

Offboard 3D Object Detection from Point Cloud Sequences

Supplementary Material

Charles R. Qi Yin Zhou Mahyar Najibi Pei Sun Khoa Vo Boyang Deng Dragomir Anguelov
Waymo LLC

A. Overview

In this document, we provide more details of models, experiments and show more analysis results. Sec. B presents more evaluation results on the Waymo Open Dataset test set and shows how our offboard 3D detection can help domain adaptation and 3D tracking. Sec. C explains more details of MVF++ detectors. Sec. D and Sec. E describe implementation details of our multi-object tracker and track-based motion state classifier respectively. Sec. F covers network architectures, losses and training details of object auto labeling models. Sec. G describes the specifics of the human label study for 3D object detection and provides more statistics. Sec. H provides more information about the semi-supervised learning experiments. Lastly Sec. I gives more analysis results supplementary to the main paper.

B. More Evaluation Results

B.1. 3D Detection Results on the Test Set

In Table. 1 we report detection results on the Waymo Open Dataset *test* set comparing our pipeline with a few leading methods in the leaderboard [1]. Note that our pipeline achieves the best results among Lidar-only methods. It also outperforms the HorizonLidar3D which uses both camera and Lidar input in the L1 metrics. We expect that adding camera input to our pipeline can further improve our pipeline in hard cases (L2).

Method	Sensor	AP L1	APH L1	AP L2	APH L2
PV-RCNN	L	81.06	80.57	73.69	73.23
CenterPoint	L	81.05	80.59	73.42	72.99
HorizonLidar3D	CL	85.09	84.68	78.23	77.83
3DAL (ours)	L	85.84	85.46	77.24	76.91

Table 1. **3D detection AP on the Waymo Open Dataset main *test* set for vehicles.** Evaluation results were obtained from submitting to the test server. For the sensor the ‘L’ means Lidar-only; the ‘CL’ means camera and Lidar. Note that our method peeks into the future for object-centric refinement, which is feasible in the offboard setting.

Method	3D AP	0-30m	30-50m	50+m
PointPillar	45.48	74.02	36.49	14.94
Multi-frame MVF++	70.01	86.54	67.72	43.25
PV-RCNN-DA	71.40	90.00	66.45	45.92
CenterPoint	67.04	86.62	60.95	38.59
HorizonLidar3D	72.48	90.65	67.26	47.89
3DAL (ours)	78.04	91.90	73.47	52.53

Table 2. **3D detection AP on the Waymo Open Dataset domain adaptation *test* set for vehicles.** The PointPillar, MVF++ and 3DAL models were trained by us on the Waymo Open Dataset main *train* set. Evaluation results were obtained from submitting to the test server. The PV-RCNN-DA, CenterPoint and HorizonLidar3D results are leading entries from the leaderboard [2].

B.2. Domain Adaptation Results

In Table 2 we report detection results in another domain and compare our 3D Auto Labeling (3DAL) pipeline with two baselines: the popular PointPillars [5] detector and our offboard multi-frame MVF++ detector. We see that our 3D Auto Labeling pipeline achieves significantly higher detection APs compared to the baselines (**32.56** higher 3D AP than the PointPillars and **8.03** higher 3D AP than the multi-frame MVF++), showing the strong generalization ability of our models. Compared to a few leading methods on the leaderboard [2] our method also shows significant gains. These large gains are probably due to the temporal information aggregation, which compensates the lower point densities in the WOD domain adaptation set (collected in Kirkland with mostly rainy weather).

B.3. 3D Tracking Results

In table 3 we show how our improved box estimation from the offboard 3D Auto Labeling enhances the tracking performance, compared to using the boxes from the single-frame or multi-frame detectors. All methods used the same tracker (Sec. D). This reflects that in the tracking-by-detection paradigm, the localization accuracy plays an important role in determining tracking quality in terms of MOTA and MOTP.

Method	MOTA \uparrow	MOTP \downarrow
Single-frame MVF++ with KF	52.20	17.08
Multi-frame MVF++ with KF	61.92	16.31
3D Auto Labeling	66.90	15.45

Table 3. **3D tracking results for vehicles on the Waymo Open Dataset *val* set.** The metrics are L1 MOTA and MOTP for vehicles on the Waymo Open Dataset *val* set. KF stands for using Kalman Filtering for the track state update. The arrows indicate whether the metric is better when it is higher (up-ward arrow) or is better when it is lower (down-ward arrow).

C. Implementation Details of the MVF++ Detectors

Network Architecture Figure 1 illustrates the point-wise feature fusion network within the proposed MVF++. Given C -dimensional input encoding of N points [14], the network first projects the points into a 128-D feature space via a multi-layer perceptron (MLP), where shape information can be better described. The MLP is composed of a linear layer, a batch normalization (BN) layer and a rectified linear unit (ReLU) layer. Then it processes the features by two separate MLPs for view-dependent information extraction, *i.e.* one for the Bird’s Eye View and one for the Perspective View [14]. Next, the network employs voxelization [14] to transform view-dependent point features into the corresponding 2D feature maps, which are fed to view-dependent ConvNets (*i.e.* ConvNet_b and ConvNet_p) to further extract contextual information within an enlarged receptive field. Different from MVF [14] using one ResNet [3] layer in obtaining each down-sampled feature maps, we increase the depth of ConvNet_b and ConvNet_p by applying one more ResNet block in each down-sampling branch. At the end of view-dependent processing, it applies devoxelization to transform the 2D feature map back to point-wise features. The model fuses point-wise features by concatenating three sources of information. To reduce computational complexity, it applies two MLPs consecutively, reducing the feature dimension to 128. For improving the discriminative capability of features, it introduces 3D segmentation auxiliary loss and augments the dimension-reduced features with segmentation features. The output of point-wise feature fusion network has shape $N \times 144$.

Upon obtaining point-wise features, we voxelize them into a 2D feature map and employ a backbone network to generate detection results. Specifically, we adopt the same architecture as in [5, 14]. To further boost detection performance in the offboard setting, we replace each plain convolution layer with a ResNet [3] layer maintaining the same output feature dimension and feature map resolution.

Loss Function We train MVF++ by minimizing a loss function, defined as $L = L_{\text{cls}} + w_1 L_{\text{centerness}} + w_2 L_{\text{reg}} + w_3 L_{\text{seg}}$. L_{cls} and $L_{\text{centerness}}$ are focal loss and centerness loss as in [11]. L_{reg} represents Smooth L1 loss learning to regress x , y , z center locations, length, width, height and heading orientation at foreground pixels, as in [5, 14]. L_{seg} is the auxiliary 3D segmentation loss for distinguishing foreground from background points (points are labeled as foreground/background if they lie inside/outside of a ground truth 3D box) [9, 6]. In our experiments, we set $w_1 = 1.0$, $w_2 = 2.0$, $w_3 = 1.0$. At inference time, the final score for ranking all detected boxes is computed as the multiplication of the classification score and the centerness score. By doing so, the centerness score can downplay the boxes far away from an object center and thus encourage non-maximum suppression (NMS) to yield high-quality boxes, as recommended in [11].

Data Augmentation We perform three global augmentations that are applied to the LiDAR point cloud and ground truth boxes simultaneously [15]. First, we apply random flip along the x axis, with probability 0.5. Then, we employ a random global rotation and scaling, where the rotation angle and the scaling factor are randomly drawn uniformly from $[-\pi/4, +\pi/4]$ and $[0.9, 1.1]$, respectively. Finally, we add a global translation noise to x , y , z drawn from $\mathcal{N}(0, 0.6)$.

Hyperparameters For vehicles, we set voxel size to $[0.32, 0.32, 6.0]$ m and detection range to $[-74.88, 74.88]$ m along the X and Y axes and $[-2, 4]$ m along Z axis, which results in a 468×468 2D feature map in the Bird’s Eye View. For pedestrians, we set voxel size to $[0.24, 0.24, 4.0]$ m and detection range to $[-74.88, 74.88]$ m along the X and Y axes and $[-1, 3]$ m along Z axis, which corresponds to a 624×624 2D feature map in the Bird’s Eye View. During test-time augmentation, we set (IoU threshold, box score) to be (0.275, 0.5) for vehicles and (0.2, 0.5) for pedestrians, to trigger weighted box fusion [10].

Training During training, we use the Adam optimizer [4] and apply cosine decay to the learning rate. The initial learning rate is set to 1.33×10^{-3} and ramps up to 3.0×10^{-3} after 1000 warm-up steps. The training used 64 TPUs with a global batch size of 128 and finished after 43,000 steps.

D. Implementation Details of the Tracker

Our multi-object tracker is a similar implementation to [13]. To reduce the impact of sensor ego-motion in tracking, we transformed all the boxes to the world coordinate for tracking. To reduce false positives, we also filter out all detections with scores less than 0.1 before the tracking. We used Bird’s Eye View (BEV) boxes for detection and

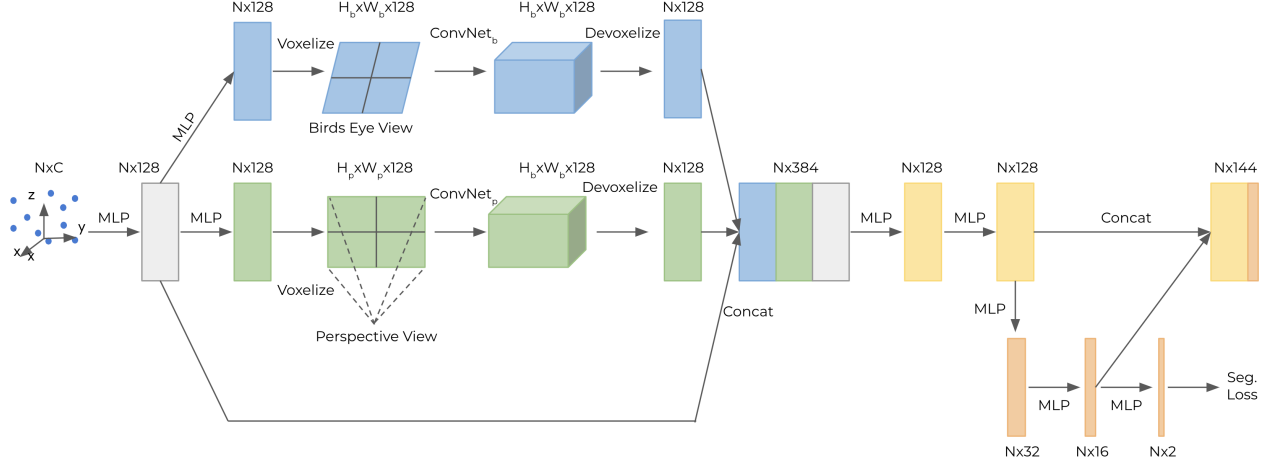


Figure 1. **Point-wise feature fusion network of MVF++**. Given an input point cloud encoding of shape $N \times C$, the network maps it to high-dimensional feature space and extracts contextual information from different views *i.e.* the Bird’s Eye View and the Perspective View. It fuses view-dependent features by concatenating information from three sources. The final output has shape $N \times 144$, as a result of concatenating dimension-reduced point features of shape $N \times 128$ with 3D segmentation features of shape $N \times 16$.

track association, using the Hungarian algorithm with an IoU threshold of 0.1. During the states update, the heading is handled specially as there can be flips and cyclic patterns. Before updating the heading state, we first adjust the detection heading to align with the track state heading – if the angle difference is obtuse, we add π to the detection angle before the update; we also average the angles in the cyclic space (*e.g.* the average of 6 rad and 0.5 rad is 0.1084 rather than 3.25).

E. Implementation Details of the Motion State Estimator

As we introduced in the main paper, we use the object track data for motion state estimation, which is much easier compared to classifying the static/non-static state from a single or a few frames. Note that we define an object as static only if it is stationary in the entire sequence. Specifically, we extract two heuristic-based features and fit a linear classifier to estimate the motion state. The two features are: the detection box centers’ variance and the begin-to-end distance of the tracked boxes (the distance from the center of the first box of the track to the center of the last box of the track), with boxes all in the world coordinate. To ensure that the statistics are reliable we only consider tracks with at least 7 valid measurements. For tracks that are too short, we do not run the classification nor the auto labeling models. The boxes of those short tracks are merged directly to the final auto labels.

The ground truth motion states are computed from ground truth boxes with pre-defined thresholds of begin-to-end distance (1.0m) and max speed (1m/s). The thresholds are needed because there could be small drifts in sensor poses, such that the ground truth boxes in the world coordi-

nate are not exactly the same for a static object.

For vehicles, such a simple linear model can achieve more than 99% classification accuracy. The remaining rare error cases usually happen in short tracks with noisy detection boxes, or for objects that are heavily occluded or far away. For pedestrians, as most of them are moving and even the static ones tend to move their arms and heads, we consider all pedestrian tracks as dynamic.

F. Details of the Object Auto Labeling Models

F.1. Static Object Auto Labeling

Network architecture. In the static object auto labeling model, the foreground segmentation is a PointNet [7] segmentation network, where each point is firstly processed by an multi-layer perceptron (MLP) with 5 layers with output channel sizes of 64, 64, 64, 128, 1024. For every layer of the MLP, we have batch normalization and ReLU. The 1024-dim per point embeddings are pooled with a max pooling layer and concatenated with the output of the 2nd layer of the per-point MLP (64-dim). The concatenated 1088-dim features are further processed by an MLP of 5 layers with output channel sizes 512, 256, 128, 128, 2, where the last layer does not have non-linearity or batch normalization. The predicted foreground logit scores are used to classify each point as foreground or background. All the foreground points are extracted.

The box regression network is also a PointNet [7] variant that takes the foreground points and outputs the 3D box parameters. It has a per-point MLP with output sizes of 128, 128, 256, 512, a max pooling layer and a following MLP with output sizes 512, 256 on the max pooled features. There is a final linear layer predicting the box pa-

rameters. We parameterize the boxes in a way similar to [6] as the box center regression (3-dim), the box heading regression and classification (to each of the heading bins) and the box size regression and classification (to each of the template sizes). For iterative refinement, we apply the same box regression network one more time on the foreground points transformed to the estimated box’s coordinate. We found that if we use multi-frame MVF++ boxes, using shared weights for the two box regression networks works better than not sharing the weights; while if we use the single-frame MVF++, the cascaded design without sharing the weights works better. The numbers in the main paper are from the iterative model (shared weights).

For simplicity and higher generalizability of the model, we only used XYZ coordinates of the points in the segmentation and box regression networks. Intensities and other point channels were not used. We have also tried to use the more powerful PointNet++ [8] models but did not see improvement compared to the PointNet-based models in this problem.

Losses. The model is trained with supervision of the segmentation masks and the ground truth 3D bounding boxes. For the segmentation, the sub-network predicts two scores for each point as foreground or background and is supervised with a cross-entropy loss L_{seg} . For the box regression, we implement a process similar to [6], where each box regression network regresses the box by predicting its center cx, cy, cz , its size classes (among a few pre-defined template size classes) and residual sizes for each size class, as well as the heading bin class and a residual heading for each bin. We used 12 heading bins (each bin account for 30 degrees) and 3 size clusters: (4.8, 1.8, 1.5), (10.0, 2.6, 3.2), (2.0, 1.0, 1.6), where the dimensions are length, width, height. The box regression loss is defined as $L_{box_i} = L_{c-reg_i} + w_1 L_{s-cls_i} + w_2 L_{s-reg_i} + w_3 L_{h-cls_i} + w_4 L_{h-reg_i}$ where $i \in \{1, 2\}$ represents the cascade/iterative box estimation step. The total box regression loss is $L = L_{seg} + w(L_{box_1} + L_{box_2})$. The w_i and w are hyperparameter weights of the losses. Empirically, we use $w_1 = 0.1, w_2 = 2, w_3 = 0.1, w_4 = 2$ and $w = 10$.

Training and data augmentation. We train our models using the extracted object tracks (with the proposed multi-frame MVF++ model and our multi-object tracker) from the Waymo Open Dataset for each class type separately. Ground truth boxes are assigned to every frame of the track (frames with no matched ground truth are skipped).

During training, for each static object track, we randomly select an initial box from the sequence. We also randomly sub-sample Uniform $[1, |S_j|]$ frames from all the visible frames S_j of an object j . This naturally leads to a data augmentation effect. Note that at test time we al-

ways select the initial box with the highest score and use all frames. The merged points are randomly sub-sampled to 4,096 points and randomly flipped along the X, Y axes with 50% chance respectively and randomly rotated around the up-axis (Z) by Uniform $[-10, 10]$ degrees. To increase the data quantity, we also turn the dynamic object track data to pseudo static track. To achieve that, we use the ground truth object boxes to align the dynamic object points to a specific frame’s ground truth box coordinate. This increases the number of object tracks of vehicles by 30%.

In total, we have extracted around 50K (vehicle) object tracks for training (including the augmented ones from dynamic objects) and around 10K object tracks for validation (static only). We trained the model using the Adam optimizer with a batch size of 32 and an initial learning rate of 0.001. The learning rate was decayed by 10X at the 60th, 100th and 140th epochs. The model was trained with 180 epochs in total, which took around 20 hours with a V100 GPU.

F.2. Dynamic Object Auto Labeling

Network architecture. For the foreground segmentation network, we adopt a similar architecture as that for the static auto labeling model except that the input points have one more channel besides the XYZ coordinate, the time encoding channel. The temporal encoding is 0 for points from the current frame, $-0.1r$ for the r -th frame prior to the current frame and $+0.1r$ for the r -th frame after the current frame. In our implementation we take 5 frames of object points with each frame’s points subsampled to 1,024 points, so in total there are 5,120 points input to the segmentation network. The point sequence encoder network takes the segmented foreground points and uses a PointNet [7]-like architecture with a per-point MLP of output sizes 64, 128, 256, 512, a max-pooling layer and another MLP with output sizes 512, 256 on the max-pooled features. The output is a 256-dim feature vector.

For the box sequence encoder network, we consider each box (in the center frame’s box coordinate) as a parameterized point with channels of box center (cx, cy, cz), box size (length, width, height), box heading θ and a temporal encoding. We use nearly the entire box sequence (setting s in the main paper to 50, leading to a sequence length of 101).

The box sequence can be considered as a point cloud and processed by another PointNet. Note, such a sequence can also be processed by a 1D ConvNet, or be concatenated and processed by a fully connected network, or we can even use a graph neural network. Through empirical study we found using a PointNet to encode the box sequence feature is both effective (compared with ConvNet and fully connected layers) and simple (compared with graph neural networks). The box sequence encoding PointNet has a per-point MLP with output sizes 64, 64, 128, 512, a max-pooling layer and

another MLP with output sizes 128, 128 on the max-pooled features. The final output is a 128-dim feature vector, which we call the trajectory embedding.

The point embedding and the trajectory embedding are concatenated and passed through a final box regression network, which is a MLP with two layers with output sizes 128, 128 and a linear layer to regress the box parameters (similar to that of the static object auto labeling model).

To encourage contributions from both branches, we follow [12] and also pass the trajectory embedding and the object embedding to two additional box regression sub-networks to predict boxes independently. The sub-networks have the same structure as the one for the joint-embedding, but with non-shared weights.

Losses. Similar to the static auto labeling model, we have two types of loss, the segmentation loss and the box regression loss. The box regression outputs are defined in the same way as that for the static objects. We used 12 heading bins (each bin account for 30 degrees) and the same size clusters as those for the static vehicle auto labeling. For pedestrians we use a single size cluster: (0.9, 0.9, 1.7) of length, width, height. The final loss is $L = L_{\text{seg}} + v_1 L_{\text{box-traj}} + v_2 L_{\text{box-obj-pc}} + v_3 L_{\text{box-joint}}$ where we have three box losses from the trajectory head, the object point cloud head and the joint head respectively. The $v_i, i = 1, 2, 3$ are the weights for the loss terms to achieve a balanced learning of the three types of embeddings. Empirically, we use $v_1 = 0.3, v_2 = 0.3, v_3 = 0.4$.

Training and data augmentation. During training, we randomly select the center frame from each dynamic object track. If the context size is less than the required sequence length $2r + 1$ or $2s + 1$, or when the frames in the sequence are not consecutive (e.g. the object is occluded for a few frames), we use placeholder points and boxes (all zeros) for the empty frames. As there may be tracking errors, we match our object track with ground truth tracks and avoid training on the ones with switched track IDs.

As to augmentation, both points and boxes are randomly flipped along the X and Y axis with a 50% chance and randomly rotated around the Z axis by Uniform $[-10, 10]$ degrees. We also add a light random shift and a random scaling to the point clouds. Point cloud from each frame is also randomly sampled to 1, 024 points from the full point cloud observed.

We train vehicle and pedestrian models separately. For vehicles we extracted around 15.7K dynamic tracks for training and 3K for validation. For pedestrians, we extracted around 22.9K dynamic tracks for training and around 5.1K for validation. We train the model using the Adam optimizer with batch size 32 and an initial learning rate 0.001. The learning rate is decayed by 10 times at the

```
segment-17703234244970638241.220.000.240.000
segment-15611747084548773814.3740.000.3760.000
segment-11660186733224028707.420.000.440.000
segment-1024360143612057520.3580.000.3600.000
segment-6491418762940479413.6520.000.6540.000
```

Table 4. **Sequence (run segment) list for the human label study.** The sequences are all from the Waymo Open Dataset *val* set.

180th, 300th and 420th epochs. The model is trained with 500 epochs in total, which takes 1-2 days with a V100 GPU.

G. Details of the Human Label Study

We randomly selected 5 sequences from the Waymo Open Dataset *val* set as listed in Table 4 to run the human label study. The 15 labeling tasks (3 sets of re-labels for each run segment) involved 12 labelers with experiences in labeling 3D Lidar point clouds. In total we collected around 2.3K labels (one label for one object track) for the 3 repeated labelings.

How consistent are human labels? Auxiliary to the AP results in the main paper, we also analyze the IoUs between human labels. We found that even for humans, 3D bounding box labeling can be challenging as the input point clouds are often partial and occluded. To understand how consistent human labels are, we compare labels from one labeler with the labels from the other and measure the 3D box consistency by their IoUs. Since we already have the verified public ground truth, we can compare the 3 sets of labels with the public WOD ground truth and get the average box IoU for all objects that are matched (due to occlusions, some objects may be labeled or not labeled by a specific labeler). Specifically, for human boxes that we cannot find a ground truth box with more than 0.03 BEV IoU overlap (false positive or false negative), they were ignored and not counted in the computation.

The statistics are summarized in Table 5. Surprisingly, human labels do not have the consistency one may expect (e.g. 95% IoU). Due to the inherent uncertainty of the problem, even humans can only achieve around 81% 3D IoU or around 88% BEV IoU in their box consistency. As we break down the numbers by distance we see, intuitively, that nearby objects have a significantly higher mean IoU as they have more visible points and more complete view-points. The BEV 2D IoU is also higher than the 3D IoU as we do not require the correct height estimation in the BEV box, which simplifies the problem.

To have a rough understanding of how the boxes generated by our 3D Auto Labeling pipeline compare with human labels, we compute the average IoU of auto labels with the WOD ground truth. Note that those numbers are not directly comparable to the average human IoUs as they cover

different sets of objects (due to the false positives and false negatives). However, it still gives us an understanding that the auto labels are already on par in quality to human labels.

H. More Details about the Semi-supervised Learning Experiment

In the semi-supervised experiments, we use an onboard single-frame *MVF++* detector as the student. We train all networks with an effective batch size of 256 scenes per iteration. The training schedule starts with a warmup period where the learning rate is gradually increased to 0.03 in 1000 iterations. Afterward, we use a cosine decay learning rate schedule to drop the learning rate from 0.03 to 0.

For the intra-domain semi-supervised learning, we randomly select 10% of the sequences (around 15K frames from 79 sequences) to train the 3DAL pipeline which gets an AP of 78.11% on the *validation* set. Then, the 3DAL annotates the rest of the training set (around 142K frames from 719 sequences). Finally, the student is trained on the union of these sets (798 sequences). We train the models for a total of 43K iterations.

For the cross-domain semi-supervised learning, we first train 3DAL on the regular Waymo Open *training* set. Since the domain adaptation validation set is relatively small (*i.e.* only contains 20 sequences), we submit the results to the submission server and report on the domain adaptation *test* set (containing 100 sequences). The 3DAL gets an AP of 78.0% on the domain adaptation *test* set without using any data from that domain. Then, we use the trained pipeline to annotate the domain adaptation *training* and *unlabeled* sets of the Waymo Open Dataset. Finally, the student is trained on the union of the regular *training* set annotated by humans, and domain adaptation *training+unlabeled* sets annotated by 3DAL. Since the data used for training the student is around 2X larger compared to the intra-domain experiment, we also increase the training iterations to 80K.

I. More Analysis Experiments for Object Auto Labeling

In this section, we provide more analysis results auxiliary to the main paper.

Effects of key frame selection for static object auto labeling. Table 6 compares the effects of using different initial boxes (from the detectors) for the model (for the coordinate transform before foreground segmentation): a uniformly chosen random box, the average box and the box with the highest score. We also show the box accuracy of the detectors as a reference (*i.e.* the accuracy of those initial boxes).

Choosing a uniformly random box from the sequence is equivalent to the setting of a frame-centric approach. As for

IoU type	Label type	all	0-30m	30-50m	50m+
valid boxes	human	25,641	11,543	7,963	6,135
	auto	24,146	11,360	7,448	5,338
3D mIoU	human	80.92	85.78	80.29	72.59
	auto	80.29	84.04	77.45	76.28
BEV mIoU	human	87.98	91.26	87.31	82.68
	auto	87.50	90.36	85.09	84.78

Table 5. **The mean IoU of human labels and auto labels compared with the Waymo Open Dataset ground truth for vehicles.** Note that since different labels (human or machine) annotate different number of objects for each frame, those numbers are not directly comparable. They are summarized here for a reference. For a more fair comparison between human and auto labels, see the Average Precision comparison table in the main table. We only evaluate using ground truth boxes with at least one point in it and only evaluate boxes that have a BEV IoU larger than 0.03 with any ground truth box.

Model	detector box	Acc@0.7/0.8
Single-frame MVF++	random	67.17 / 36.61
Multi-frame MVF++	random	73.96 / 43.56
	average	79.29 / 48.67
	highest score	78.67 / 52.42
Auto labeling model	random	79.66 / 52.46
	average	81.22 / 53.96
	highest score	82.28 / 56.92

Table 6. **Effects of initial box selection in static object auto labeling.** Numbers are averaged over 3 runs for the random boxes.

the detector baselines (row 1 and row 2), it directly evaluates the average accuracy of the detector boxes. As for the auto labeling, it means the box estimation is running for every frame, similar to a two-stage refinement step in two-stage detectors. We see that such a frame-centric box estimation achieves the most unfavorable results, as it is not able to leverage the best viewpoint in the sequence (in the object-centric way).

In the average box setting, we average all the boxes from the sequence (in the world coordinate) and use the averaged box for the transformation. For the highest score setting, we select the box with the highest confidence score as the initial box, which is similar to choosing the *best* viewpoint of the object. We see that the strategy to choose the initial box has a great impact and can cause a 4.46 Acc@0.8 difference for the auto labeling model (the highest score box vs. a random box).

Causal model performance. Table 7 compares non-causal models and causal models (for static object auto labeling). The causal model was trained using the causal in-

put (the last row) and only used causal input for inference at every frame. We see the causal model has a relatively lower accuracy compared to non-causal ones, probably due to two reasons. First, it has limited contexts especially for the beginning frames of the track. Second, the pool of initial boxes are much more restricted if the input has to be causal. For non-causal models, we can select the frame with the highest confidence as the key frame and use the box from that frame for the initial coordinate transform. However, for the causal model, it can only select the highest confidence box from the *history* frames, which are not necessarily the well visible ones.

As the causal model’s accuracy is even inferior to the one that just uses a single frame’s points for refinement (the first row in Table 7), the ability to select the best key frame weighs more than the added points from a few history frames. Note that the performance is still better than the detector boxes without refinement (row 2 in Table 6) Such results indicate the benefit of having a non-causal model for the offboard 3D detection.

Table 8 reports a similar study for dynamic object auto labeling. We also see that the causal model’s performance is inferior to the non-causal one, although it still improves upon the raw detector accuracy (Table 8 in the main paper row 2, where the multi-frame MVF++ gets 82.21 / 59.52 accuracy).

Ablations of data augmentation. Table 9 compares the performance of our auto labeling pipeline when trained with different data augmentations. The most accurate models are consistently trained with all the proposed augmentations. For static objects, all the augmentations contribute similarly in terms of accuracy, while for dynamic objects, random rotation around Z-axis appears to be the most critical.

ref frame	context frames	Acc@0.7/0.8
all highest score	$[-0, +0]$ all	78.13 / 50.30 82.28 / 56.92
past highest score	all history	77.56 / 49.21

Table 7. **Effects of temporal contexts for static object auto labeling.** Note that for a causal model, we cannot output a single box for a static object – we have to output the best estimation for every frame using the current frame and the history frames. We compute the average accuracy across all frames for the causal case.

Point cloud context	Box context	Acc@0.7/0.8
$[-2, +2]$	all	85.67 / 65.77
$[-4, 0]$	all history	84.30 / 62.68

Table 8. **Effects of temporal contexts for dynamic object auto labeling.** For causal models, we only use the causal point and box sequence input.

Static		Dynamic	
Aug.	Acc.@0.7/0.8	Aug.	Acc@0.7/0.8
All	82.28/56.92	All	85.67/ 65.77
–D2S	81.72/55.96	–Shift	85.15/65.23
–FlipX	81.42/55.98	–Scale	85.15/65.91
–FlipY	81.50/55.49	–FlipY	85.76 /63.66
–RotateZ	81.72/56.52	–RotateZ	84.94/64.20

Table 9. **Ablations of data augmentation.** We use different data augmentations for static objects and dynamic objects. “All” means all the augmentations are used. “–X” means removing a specific augmentation, “X” from the augmentation set. “D2S” represents the dynamic-to-static augmentation. Best results in each column are in bold.

Tracker		3D AP		BEV AP	
		MOT	GT	MOT	GT
Vehicle	IoU=0.7	84.50	85.77	93.30	96.74
	IoU=0.8	57.82	58.81	84.88	86.18
Pedestrian	IoU=0.5	82.88	83.02	86.32	86.24
	IoU=0.6	63.69	64.80	75.60	75.65

Table 10. **Effects of the tracking accuracy.** “MOT” stands for Multi Object Tracker. “GT” represents Ground Truth Tracker where the ground truth boxes are used. Best results of each comparable pairs are in bold.

Motion State	3D AP		BEV AP	
	IoU=0.7	IoU = 0.8	IoU = 0.7	IoU = 0.8
Pred	84.50	57.82	93.30	84.88
GT	84.98	57.95	93.36	85.13

Table 11. **Effects of the motion state estimation on the offboard 3D detection.** The metric is AP for vehicles on the Waymo Open Dataset *val* set. “Pred” means we are classifying the motion state (static or not) using our linear classifier. “GT” means we are using the ground truth boxes to classify the motion state.

Effects of the tracking accuracy. To study how tracking (association) accuracy affects our offboard 3D detection, we compare results using our Kalman filter tracker and an “oracle” tracker (they share the detection and object auto labeling models, just the tracker is different). For the “oracle” tracker, we associate detector boxes using the ground truth boxes. Specifically, for each detector box, we find its closest ground truth box and assign the ground truth box’s object ID to it. In Table. 10, we observe that better performances can be obtained when a more reliable tracker is used, although the difference is subtle. In particular, the “oracle” tracker introduces more improvement for vehicles than pedestrians. This difference can imply that, for pedestrians, there is more space for improvement in detecting targets than associating detected boxes.

Effects of the motion state estimation accuracy. In Table 11 we study how motion state classification accuracy affects the offboard 3D detection AP. We replace the motion state classifier (“Pred”) with a new one using the ground truth boxes (“GT”) for classification and see how much AP improvements it can bring us. We see that while there are some gains, they are not significant. This is understandable as our linear classifier can already achieve a 99%+ accuracy.

Inference speed. Processing a 20-second sequence (200 frames with the 10Hz sensor input) using a V100 GPU, the detector takes the majority time (around 15 minutes) due to the multi-frame input and test-time augmentation. The tracking takes around 3s and the object-centric refinement takes around 25s, which is 28s in total, 0.14s per frame, or a 3% extra time over the detection. In the offboard setting, we can run detection or the refinement steps in parallel to further reduce the processing latency.

J. More Visualizations

We provide a MP4 video file (named **3dal_video.mp4**) showcasing the 3D auto labeling results of vehicles and pedestrian in point cloud sequences. The sequences are from the Waymo Open Dataset *val* set.

References

- [1] Waymo open dataset: 3d detection challenge. <https://waymo.com/open/challenges/3d-detection/>. Accessed: 2021-01-25.
- [2] Waymo open dataset: Domain adaptation challenge. <https://waymo.com/open/challenges/domain-adaptation/>. Accessed: 2021-01-25.
- [3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.
- [4] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, 2014.
- [5] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, 2019.
- [6] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *CVPR*, 2018.
- [7] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CVPR*, 2017.
- [8] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.
- [9] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointcnn: 3d object proposal generation and detection from point cloud. *arXiv preprint arXiv:1812.04244*, 2018.
- [10] Roman Solovyev, Weimin Wang, and Tatiana Gabruseva. Weighted boxes fusion: ensembling boxes for object detection models. *arXiv preprint arXiv:1910.13302*, 2019.
- [11] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. In *Proc. Int. Conf. Computer Vision (ICCV)*, 2019.
- [12] Weiyao Wang, Du Tran, and Matt Feiszli. What makes training multi-modal networks hard? *arXiv preprint arXiv:1905.12681*, 2019.
- [13] Xinshuo Weng and Kris Kitani. A baseline for 3d multi-object tracking. *arXiv preprint arXiv:1907.03961*, 2019.
- [14] Yin Zhou, Pei Sun, Yu Zhang, Dragomir Anguelov, Jiyang Gao, Tom Ouyang, James Guo, Jiquan Ngiam, and Vijay Vasudevan. End-to-end multi-view fusion for 3d object detection in lidar point clouds. In *Conference on Robot Learning*, pages 923–932, 2020.
- [15] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, 2018.