the classification branch and the regression branch. The classification branch applies 4 layers of (`conv3x3` $\rightarrow$ `SyncBN` $\rightarrow$ `ReLU`) with channel size 48 and then a `conv1x1` layer at the end; the regression branch is similar but with channel size 128. The classification branch produces binary classification logits to predict the presense of class-agnostic objects. The regression branch predicts a 6-dimensional vector $(dx, dy, \log l, \log w, \sin\theta, \cos\theta)$, where (dx, dy) represent the predicted offsets of object centers from the BEV `meshgrid`, and $(\log l, \log w)$ represent the log of the predicted physical lengths of the objects in terms of meters. The outputted values for $(\sin\theta, \cos\theta)$ are unconstrained since we only use $\theta = \texttt{atan2}(\sin\theta, \cos\theta)$ in the prediction and loss calculations. The classification branch uses focal initiailzation for the binary logits [7].

Ray-Dropping based Data Augmentations: There are two ray-dropping techniques we use as data augmentations to artifically make dense LiDAR point clouds sparser.

Given a 64-beam LiDAR and their (ordered) beam IDs, we first randomly drop points based on their beam IDs. More specifically, we randomly sample a beam drop ratio from $[1, 2, 3]$, then randomly sample a starting beam index between 0 and this beam drop ratio, and finally, only keep

†Work done at Waabi. Mengye is now at New York University.

the points whose beam IDs minus the starting beam index is a multiple of the beam drop ratio. In other words, we only keep the points inside beams that are uniformly spaced according to beam IDs, and the spacing is determined by the sampled beam drop ratio. Note that, when beam drop ratio is 1, the point clouds stay intact.

Then, we convert the 3D points from Euclidean coordinates to spherical coordinates, discretize the spherical coordinates according to a certain resolution, and randomly subsample the points with uniform spacing in the discretized spherical coordinates. More specifically, we convert the 3D points whose Euclidean norms are bigger than 0.1 into spherical coordinates ($\texttt{theta}, \texttt{phi}, \texttt{radial\_dist}$), and then randomly sample the discretization resolutions of $\texttt{theta}$ and $\texttt{phi}$ from $[600, 900, 1200, 1500]$. We then randomly sample spherical drop ratios from $[1, 2]$ (when this ratio is 1, the point clouds stay intact). We only keep the points inside evenly-spaced discretized spherical coordinates for both rows and columns, and the spacing is determined by the sampled spherical drop ratio.

We apply the two data augmentations in the sequential order above. Ray dropping is applied not only during initial bootstrapping to help the near-range detector generalize to longer range, but during later rounds of self-training as well when training is directly done on full range.

Point Clustering and Bounding Box Fitting: For point clustering, we first remove the points outside the detection ROI. Then, we try to remove the ground points based on a fitted ground plane. To obtain a robust estimate of the ground plane, we perform the first round of DBSCAN [1] point clustering to get the 50-th percentile z-axis (height) of non-outlier points, and then only fit a linear plane on the points below this height. (On Argoverse 2, we change the 50-th percentile to 30-th percentile.) For DBSCAN, we use $\texttt{eps} = 0.4$ and $\texttt{min\_samples} = 8$. The plane fitting module uses Random sample consensus (RANSAC) algorithm [2] with a minimum sample size of 1000 to fit a linear plane. To obtain a conservative estimate of the ground, we apply two rounds of ground fitting, with a small negative offset applied to the z-axis proportionally and all the points above the fitted plane removed after the first round and before the second round of fitting, similar to [10]. We then remove the points below this fitted plane, and then do DBSCAN clustering again with the same parameters. For each cluster, we fit a tightest bounding box using the algorithm described in [11]. For persistence-based clustering [10], we first count the point-wise number of neighbors with maximum threshold $0.5m$ among the adjacent 5 past frames (including the current frame), and then construct a KNN-based neighborhood graph with maximum number of neighbors equal to 70 and the distance metric being L1 distance between ephemerality scores [10]. DBSCAN for persistence-based clustering uses parameters $\texttt{eps} = 0.1$ and $\texttt{min\_samples} = 10$. We remove the boxes that either have a BEV area of less than $0.4m^2$, or have a length l or width w bigger than $15m$. We use *the same* ground fitting, point clustering, and boudning box fitting algorithms for our method and all our baselines.

The near-range training in the initial bootstrapping phase uses an RoI of $[0, 40]$ meters longitudinally and $[-20, 20]$ meters laterally w.r.t the traveling direction of the ego vehicle, since the point clustsers and their fitted bounding boxes tend to have higher quality in this near-range RoI.

Unsupervised Tracking: We employ a simple online tracker. For each new frame at time step t with detections $\mathbf{B}_t = \{\mathbf{b}_t^l\}$ where each $\mathbf{b}_t^l = (x_t^l, y_t^l, l_t^l, w_t^l, \theta_t^l) \in \mathbb{R}^5$ is the individual 2D BEV bounding box, we compute a cost matrix with existing tracklets $\mathbf{S}_t = \{\mathbf{s}_t^j\}$ as follows. For each tracklet j, we first predict its bbox position (x_t^j, y_t^j) at time t: if the tracklet has at least two past frames, we set $(x_t^j, y_t^j) = 2 * (x_{t-1}^j, y_{t-1}^j) - (x_{t-2}^j, y_{t-2}^j)$ via naive extrapolation (assuming constant velocity between two adjacent frames); otherwise we simply set $(x_t^j, y_t^j) = (x_{t-1}^j, y_{t-1}^j)$. Then, for each pair of the detected bbox $\mathbf{b}_t^l$ and the predicted tracklet bbox $\mathbf{b}_t^j$, we compute the Euclidean distance between the bbox centroids as $\ell^{j,l} = \sqrt{(x_t^j - x_t^l)^2 + (y_t^j - y_t^l)^2}$. For each existing tracklet, we simply employ a greedy strategy to find the nearest detection $l^* = \arg\min_l \ell^{j,l}$, and if the closest distance l^{j,l^*} is greater than a threshold of 5.0m, then the tracklet has no match. We use greedy matching instead of a more sophisticated matching strategy such as Hungarian matching because it is more robust to noisy and spurious detections.

If a tracklet is matched to a new detection, we add the detection to the tracklet and update the tracklet score $c_t^j = \frac{w \cdot c_{t-1}^j + c_t^l}{w + 1.0}$, where c_{t-1}^j is the old tracklet score, $c_t^l = 1.0$ is the detection confidence score we set for every new detection, and $w = \sum_{i=1}^{n_{t-1}^j} 0.9^i$ where n_{t-1}^j is the number of tracking steps in the tracklet.

If a tracklet is not matched, we grow the tracklet by naively extrapolating the position and angle, and set the new confidence score as $c_t^j = 0.9 c_{t-1}^j$.

If a new detection is not matched to any tracklet, we start a new tracklet with confidence score $c_t^j = 0.9$. We set the starting confidence score as 0.9 instead of 1.0 so that the longer tracklet will survive the subsequent NMS due to its higher confidence score.

We terminate all tracklets with a tracking confidence score less than 0.1, and apply NMS at the end over all existing tracklets in the current frame with an IoU threshold of 0.1. We repeat this process for the next frame at time $t + 1$ until the end of the sequence.

As mentioned in the main paper, unsupervised tracking is done in both the forward and the reverse direction of time, to compute the temporal consistency score of a given pseudo-label; pseudo-labels with low consistency scores are ignored. This procedure is applied in every round of self-training and the initial bootstrapping phase as well.

Training Details: The training loss for the detector is a combination of the focal loss [7] and the rotation-robust generalized IoU (rgiou) loss [12]. The focal loss uses an `alpha` of 0.5 and a `gamma` of 2.0; the losses for positive labels and negative labels are computed and summed separately, both normalized by the number of positive labels. To determine which pixels on the feature map (note that the feature map has a $4\times$ downsampled resolution) count as positive or negative, we first compute the axis-aligned intersection-over-union (IoU) for all pixels in the output feature map with respect to every object (in other words, the IoU is calculated between a point and a box with different centers but aligned sizes and yaws). For every object:

- If there exist pixels with this IoU value bigger than 0.5, then we randomly sample one such pixel (per object) to apply the losses on (positive focal loss + rgiou loss), and do not apply any losses on other pixels where this IoU value is also bigger than 0.5;
- Otherwise, if there are no pixels with this IoU value bigger than 0.5, then we select the pixel with the highest IoU value (if the highest value is zero, then this label is ignored) to apply the losses on, and do not apply any losses on other pixels where this IoU value is smaller than the selected pixel but bigger than 0.3.
- For all the pixels that are selected as *positive*, apply positive focal loss + rgiou loss; for all the pixels that are not selected as *positive* and not set to be *ignored*, apply negative focal loss.

We use Adam optimizer [5] with decoupled weight decay [8], with a learning rate of 0.004 and a weight decay coefficient of 0.0001. For PandaSet, we train the detector for 20 epochs without learning rate decay. For Argoverse 2, we train on a $10\times$ subsampled dataset (meaning that for each sequence in the training set, we subsample the number of frames by $10\times$ per sequence to speed up training), and train for 40 epochs, with the learning rate decreased by $10\times$ at the 20-th epoch. On both datasets, we use a batch size of 8 per GPU, and apply distributed training with 4 GPUs.

We use non-maximum suppression (NMS) to generate the final detections. Prior to NMS, we first take the top 1000 scored detections per frame. We apply NMS with a threshold of 0.1. After NMS, we take the top 100 scored detections per frame. For pseudo-label generation of the next self-training round, We use a score threshold of 0.4 after NMS.

2. Additional Quantitative Results

Detailed breakdown of near-range and far-range metrics: To showcase the effect of near-range training and zero-shot extension to far-range and training with ray drop, we additionally break down the [0, 80]m metrics in the first three rows of Tab. 3 in the main paper into near-range ($[0, 40)$m) and far-range ($[40, 80]$m). As shown in Tab. 1, by training in the near range with ray dropping, the detection metrics for far-range objects significantly improve.

Measuring the gap between unsupervised and supervised methods: Tab. 2 shows a comparison between our method and multiple supervised counterparts, trained with varying levels of supervision. We evaluate with all classes (vehicle, pedestrian, cyclist) combined, in a class-agnostic manner. The table shows that despite the advancement of unsupervised object detection made in this work, the remaining gap between unsupervised learning and supervised learning for object detection remains large in the context of self-driving. More specifically, our method *OYSTER* surpasses supervised learning on a single log and starts to match its supervised counterpart on 2 logs; however, compared to the performance of supervised learning on frames sub-sampled from the whole dataset (in other words, from all the logs), our method is not yet as competitive.

Performance vs. number of self-training iterations: We additionally study how the number of self-training iterations affects unsupervised detection performance. Fig. 1 showcases IoU and DTC metrics with respect to the number of self-training iterations on both Pandaset and AV2. Note that in the main paper we apply long tracklet refinement only in the last iteration of the Pandaset experiments. For a more fair comparison, in this ablation study we do not apply the long tracklet refinement in the last iteration so that each Pandaset self-training iteration employs the same pseudo-label post-processing process. On the other hand, similar to the main paper, we apply full refinement (both long tracklet refinement and short tracklet filtering) in every AV2 self-training iteration. Our results show that overall, for both datasets, more self-training iterations improved almost all metrics, with the exception that DTC-based recall dropped slightly with the third self-training iteration. Therefore, we stop the self-training loop at iteration 3 for Pandaset and iteration 2 for AV2. We leave DTC recall improvement with more self-training iterations as future work.

3. Additional Qualitative Results:

Qualitative results of ours vs. baselines: Fig. 2 and Fig. 3 show additional qualitative results comparing our method and several baselines [1, 10] on Pandaset and AV2. Our method is able to identify stationary actors and output

	ID	ITR	RD	AP @ IoU			Recall @ IoU			AP @ ΔDTC			Recall @ ΔDTC		
				0.3	0.5	0.7	0.3	0.5	0.7	1.5	1.0	0.5	1.5	1.0	0.5
$[0, 40)$m	M_1	80		39.7	23.9	**12.7**	62.7	**43.6**	**25.1**	52.8	50.6	45.4	79.5	77.1	70.7
	M_2	40		42.7	25.6	10.6	**64.5**	43.1	22.8	55.1	52.7	46.7	**80.4**	**78.0**	**71.5**
	M_3	40	✓	**44.0**	**27.0**	7.6	61.7	39.1	17.0	**56.7**	**54.1**	**47.0**	77.5	74.8	67.4
$[40, 80]$m	M_1	80		2.1	0.3	0.0	19.5	7.4	2.0	25.0	23.5	18.7	61.6	59.1	51.1
	M_2	40		2.5	0.5	0.1	19.5	7.2	2.2	22.7	20.7	16.1	61.0	58.2	50.2
	M_3	40	✓	**6.7**	**1.3**	**0.3**	**25.8**	**9.1**	**2.6**	**30.7**	**28.2**	**22.0**	**64.3**	**61.1**	**52.0**

Table 1. **[Pandaset] Ablation study with ranges**. All class evaluation in the range 0-40m (top) and 40-80m (bottom). Legend: ID=Model identifier, ITR=Initial training range (first iteration) from 0 to X meters, RD=Ray-dropping.

Supervision	AP @ IoU			Recall @ IoU			AP @ ΔDTC			Recall @ ΔDTC		
	0.3	0.5	0.7	0.3	0.5	0.7	1.5	1.0	0.5	1.5	1.0	0.5
1 log (80 frames)	36.8	30.7	18.9	46.9	36.4	23.2	48.6	45.9	37.9	65.1	61.4	52.5
2 logs (160 frames)	51.9	45.0	29.8	61.9	52.0	35.0	56.9	54.5	46.5	69.9	67.0	59.3
80 random frames	70.6	61.0	41.1	80.1	67.6	46.1	75.6	73.3	65.8	87.3	85.0	78.1
160 random frames	81.3	74.2	55.4	87.1	78.6	59.7	84.3	83.1	77.1	91.3	90.0	84.8
DBSCAN [1] (unsupervised)	3.5	1.1	0.3	28.6	15.9	8.3	10.9	9.9	8.0	50.9	48.3	43.1
MODEST [10] (unsupervised)	22.8	7.5	2.8	49.7	28.9	14.9	38.8	36.4	30.2	70.2	66.9	59.0
Ours (unsupervised)	43.5	29.5	18.1	62.8	44.8	28.1	33.8	44.3	48.7	54.4	67.0	72.3

Table 2. **[Pandaset] Comparison against few-shot supervised methods**. All-class evaluation on the range 0-80m.

detections with improved sizes and positions. Please also refer to the supplementary video for more results.

Qualitative results of refinement with self-training: Fig. 4 showcases the qualitative pseudo-labels after one iteration of self-training on refined labels. We show the model output after the second iteration of self-training on Pandaset, the long tracklet refinement results on the pseudo-labels, followed by short tracklet filtering and a self-training iteration with the refined pseudo-labels. Note that the long tracklet refinement step effectively improves the bounding box sizes for some actors, and the short tracklet filtering is able to identify a few false positives and ignores them during the re-training. With an additional round of self-training, more labels are discovered and the bounding box sizes are generally improved.

Failure cases: We additionally point to various failure cases in different stages of our pipeline, including seed labels, temporal-based filtering and iterative self-training. First of all, our initial seed labels (produced by our baseline DBSCAN) have many failure cases showcased in Fig. 6 of the main paper and Fig. 2 and Fig. 3 of the supplementary. The DBSCAN labels in the left column of the figures contain false positives and missed detections. Note that thanks to our self-training iterations, early-stage missed detections are discovered, as shown in Fig. 6 in the main paper. However, there is a limit to what self-training can do. In the second row of Fig. 5 of the main paper, our method still misses detections from DBSCAN in the top left corner, even though it discovers many objects on the right side of the image. As for temporal consistency, in the third row of Fig. 4 in the supplementary, temporal-consistency based short tracklet filtering incorrectly filters out some cars in the parking lot.

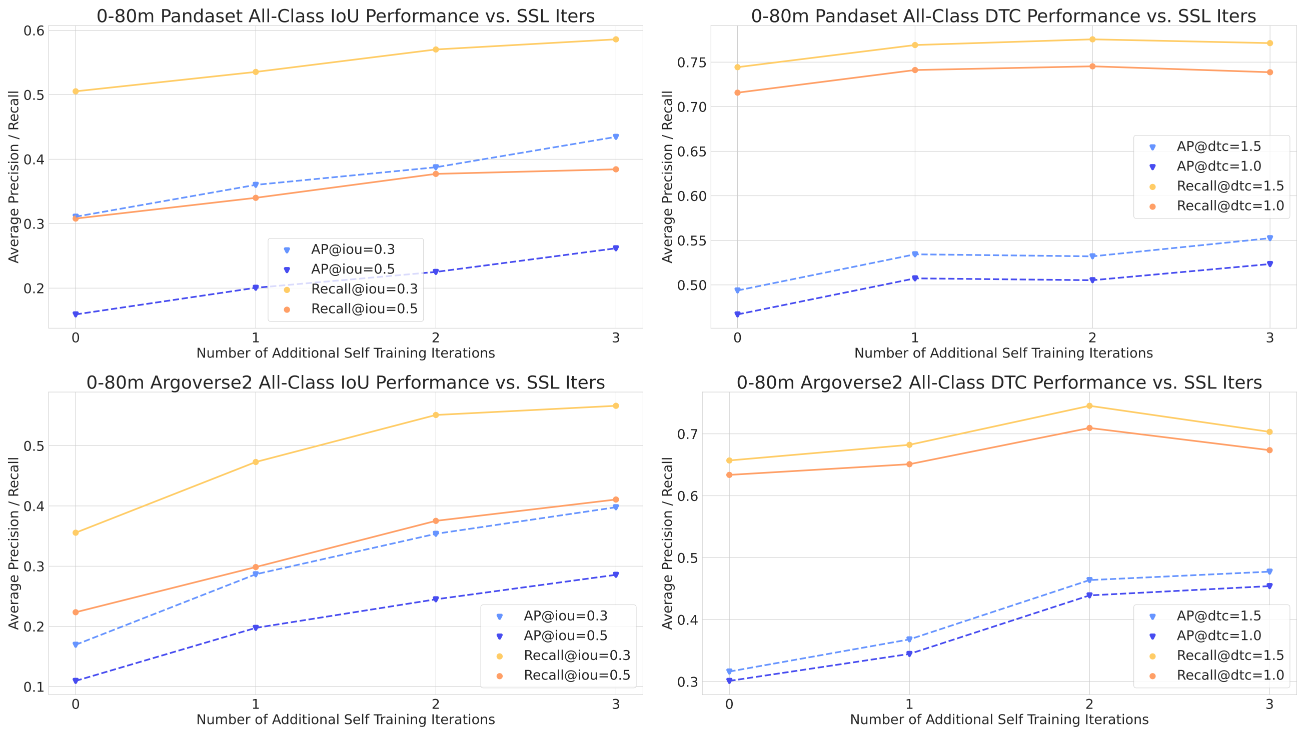

Figure 1. **Performance vs. number of self-training iterations**. We showcase the IoU (left column) and DTC (right column) mean AP and recall metrics with respect to the number of self-training iterations after the initial round. We denote the initial round results as iteration 0 in the x-axis. For both Pandaset (top row) and Argoverse (bottom row) results, more self-training iterations improved almost all metrics, while DTC-based recall dropped slightly with the third self-training iteration.

References

[1] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pages 226–231, 1996. 2, 3, 4

[2] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 2

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 1

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

[6] 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

[7] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 1, 3

[8] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 3

[9] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2018. 1

[10] Yurong You, Katie Luo, Cheng Perng Phoo, Wei-Lun Chao, Wen Sun, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Learning to detect mobile objects from lidar scans without labels. In *CVPR*, 2022. 2, 3, 4

[11] Xiao Zhang, Wenda Xu, Chiyu Dong, and John M Dolan. Efficient l-shape fitting for vehicle detection using laser scanners. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 54–59. IEEE, 2017. 2

[12] Yu Zheng, Danyang Zhang, Sinan Xie, Jiwen Lu, and Jie Zhou. Rotation-robust intersection over union for 3d object detection. In *European Conference on Computer Vision*, pages 464–480. Springer, 2020. 3

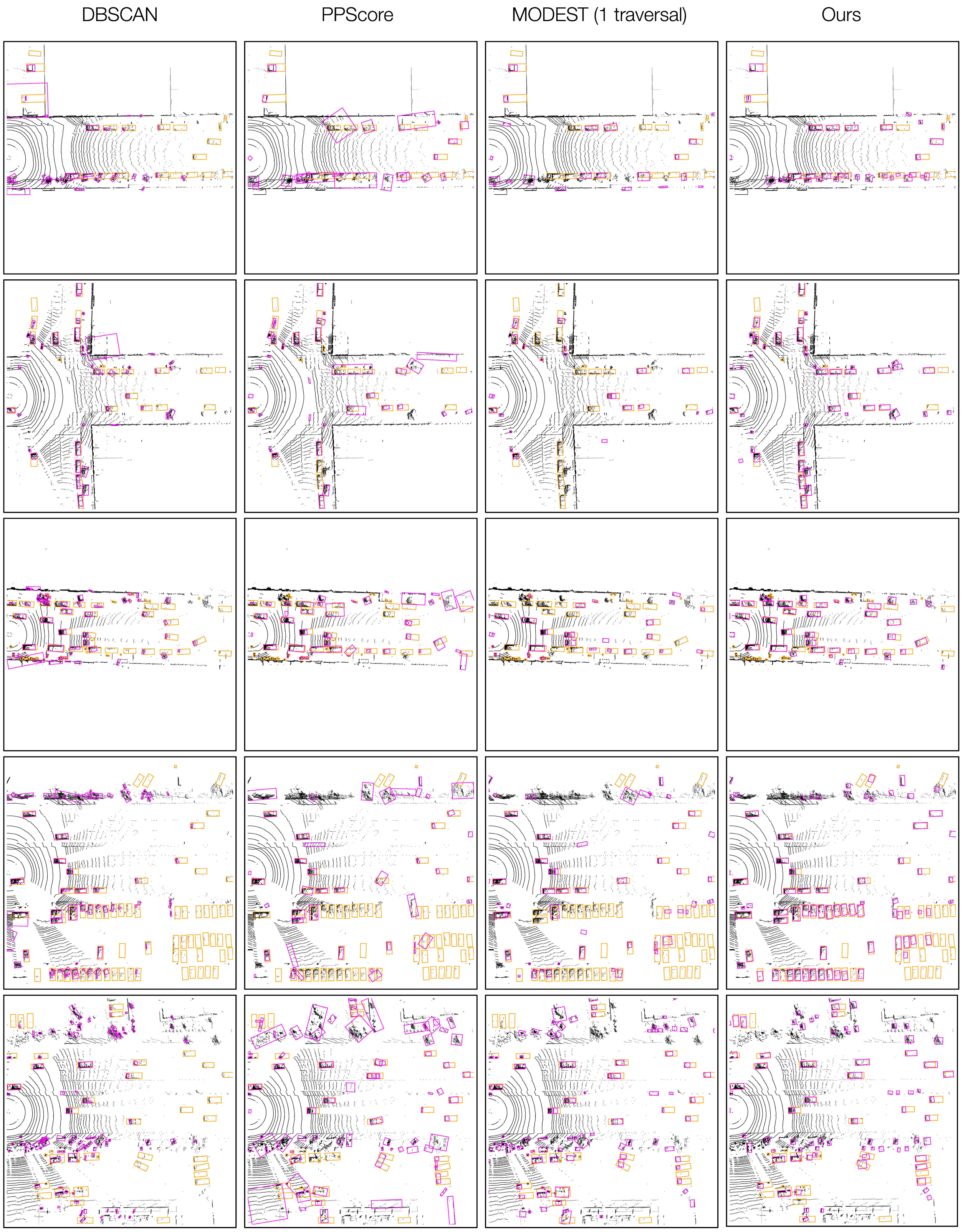

Figure 2. **[Pandaset] Additional qualitative results** comparing ours with baselines on frames in the validation split. We draw the detections in purple and ground-truth bounding boxes in orange.

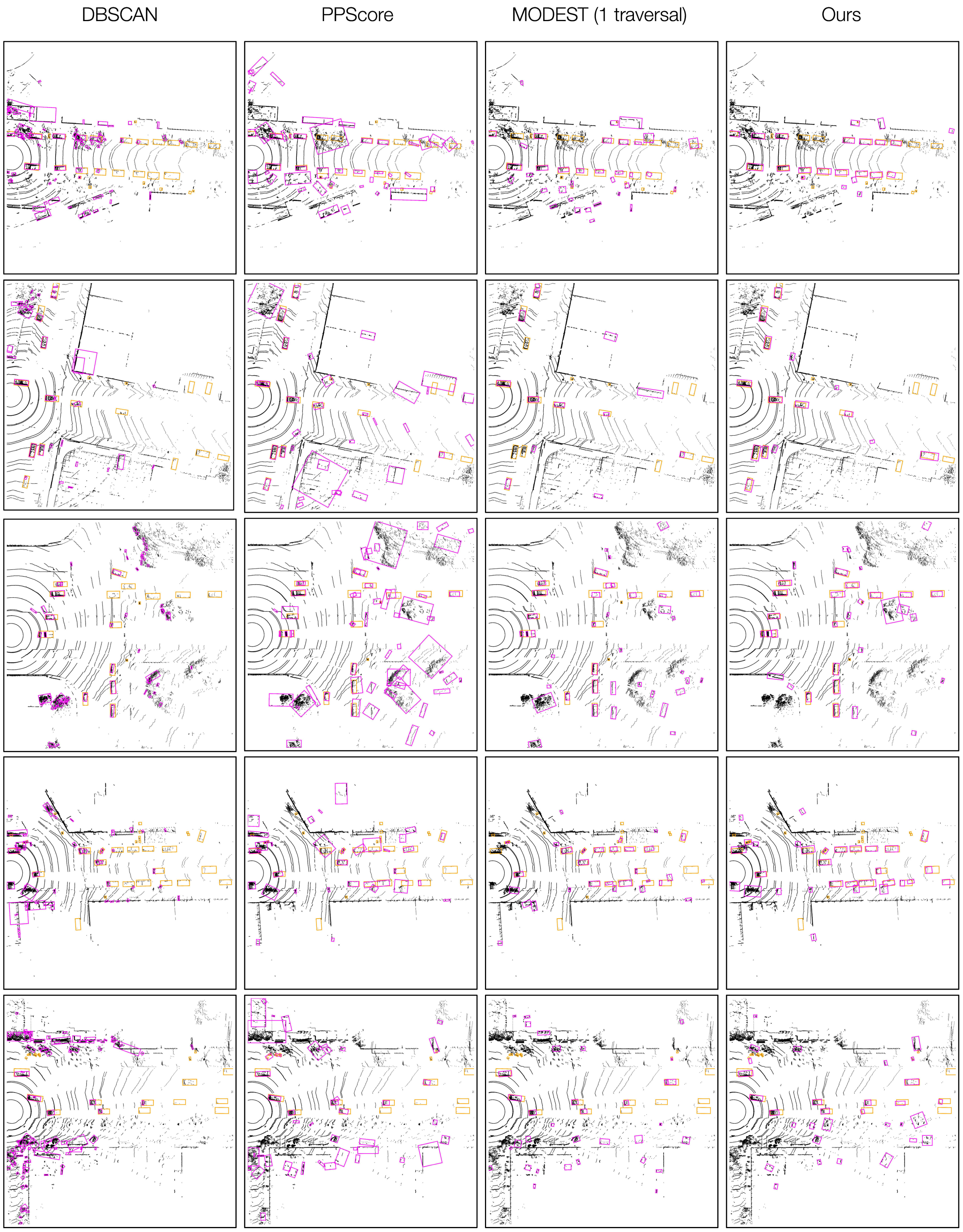

Figure 3. **[AV2] Additional qualitative results** comparing ours with baselines on frames in the validation split. We draw the detections in purple and ground-truth bounding boxes in orange.

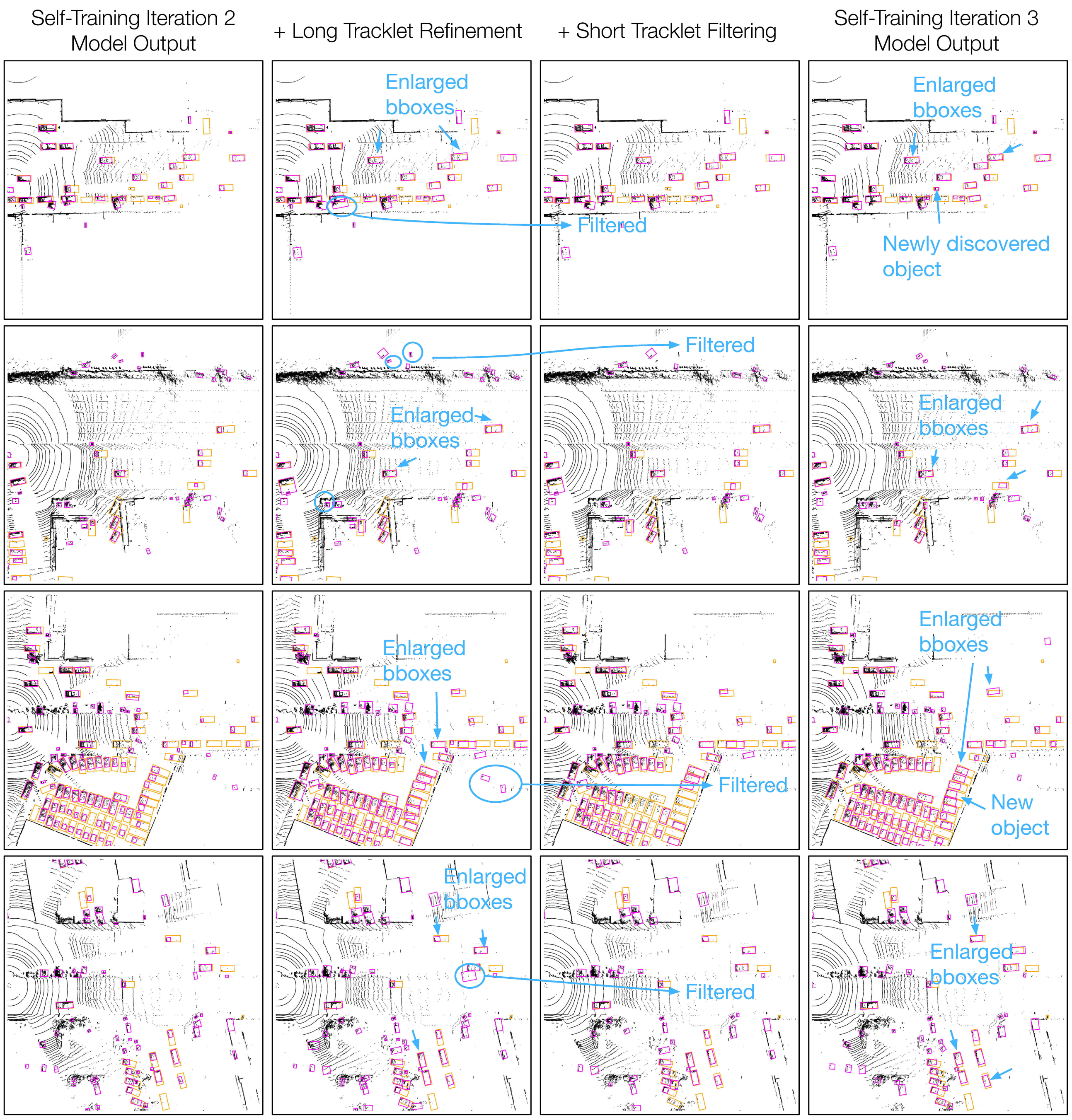

Figure 4. **[Pandaset] Qualitative results before and after self-training with refinement**. We take the model output from self-training iteration 2 on the left, apply long tracklet refinement (middle left) and short tracklet filtering (middle right), and re-train with the refined pseudo-labels, leading to new model output on the right. The new detections have larger bbox sizes and discover new objects as a result of the long tracklet refinement and self-training. We draw the detections in purple and ground-truth bounding boxes in orange.