PnPNet: End-to-End Perception and Prediction with Tracking in the Loop

Ming Liang[1*]   Bin Yang[1,2*]

Wenyuan Zeng[1,2]   Yun Chen[1]   Rui Hu[1]   Sergio Casas[1,2]   Raquel Urtasun[1,2]

[1]Uber Advanced Technologies Group   [2]University of Toronto

{ming.liang, byang10, wenyuan, yun.chen, rui.hu, sergio.casas, urtasun}@uber.com

Abstract

We tackle the problem of joint perception and motion forecasting in the context of self-driving vehicles. Towards this goal we propose PnPNet, an end-to-end model that takes as input sequential sensor data, and outputs at each time step object tracks and their future trajectories. The key component is a novel tracking module that generates object tracks online from detections and exploits trajectory level features for motion forecasting. Specifically, the object tracks get updated at each time step by solving both the data association problem and the trajectory estimation problem. Importantly, the whole model is end-to-end trainable and benefits from joint optimization of all tasks. We validate PnPNet on two large-scale driving datasets, and show significant improvements over the state-of-the-art with better occlusion recovery and more accurate future prediction.

1. Introduction

We focus on the task of joint perception and prediction (motion forecasting) in the context of self-driving vehicles. This is a crucial task as in order to plan a safe maneuver, anticipating the future decisions of surrounding agents is as important as estimating their current state.

Different paradigms have been proposed to solve the perception and prediction problem, which are compared in Figure 1. Traditional self-driving autonomy stacks [2, 9, 16] decompose the problem into three subtasks: object detection, object tracking, and motion forecasting, and rely on independent components that perform these subtasks sequentially. However, as each component is developed separately, this paradigm makes compromises in each module in order to meet the computing budget. Furthermore, the interface between these modules is very compact (typically the object's position, velocity, acceleration, and their uncertainty estimates), which prevents downstream tasks from correcting the mistakes made by upstream ones.
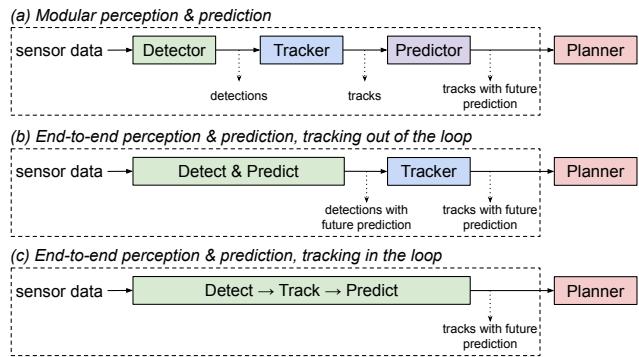
---

*[*]Equal contribution.

Figure 1. **Three paradigms for perception and prediction.** Traditional approach (a) adopts the modular design that decomposes the stack into subtasks and solves them with individual models. End-to-end method like [29] (b) uses a joint model to solve detection and prediction simultaneously, but performs tracking as post-processing. As a result, the full temporal history contained in tracks is not used by detection and prediction. Our approach (c) brings tracking into the loop so that all tasks benefit from rich temporal context.

Recently, models that solve the detection and prediction tasks jointly with a single neural network have been proposed [29], resulting in more efficient computation and improved accuracy. This paradigm is later extended to further solve the driver intention [8] and motion planning [52] by adding the corresponding modules on top of the shared backbone network. These approaches, however, suffer from limited use of temporal history because object tracking is not included in the loop and thus only leverage up to 1 second of past sensor data due to limited model capacity. This may cause problems when dealing with occluded actors and may produce temporal inconsistency in predictions.

In this paper we argue that leveraging the past is key for sequential decision making process like motion forecasting. Towards this goal we propose *PnPNet*, a new paradigm that combines ideas from multi-object tracking and joint perception and prediction models. While the detection module processes sequential sensor data and generates object de-

tections at each time step, the tracking module associates these estimates across time for better understanding of object states (*e.g.*, occlusion reasoning, trajectory smoothing), which in turn provides richer information for the prediction module to produce accurate future trajectories. Importantly, all modules share computation as there is a single backbone network, and the full model can be trained end-to-end.

We make two main technical contributions in PnPNet. First, we propose a novel object trajectory representation defined on a sequence of object detections to fully capture the temporal characteristics of the actors. In particular, for each object we first extract its inferred motion (from past detection estimates) and raw observations (from sensor features) at each time step, and then model its dynamics using a recurrent network. Importantly, this trajectory representation is utilized in both tracking and prediction modules. Second, we propose a multi-object tracker that solves both the discrete problem of data association and the continuous problem of trajectory estimation [34] via learnable functions that can handle object occlusion, new birth of trajectories and false positive detections.

We validate PnPNet on two large-scale driving datasets, and demonstrate its effectiveness with both modular metrics (standard benchmark for each subtask) and system metrics (end-to-end performance under the real-world setting). Experiments show that PnPNet achieves significant improvements over previous state-of-the-art paradigms in both perception and prediction tasks. Specifically, PnPNet recovers objects from occlusion, produces more complete object trajectories, and outputs more accurate future predictions.

2. Related Work

In this section we review works that tackle the tasks of 3D object detection, tracking, and motion forecasting separately, followed by approaches that tackle these jointly.

3D Object Detection: While several approaches [11, 10, 43] try to perform 3D object detection from images, the inherent depth ambiguity hinders them from being applied in safety-critical applications. Methods that exploit depth sensors (*e.g.* LiDAR) achieve superior performance with various representations of point clouds [50, 54, 48, 39, 51, 33]. Recently sensor fusion methods [12, 46, 27, 35, 49, 26, 32] further push the performance by exploiting complementary information from cameras and/or maps. For efficiency and accuracy, PnPNet utilizes the bird's eye view representation of LiDAR and maps and performs single shot detection.

Multi-Object Tracking: Most approaches mainly follow the tracking-by-detection paradigm [5], which comprise the discrete problem of data association and continuous problem of trajectory estimation [34]. Many frameworks have

been proposed to solve the data association problem: *e.g.*, Markov Decision Processes [45], min-cost flow [24, 17, 37], linear assignment problem [38, 44] and graph cut [31, 42]. To handle object occlusion when there's no detection available, hand-crafted heuristics [19] or single-object tracking approach [47, 14] has been explored. Apart from the association paradigm, different representations are used to computes the affinity. While [44] exploits the 3D motion clues only, approaches that extract sensor features [53, 14] typically limit the temporal history to 3 time steps. In contrast, PnPNet solves both discrete and continuous problems, with a long-term trajectory representation that captures both sensor observation and motion clue of actors.

Motion Forecasting: Various approaches have been proposed to model the multi-agent interactions and multi-modal behaviors in motion forecasting. DESIRE [23] uses a variational auto-encoder to generate trajectory proposals and refines them based on semantic scene context and interactions between agents. To better model the interactions, game theory is used to formulate the problem [30]. Social-LSTM [1] introduces social pooling to model nearby agents' trajectory patterns, while Social-GAN [18] further improves the performance by adding adversarial training. In parallel to different predictive models, various input representations are also explored. Besides the past states of actors, sensor features are also explored to provide more context [23, 25, 36]. However, these methods are typically developed on ground-truth object labels, and have generalization issues when applied to noisy detections [36]. In self-driving domain, raster representation in bird's eye view that encodes both the perception output and map information is widely used [2, 9, 16, 15]. In contrast, the prediction module in PnPNet directly reuses the perception features for rich scene context, and also extracts object states explicitly from past object tracks.

Joint Models for Perception and Prediction: FAF [29] proposes to jointly reason about 3D object detection and motion forecasting by exploiting temporal features from multi-sweep LiDAR point clouds. An efficient bird's eye view representation and network architecture are utilized for real-time inference. IntentNet [8] extends the approach by adding the prediction of high-level intentions of each agent from semantic HD maps. SpAGNN [7] leverages graph neural networks with spatial reasoning to model multi-agent interactions. NeuralMP [52] takes one step further by sharing the feature for motion planning with perception and prediction, leading to an end-to-end motion planner. While all these approaches share the sensor features for detection and prediction, they fail to exploit the rich information of actors along the time dimension. PnPNet addresses this by incorporating online tracking and extract-
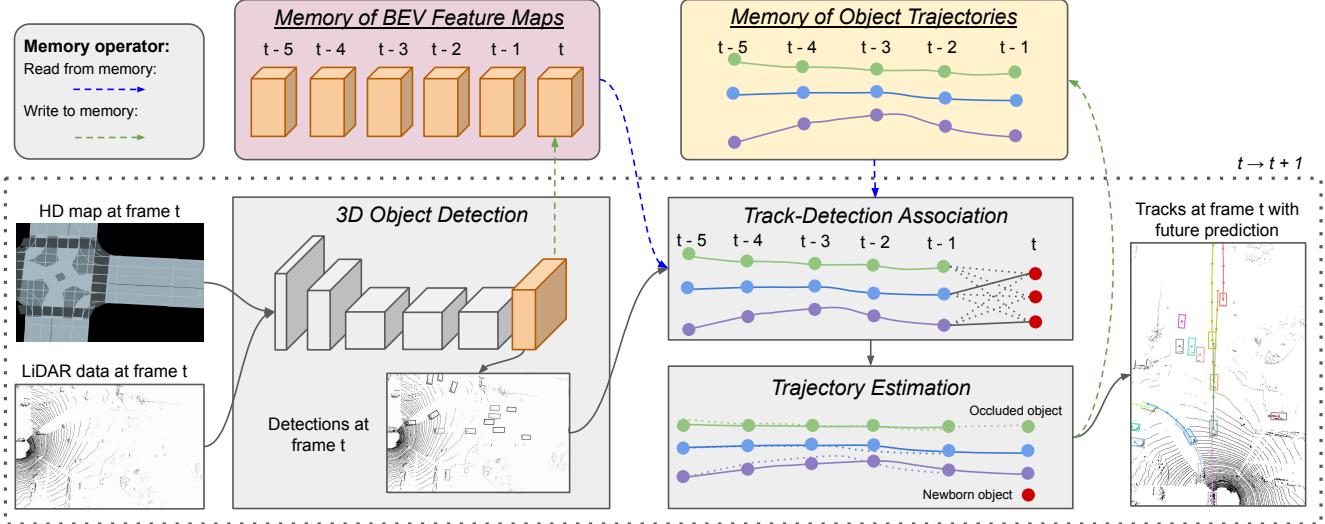
Figure 2. **The proposed PnPNet for end-to-end perception and prediction.** The model consists of three modules that perform 3D object detection, discrete-continuous tracking, and motion forecasting sequentially. To extract trajectory level actor representations used for tracking and prediction, we also equip the model with two explicit memories: one for global sensor feature maps, and one for past object trajectories. Both memories get updated at each time step with up-to-date sensor features and tracking results.

ing trajectory-level actor representation to encode long-term history, which in turn improves all tasks.

3. End-to-End Perception and Prediction

We introduce *PnPNet* (Figure 2), an end-to-end model designed for efficient and accurate joint perception and prediction in the context of autonomous driving. Instead of designing individual models for each subtask like the traditional engineering stack, we follow the recent advances of joint modeling with shared feature computation [29, 8]. However, the main weakness of this paradigm is the limited exploitation of history information. Since these approaches do not have explicit tracking in the loop, to perform motion forecasting the object's motion history has to be estimated from the raw sensor data, which can be particularly difficult for occluded objects. As a result, the performance of the model usually saturates with fewer than 1 second of sensor data [29, 52]. Furthermore, these approaches cannot track through occlusions longer than the input time horizon, as there's no evidence. All these drawbacks hinder the performance of these approaches in the task of motion forecasting.

In contrast, PnPNet addresses the issue with two key components: a novel trajectory level representation that captures the rich temporal characteristics of actors, and a new online discrete-continuous tracking module that generates such trajectories from detections across time. In the remainder we first present the three modules that perform detection, tracking and prediction sequentially, and then show how the full model can be trained end-to-end.

3.1. Object Detection Module

We adopt a 3D object detector that takes multi-sweep LiDAR point clouds (up to 0.5 second) and an HD map as input, and outputs object detections in bird's eye view (BEV). We use a voxel based representation of LiDAR data in BEV, and combine multiple sweeps by concatenating along the height dimension (similar to [8], with the ego motion compensated for the previous sweeps). We follow [49] to encode the geometric and semantic information of the HD map (if available) into the voxel representation. We apply a 2D convolutional neural network (CNN) based backbone with multi-scale feature fusion to create our intermediate feature representation that will be later used for tracking and motion forecasting

$$\mathcal{F}_{\text{bev}}^t(\mathbf{x}^t) = \text{CNN}_{\text{bev}}(\mathbf{x}^t) \tag{1}$$

where $\mathbf{x}^t$ is our input composed of multiple LiDAR sweeps (up to frame t) and the HD map. Following the single stage detector [50] we then use a convolutional detection header to output dense detections, each parameterized as $(u_i^t, v_i^t, w_i, l_i, \theta_i^t)$ representing its position, size and orientation in the ego-centric BEV space at frame t. Thus

$$\mathcal{D}^t = \text{CNN}_{\text{det}}(\mathcal{F}_{\text{bev}}^t) \tag{2}$$

where the number of detections $N_t = |\mathcal{D}^t|$ varies per frame. While the detection module generates object detections at each frame independently, the tracking module links them through time, which we review next.
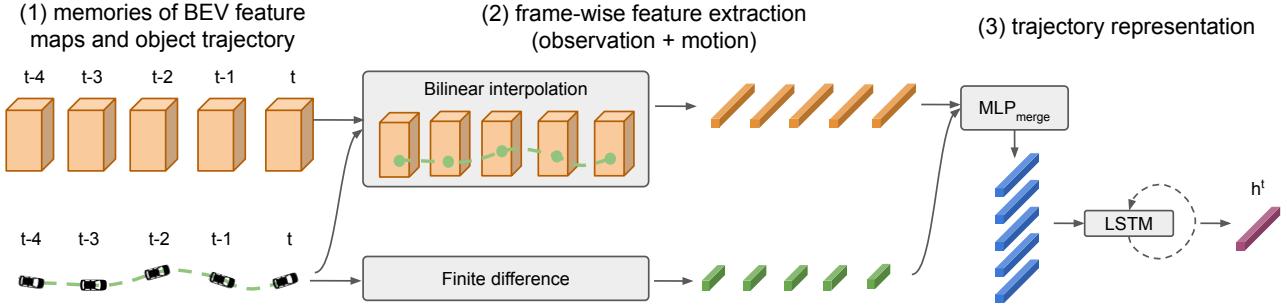
Figure 3. **The proposed trajectory level object representation.** Given an object trajectory, we first extract its sensor observation and motion features at each time step, and then apply an LSTM network to model the temporal dynamics.

3.2. Discrete-Continuous Tracking Module

There exist two distinct challenges in multi-object tracking: the discrete problem of data association and the continuous problem of trajectory estimation [34]. While previous methods mostly focus on the discrete problem, we argue that the continuous problem is as important in our application. From the tracking perspective, it helps to prevent association errors (*i.e.*, identity switches) from accumulating through time. From the prediction perspective, it reduces the variance in motion history caused by the localization error of detections. Towards this goal, we propose a two-stage tracking framework, where the first stage solves the association problem between previous tracks and current detections, and the second stage refines the associated new tracks to generate smoother trajectories.

Trajectory level object representation: We now show how to learn rich and concise representations for the tracking and prediction tasks. We formulate the representation learning as a sequence modeling problem (Figure 3) and exploit a Long Short-Term Memory (LSTM) network to capture the relevant information. Key to the success of the LSTM is to have informed input features. For the task at hand, these features should contain both the object's observation as well as information about its motion. Given an object track $\mathcal{P}_i^t = \mathcal{D}_i^{t_0 \dots t}$ from frame t_0 to frame t, let $f_i^{\text{bev},t}$ and $f_i^{\text{velocity},t}$ be features representing the observation and motion of each object

$$f_i^{\text{bev, t}} = \text{BilinearInterp}(\mathcal{F}_{\text{bev}}^t, (u_i^t, v_i^t)) \qquad (3)$$
$$f_i^{\text{velocity},t} = (\dot{x}_i^t, \dot{x}_{\text{ego}}^t, \dot{\theta}_{\text{ego}}^t) \qquad (4)$$

where $\mathcal{F}_{\text{bev}}^t$ is the BEV feature map from the backbone network, $\dot{x}_i$ and $\dot{x}_{\text{ego}}$ are the 2-dimensional velocities of the i-th object and the ego-car respectively, and $\dot{\theta}_{\text{ego}}$ is the angular velocity of the ego-car. Note that we estimate the velocities by finite differences over positions, and we use the velocities of each object and the ego car so that we can estimate the absolute velocities. For newborn objects we initialize

its velocity to 0. For angular velocity of ego car we parameterize it as its cosine and sine values. We then combine the two features into a single feature representation

$$f(\mathcal{D}_i^t) = \text{MLP}_{\text{merge}}(f_i^{\text{bev},t}, f_i^{\text{velocity},t}) \qquad (5)$$

The combined object feature is computed for each frame from t_0 to t, and they are fed to an LSTM network to produce the trajectory level representation

$$h(\mathcal{P}_i^t) = \text{LSTM}(f(\mathcal{D}_i^{t_0 \dots t})) \qquad (6)$$

We use an LSTM as our sequence model due to its capability to handle varying input length and capture long term dependencies. Note that PnPNet exploits the learned trajectory level representation to perform both tracking and prediction tasks.

Data association: Given N_t detections in the current frame and M_{t-1} object tracks in the previous frame, the discrete tracker needs to determine the association between the previous tracks and the current detections. In practice we find that the association problem eases when given 3D motion clues. However, properly handling newborn objects and occluded objects can be challenging. Unfortunately both cases happen frequently in driving scenarios. To handle these two challenges we propose a hybrid approach that exploits the best of multi-object tracking and single-object tracking approaches.

We first determine the identities of the N_t detections by associating them with all M_{t-1} existing tracks. The association problem is formulated as a bipartite matching problem so that exclusive track-to-detection correspondence is guaranteed. Newborn objects are handled by adding N_t *virtual candidates* to the M_{t-1} tracks. Note that the result of the association will be fully determined given the affinity matrix that captures the similarity between each detection and track. Here, we exploit the learned object representation to

compute the affinity matrix $C \in \mathbb{R}^{N_t \times (M_{t-1}+N_t)}$ as follows

$$C_{i,j} = \begin{cases} \text{MLP}_{\text{pair}}(f(\mathcal{D}_i^t), h(\mathcal{P}_j^{t-1})) & \text{if } 1 \leq j \leq M_{t-1}, \\ \text{MLP}_{\text{unary}}(f(\mathcal{D}_i^t)) & \text{if } j = M_{t-1} + i, \\ -\inf & \text{otherwise} \end{cases}$$
(7)

where f and h are the aforementioned single-frame object feature (Eq. 5) and trajectory level object feature (Eq. 6) respectively. MLP_{pair} computes the affinity score of any detection-track pair, and $\text{MLP}_{\text{unary}}$ estimates the score of any detection being a new instance. We optimally solve the bipartite matching problem defined by C with the Hungarian algorithm [21].

Note that dealing with occluded objects via Hungarian matching is very difficult since it is unclear what object estimations should be added to the detection set of the bipartite graph as they are missed by the detector. To handle such cases we take advantage of single-object tracking (SOT) that is performed on the unmatched tracks (which means the object exists in the past but fails to find a matched observation at current frame). Our SOT design inherits the philosophy of the Siamese tracker [4], but replaces the correlation filter with a learnable MLP. Specifically, for each unmatched track $\mathcal{P}_j^{t-1}$, we define its detection candidates $\tilde{\mathcal{D}}_i^t$ as voxels within a local neighborhood Ω_j centered at $(\tilde{u}_j^t, \tilde{v}_j^t)$ (estimated by transforming (u_j^{t-1}, v_j^{t-1}) to current frame t with ego motion compensation). We find the best detection candidate $\tilde{\mathcal{D}}_k^t$ by solving for the best match

$$k = \underset{i \in \Omega_j}{\arg\max} \, \text{MLP}_{\text{pair}}(f(\tilde{\mathcal{D}}_i^t), h(\mathcal{P}_j^{t-1}))$$
(8)

In practice, we set the neighborhood size according to the prior knowledge of the object's maximum velocity (110 km/h for vehicles in our case). Compared with method [44] that predicts the position of occluded object with a motion model, our SOT approach exploits additional observation (such as map context) to get a more precise estimation.

Combining the results from bipartite matching and SOT gives us the final set of tracks $\mathcal{P}^t$, which has $N_t + K_t$ instances, where K_t is the number of unmatched tracks that are processed by our single-object tracker. Note that all affinity scores in data association are predicted from learnable representations and matching functions, which can learn from data to capture the complex correlations in temporal motion and appearance clues for long-term tracking.

Trajectory estimation: The goal of this module is to re-estimate each object track (in terms of the confidence score and trajectory waypoints) given the new observation at current frame, which helps to eliminate false positives from the detector and reduce the localization error coming from either detection or association. Specifically, for each object

Methods	AP↑	AP@0.5m	@1m	@2m	@4m
Mapillary [40]	47.9	10.2	36.2	64.9	80.1
PointPillars [22]	70.5	55.5	71.8	76.1	78.6
Megvii [55]	82.3	72.9	82.5	85.9	**87.7**
PnPNet, det only	**82.7**	**73.7**	**83.3**	**86.2**	87.5

Table 1. **Evaluation of 3D object detection (*car*) on nuScenes.**

track we update its LSTM representation according to the current association, and estimate its confidence score and center position offsets for the most recent T_0 frames

$$\text{score}_i, \Delta u_i^{t-T_0+1:t}, \Delta v_i^{t-T_0+1:t} = \text{MLP}_{\text{refine}}(h(\mathcal{P}_i^t)) \quad (9)$$

T_0 is typically shorter than the full trajectory horizon, as near-term history is more relevant to the current frame. After applying the refinement to all tracks, we perform Non-Maximum Suppression (NMS) on current frame estimations ranked by the new scores and keep top M_t tracks to remove false positives and duplicates.

3.3. Motion Forecasting Module

While previous joint perception and prediction models [29, 8] make the prediction module another convolutional header on top of the detection backbone network, which shares the same features with the detection header, in PnPNet we put the prediction module after explicit object tracking, with the object trajectory representation as input

$$\Delta u_i^{t:t+\Delta T}, \Delta v_i^{t:t+\Delta T} = \text{MLP}_{\text{predict}}(h(\mathcal{P}_i^t)) \quad (10)$$

where ΔT is the length of the prediction horizon.

3.4. End-to-End Learning

We train our PnPNet end-to-end with a multi-task loss of detection, tracking and prediction:

$$\mathcal{L} = \mathcal{L}_{\text{detect}} + \mathcal{L}_{\text{track}} + \mathcal{L}_{\text{predict}} \quad (11)$$

For detection, we use a cross-entropy loss with hard negative mining for classification and sum of smooth ℓ_1 losses over the bounding box regression terms: size, position and orientation. For discrete-continuous tracking, we propose to use the max-margin loss on the affinity matrix (Eq. 7), SOT matching scores (Eq. 8) and trajectory scores (Eq. 9) respectively:

$$\mathcal{L}_{\text{track}} = \mathcal{L}_{\text{score}}^{\text{affinity}} + \mathcal{L}_{\text{score}}^{\text{sot}} + \mathcal{L}_{\text{score}}^{\text{refine}} + \mathcal{L}_{\text{reg}}^{\text{refine}} \quad (12)$$

$$\mathcal{L}_{\text{score}} = \frac{1}{N_{i,j}} \sum_{i \in \text{pos}, j \in \text{neg}} \max(0, m - (a_i - a_j)) \quad (13)$$

where a_i is the score of i-th positive sample, a_j is the score of j-th negative sample, m is the margin threshold, and $N_{i,j}$ denotes the number of positive-negative pairs. For $\mathcal{L}_{\text{score}}^{\text{affinity}}$ and $\mathcal{L}_{\text{score}}^{\text{sot}}$, we use the pairs of the positive match and each

Methods	AMOTA↑	AMOTP↓	RECALL↑	MOTA↑	MOTP↓	MT↑	ML↓	FP↓	IDS↓	FRAG↓	TID↓	LGD↓
StanfordIPRL-TRI [13]	73.5%	0.53	73.8%	62.3%	0.26	1978	1053	**6340**	367	341	0.79	1.08
PnPNet, KF tracker	76.1%	0.52	79.1%	64.8%	**0.24**	2351	**745**	7555	802	628	0.51	0.97
PnPNet	**81.5%**	**0.44**	**81.6%**	**69.7%**	0.26	**2518**	804	6771	**152**	**310**	**0.30**	**0.57**

Table 2. **Evaluation of multi-object tracking (*car*) on nuScenes.** Besides standard MOT metrics [3], four new metrics are added: **AMOTA/AMOTP**: MOTA/MOTP averaged over different recall thresholds; **TID**: average track initialization duration in seconds; **LGD**: average longest gap duration in seconds.

	Perception Metrics ↑				Prediction Metrics ↓			
	AP (%)		Max. Recall (%)		ADE (m)		FDE (m)	
	0.1 IoU	0.5 IoU	0.1 IoU	0.5 IoU	60% TP	90% TP	60% TP	90% TP
nuScenes cars								
PnPNet, w/o track	84.9	79.8	90.9	84.6	0.69	0.75	1.09	1.14
PnPNet	87.1 (+2.2)	82.1 (+2.3)	95.3 (+4.4)	88.4 (+3.8)	0.58 (-15%)	0.68 (-9%)	0.93 (-14%)	1.04 (-8%)
ATG4D vehicles								
PnPNet, w/o track	93.9	90.0	97.5	93.4	0.69	0.77	1.12	1.21
PnPNet	95.8 (+2.0)	92.2 (+2.2)	99.1 (+1.6)	95.4 (+2.1)	0.55 (-20%)	0.65 (-16%)	0.92 (-18%)	1.03 (-15%)
ATG4D pedestrians								
PnPNet, w/o track	77.7	69.0	88.3	78.5	0.39	0.41	0.57	0.60
PnPNet	79.5 (+1.8)	70.9 (+1.9)	91.0 (+2.7)	81.0 (+2.5)	0.34 (-13%)	0.36 (-11%)	0.51 (-11%)	0.54 (-10%)

Table 3. **Evaluation of end-to-end perception and prediction on nuScenes and ATG4D.** The baseline model (PnPNet, w/o track) follows the paradigm of [29], which performs joint detection and prediction without tracking in the loop.

negative matches. For $\mathcal{L}_{\text{score}}^{\text{refine}}$, we use all object pairs (ordered) in which the first object has a larger IoU with the corresponding ground-truth. In this way, the refine score is trained to be ordered by their IoU with the ground-truth. NMS based on this refine score enables higher quality tracks to be kept when there are duplicates. We set the margin to 0.2 for all scores. We use smooth ℓ_1 loss for both trajectory refinement (Eq. 9) and motion forecasting (Eq. 10).

Optimizing PnPNet is nontrivial due to the complex dependencies of the intermediate results across tasks and time. Normal training technique for sequence models like "teacher forcing" brings exposure bias to the model and leads to severe over-fitting. To address this, we fully emulate the testing phase by sampling a mini-batch of video clips during training. At each frame, the tracking and prediction modules take as input the *online estimations* from either previous modules or previous frames, and the *ground-truth labels* are only used in computing the multi-task loss.

We use Adam optimizer [20] to train PnPNet, with a frame rate of 10 Hz. At each frame we maintain at most $M = 50$ tracks and $N = 50$ detections per class. The NMS threshold on detections and tracks is 0.1 IoU. We refine the most recent $T_0 = 4$ frames and predict future $\Delta T = 3$ seconds with 0.5 second interval. For real-time efficiency, we limit the track length to $T = 16$ frames.

4. Experiments

We demonstrate the effectiveness of PnPNet on two large-scale real-world driving datasets. We focus on modular metrics on detection and tracking, as well as system metrics on end-to-end perception and prediction. While modular metrics compare our method with other state-of-the-arts under the constrained setting, system metrics reveal the model performance under the real-world setting. We show that with the proposed trajectory representation and discrete-continuous tracking, results on each subtask as well as the whole system improve significantly. We also provide ablation study of each component and qualitative results of the model.

4.1. Datasets and Metrics

nuScenes [6]: This dataset contains 1000 20-second log snippets, with 32-beam LiDAR sweeps at 20 Hz and corresponding 3D object labels (linearly interpolated from 2 Hz annotations). We train a LiDAR only PnPNet model on nuScenes due to some alignment issues in the maps [1]. We evaluate on the car class following the official train/val split.

ATG4D: Although nuScenes dataset contains 1000 snippets, they come from 84 unique driving journeys only. The object labels are also constrained to be within 50 meters range, with 63.5% of the cars being parked. In order to better evaluate the real-world performance, especially in urban areas, we evaluate also on the more challenging driving dataset ATG4D [50]. Specifically, ATG4D contains ∼5000 log snippets from ∼1000 unique journeys in North America. Each snippet has 64-beam LiDAR sweeps at 10 Hz with corresponding HD maps (drivable area, lane graph, and ground height) and 3D object labels within 100 meters range (48.1% of the cars are parked). We split 500 snippets out for evaluation without journey overlap with the training

---

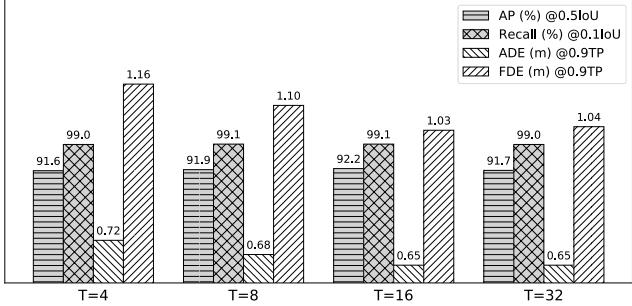[1] As of map version 1.0. Most issues are resolved in map version 1.2.

Figure 4. **Ablation study on object track length** T. Longer track achieves similar perception results but better prediction results. We use T=16 in PnPNet.

data. We train a LiDAR+map PnPNet model and evaluate on the vehicle and pedestrian classes.

Modular metrics: We simply follow the detection and tracking metrics defined by nuScenes [6] for a fair comparison with other state-of-the-arts. Specifically, we use Average Precision (AP) for detection and MOT metrics [3] for tracking. Metrics are computed under the constrained setting, where we only evaluate on *visible* objects (with at least 1 LiDAR point observation).

System metrics: We define system metrics to evaluate the performance of end-to-end perception and prediction, where prediction is conducted on detections instead of ground-truth labels. Specifically, for perception we use AP and maximum object recall, and for prediction we use Average Displacement Error (ADE) over 3 seconds (with 1 second interval) and Final Displacement Error (FDE) at 3 seconds. Prediction metrics are computed on True Positive (TP) detections at 0.5 IoU. To mimic real-world setting, we evaluate on *all* objects (including totally occluded ones, which are critical for safety on a self-driving vehicle).

4.2. Main Results

3D object detection: We evaluate the detection module of PnPNet on nuScenes, in comparison with other state-of-the-art 3D detectors. Table 1 shows that our detector outperforms the leading approach **Megvii** [55] in most metrics, with larger gains at higher localization precision (0.8% AP improvement at 0.5 meter threshold).

Multi-object tracking: We evaluate the detection and tracking modules of PnPNet on nuScenes, in comparison with the leading approach **StanfordIPRL-TRI** [13] (with **Megvii** detections [55]) on the leaderboard. We also add another tracking baseline that replaces our tracking module with a self-implemented Kalman Filtering based tracker (denoted as "PnPNet, KF tracker"). Table 2 shows that

while our KF tracker baseline surpasses [13] by 2.6% in the ranking metric AMOTA, the proposed PnPNet outperforms [13] by 8.0%. In terms of fine-grained metrics, PnPNet has more complete trajectories (fewer identity switches and fragmentations), quicker occlusion recovery (smaller track initialization duration and gap duration), and more precise trajectories (smaller AMOTP).

End-to-end perception and prediction: Now we evaluate PnPNet in terms of end-to-end perception and prediction on both nuScenes and ATG4D datasets with system metrics under the real-world setting (including totally occluded objects), with an evaluation frame rate of 10 Hz. We compare with the baseline model that also performs end-to-end perception and prediction, but without tracking in the loop (*i.e.*, we remove the tracking module, and add the prediction header on top of the detection backbone network). We denote this baseline as "PnPNet, w/o track", which can be considered as a re-implementation of [29]. By comparing with this baseline we can measure the effectiveness of the two main contributions of PnPNet, namely the trajectory representation and the discrete-continuous tracking.

Table 3 shows that PnPNet consistently outperforms the baseline in all system metrics on two object classes and two datasets, achieving up to 2% AP gain (note that AP of PnPNet here is evaluated on tracks), up to 4% recall gain, and up to 20% prediction improvement. The consistent improvements in different object classes and sensor configurations showcase the generality of the proposed method. More specifically, perception-wise, PnPNet is able to recover from long-term occlusion thanks to the proposed tracking module, which is revealed by up to 4% boost in recall at 0.1 IoU. Besides occlusion recovery, PnPNet also benefits from trajectory estimation (for both confidence scores and waypoints), suggested by around 2% absolute gain in AP at 0.5 IoU. Prediction-wise, PnPNet achieves 8% to 20% relative improvements, which mainly come from two aspects: better perception results from tracking, and stronger object representation at trajectory level. In particular, the gain becomes larger at lower recall (60% TP) when the perception results are more confident and precise, and still remains significant at 90% TP where the perception results are noisy. We also observe larger gains on ATG4D dataset compared with nuScenes due to larger proportion of moving objects.

4.3. Ablation Study

We conduct ablation studies on two key components of PnPNet: the object trajectory representation and the discrete-continuous tracker. Note that all ablations are evaluated on ATG4D vehicles with system metrics.
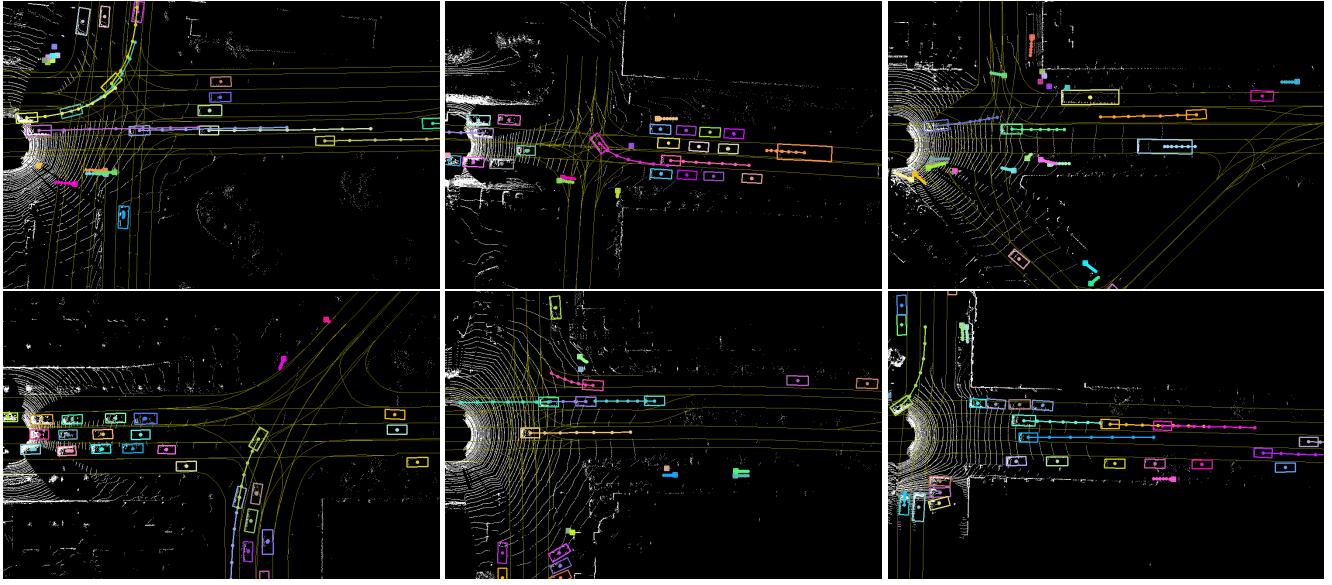
Figure 5. **Qualitative results of PnPNet on ATG4D.** We visualize the perception and prediction results of vehicles and pedestrians up to 100 meters far away, where the ego car is located at the middle left of each frame heading to the right.

Module removed	AP (%) ↑ @0.5IoU	MaxRec. (%) ↑ @0.1IoU	ADE (m) ↓ 90%TP	FDE (m) ↓ 90%TP
motion feature	-0.2	-0.1	+6.2%	+5.5%
single object track	-1.7	-1.7	+2.0%	+2.0%
trajectory rescore	-7.9	-0.4	+4.7%	+4.8%
trajectory refine	-2.1	-1.6	+4.8%	+4.7%
whole track module	-2.2	-1.6	+18%	+17%

Table 4. **Ablation study on discrete-continuous tracking.** We remove one module from the full PnPNet with other modules unchanged and report the relative performance change.

Object track length: We compare PnPNet using different lengths of object track in Figure 4. From the results we see the history length does not affect perception performance much, suggesting that perception relies more on short-term observations. However, longer track does achieve lower prediction errors, which indicates that long-term history is helpful to future prediction. Prediction performance plateaus at around 16 frames (1.6 seconds), as real-world traffic changes often.

Importance of explicit motion: One strong finding of PnPNet that was not exploited in previous joint models [29, 8] is the fact that exploiting motion from explicit object trajectories is more accurate than inferring motion from the features computed from the raw sensor data. We verify this by removing the motion feature from the trajectory representation of PnPNet, with other components unchanged. As shown in Table 4, the detection performance remains almost the same, but the prediction error increases significantly (∼6%). This suggests that the explicit motion history

obtained from tracking is helpful for prediction.

Single-object tracking for occlusion recovery: PnPNet recovers from object occlusion by tracking existing tracks through time. We implement this with a single-object tracker so that current frame's information (*e.g.*, map context) is leveraged as well. If this capability is removed from PnPNet, we observe a performance drop in both perception and prediction (see Table 4). In particular, in the absence of the single-object tracker the recall drops by 1.7% due to object occlusion, and prediction errors increase by 2% due to incomplete motion history.

Effect of trajectory estimation: In addition to solving the data association problem in multi-object tracking, PnPNet also re-estimates the trajectory by re-scoring it and refining its waypoints. While re-scoring does not affect the maximum object recall, it determines the order of object trajectories from multiple sources (newborn objects, matched tracks, and tracks through occlusion) and therefore affects the order-dependent metrics. From the results shown in Table 4 we can see that, without re-scoring the detection AP drops significantly. Similar performance drop happens in the prediction metric as well. For trajectory refinement, since it reduces the localization error of online generated perception results, it helps establish a smoother and more accurate motion history. From the results we see that without the trajectory refinement all metrics degrade.

4.4. Qualitative Results

In Figure 5 we showcase some qualitative results of the proposed model, which illustrate that by learning trajectory representations and explicitly solving multi-object tracking, PnPNet is able to recover from long-term object occlusion, and generate more accurate future trajectories.

5. Conclusion

In this paper we proposed PnPNet, an end-to-end model for perception and prediction in autonomous driving. Instead of designing individual models for each subtask like the traditional engineering stack, we follow the recent advances of joint modeling with shared feature computation, and further improve upon the paradigm with a novel multi-object tracker that generates object trajectories online from detections and exploits trajectory level features for motion forecasting. We validate PnPNet on two large-scale driving datasets, and show significant improvements in both perception and prediction metrics. In the future we plan to apply our approach to more complex downstream tasks like multi-agent behavior prediction and motion planning.

References

[1] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *CVPR*, 2016. 2

[2] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *arXiv preprint arXiv:1812.03079*, 2018. 1, 2

[3] Keni Bernardin, Alexander Elbs, and Rainer Stiefelhagen. Multiple object tracking performance metrics and evaluation in a smart room environment. In *Sixth IEEE International Workshop on Visual Surveillance, in conjunction with ECCV*, 2006. 6, 7

[4] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. Fully-convolutional siamese networks for object tracking. In *ECCV*, 2016. 5

[5] Michael D Breitenstein, Fabian Reichlin, Bastian Leibe, Esther Koller-Meier, and Luc Van Gool. Online multiperson tracking-by-detection from a single, uncalibrated camera. *TPAMI*, 33(9):1820–1833, 2010. 2

[6] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *CVPR*, 2020. 6, 7

[7] Sergio Casas, Cole Gulino, Renjie Liao, and Raquel Urtasun. Spatially-aware graph neural networks for relational behavior forecasting from sensor data. In *ICRA*, 2020. 2

[8] Sergio Casas, Wenjie Luo, and Raquel Urtasun. Intentnet: Learning to predict intention from raw sensor data. In *CoRL*, 2018. 1, 2, 3, 5, 8

[9] Yuning Chai, Benjamin Sapp, Mayank Bansal, and Dragomir Anguelov. Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction. *CoRL*, 2019. 1, 2

[10] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *CVPR*, 2016. 2

[11] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. In *NeurIPS*, 2015. 2

[12] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *CVPR*, 2017. 2

[13] Hsu-kuang Chiu, Antonio Prioletti, Jie Li, and Jeannette Bohg. Probabilistic 3d multi-object tracking for autonomous driving. *arXiv preprint arXiv:2001.05673*, 2020. 6, 7

[14] Peng Chu and Haibin Ling. Famnet: Joint learning of feature, affinity and multi-dimensional assignment for online multiple object tracking. In *ICCV*, 2019. 2

[15] Henggang Cui, Vladan Radosavljevic, Fang-Chieh Chou, Tsung-Han Lin, Thi Nguyen, Tzu-Kuo Huang, Jeff Schneider, and Nemanja Djuric. Multimodal trajectory predictions for autonomous driving using deep convolutional networks. In *ICRA*, 2019. 2

[16] Nemanja Djuric, Vladan Radosavljevic, Henggang Cui, Thi Nguyen, Fang-Chieh Chou, Tsung-Han Lin, Nitin Singh, and Jeff Schneider. Uncertainty-aware short-term motion prediction of traffic actors for autonomous driving. In *WACV*, 2020. 1, 2

[17] Davi Frossard and Raquel Urtasun. End-to-end learning of multi-sensor 3d tracking by detection. In *ICRA*, 2018. 2

[18] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In *CVPR*, 2018. 2

[19] Hasith Karunasekera, Han Wang, and Handuo Zhang. Multiple object tracking with attention to appearance, structure, motion and size. *IEEE Access*, 7:104423–104434, 2019. 2

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2014. 6

[21] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. 5

[22] 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. 5

[23] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *CVPR*, 2017. 2

[24] Philip Lenz, Andreas Geiger, and Raquel Urtasun. Followme: Efficient online min-cost flow tracking with bounded memory and computation. In *ICCV*, 2015. 2

[25] Junwei Liang, Lu Jiang, Juan Carlos Niebles, Alexander Hauptmann, and Li Fei-Fei. Peeking into the future: Predicting future person activities and locations in videos. In *CVPR*, 2019. 2

[26] Ming Liang, Bin Yang, Yun Chen, Rui Hu, and Raquel Urtasun. Multi-task multi-sensor fusion for 3d object detection. In *CVPR*, 2019. 2

[27] Ming Liang, Bin Yang, Shenlong Wang, and Raquel Urtasun. Deep continuous fusion for multi-sensor 3d object detection. In *ECCV*, 2018. 2

[28] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. 11

[29] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *CVPR*, 2018. 1, 2, 3, 5, 6, 7, 8

[30] Wei-Chiu Ma, De-An Huang, Namhoon Lee, and Kris M Kitani. Forecasting interactive dynamics of pedestrians with fictitious play. In *CVPR*, 2017. 2

[31] James Malcolm, Yogesh Rathi, and Allen Tannenbaum. Multi-object tracking through clutter using graph cuts. In *ICCV*, 2007. 2

[32] Gregory P Meyer, Jake Charland, Darshan Hegde, Ankit Laddha, and Carlos Vallespi-Gonzalez. Sensor fusion for joint 3d object detection and semantic segmentation. In *CVPR Workshops*, 2019. 2

[33] Gregory P Meyer, Ankit Laddha, Eric Kee, Carlos Vallespi-Gonzalez, and Carl K Wellington. Lasernet: An efficient probabilistic 3d object detector for autonomous driving. In *CVPR*, 2019. 2

[34] Anton Milan, Konrad Schindler, and Stefan Roth. Multi-target tracking by discrete-continuous energy minimization. *TPAMI*, 38(10):2054–2068, 2015. 2, 4

[35] 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. 2

[36] Nicholas Rhinehart, Rowan McAllister, Kris Kitani, and Sergey Levine. Precog: Prediction conditioned on goals in visual multi-agent settings. In *ICCV*, 2019. 2

[37] Samuel Schulter, Paul Vernaza, Wongun Choi, and Manmohan Chandraker. Deep network flow for multi-object tracking. In *CVPR*, 2017. 2

[38] Sarthak Sharma, Junaid Ahmed Ansari, J Krishna Murthy, and K Madhava Krishna. Beyond pixels: Leveraging geometry and shape cues for online multi-object tracking. In *ICRA*, 2018. 2

[39] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *CVPR*, 2019. 2

[40] Andrea Simonelli, Samuel Rota Rota Bulò, Lorenzo Porzi, Manuel López-Antequera, and Peter Kontschieder. Disentangling monocular 3d object detection. In *ICCV*, 2019. 5

[41] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017. 11

[42] Siyu Tang, Bjoern Andres, Miykhaylo Andriluka, and Bernt Schiele. Subgraph decomposition for multi-target tracking. In *CVPR*, 2015. 2

[43] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *CVPR*, 2019. 2

[44] Xinshuo Weng and Kris Kitani. A baseline for 3d multi-object tracking. *arXiv preprint arXiv:1907.03961*, 2019. 2, 5

[45] Yu Xiang, Alexandre Alahi, and Silvio Savarese. Learning to track: Online multi-object tracking by decision making. In *ICCV*, 2015. 2

[46] Danfei Xu, Dragomir Anguelov, and Ashesh Jain. Pointfusion: Deep sensor fusion for 3d bounding box estimation. In *CVPR*, 2018. 2

[47] Xu Yan, Xuqing Wu, Ioannis A. Kakadiaris, and Shishir K. Shah. To track or to detect? an ensemble framework for optimal selection. In *ECCV*, 2012. 2

[48] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018. 2

[49] Bin Yang, Ming Liang, and Raquel Urtasun. Hdnet: Exploiting hd maps for 3d object detection. In *CoRL*, 2018. 2, 3, 11

[50] Bin Yang, Wenjie Luo, and Raquel Urtasun. Pixor: Real-time 3d object detection from point clouds. In *CVPR*, 2018. 2, 3, 6

[51] Zetong Yang, Yanan Sun, Shu Liu, Xiaoyong Shen, and Jiaya Jia. Std: Sparse-to-dense 3d object detector for point cloud. In *ICCV*, 2019. 2

[52] Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. End-to-end interpretable neural motion planner. In *CVPR*, 2019. 1, 2, 3

[53] Wenwei Zhang, Hui Zhou, Shuyang Sun, Zhe Wang, Jianping Shi, and Chen Change Loy. Robust multi-modality multi-object tracking. In *ICCV*, 2019. 2

[54] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, 2018. 2

[55] Benjin Zhu, Zhengkai Jiang, Xiangxin Zhou, Zeming Li, and Gang Yu. Class-balanced grouping and sampling for point cloud 3d object detection. *arXiv preprint arXiv:1908.09492*, 2019. 5, 7

Appendix

A. Backbone Network Architecture

We first show in Figure 6 the architecture of the backbone network. We use bird's eye view (BEV) occupancy map as input representation for the LiDAR, and concatenate the BEV representations of multi-sweep LiDAR point clouds (1 current frame + N previous frames) along the height dimension. We employ $N = 9$ in nuScenes dataset and $N = 4$ in ATG4D dataset. For map representation, we exploit both geometric and semantic priors similar to HDNet [49]. Specifically, we do ground height subtraction on the multi-sweep LiDAR point clouds, and compute two additional binary raster images representing the drivable region and the lane graph respectively (for the current frame only). The map raster images are concatenated with the LiDAR BEV representation along the height dimension.

Given the aforementioned BEV representation of both LiDAR and HD maps as input, we first apply three `Conv2D` layers to down-sample the input BEV images by a factor of 4. We then apply a cross-scale module sequentially three times. This cross-scale module is inspired by the Inception block with residual connections [41]. The difference is that feature maps are spanned at multiple scales (3 in our case), and each scale receives information from all other scales. This leads to a better trade-off between accuracy and speed. After three cross-scale modules, we apply a feature pyramid network [28] to combine multi-scale feature maps, resulting in a $4\times$ down-sampled BEV feature map with 128 channels. For the detection header we simply use 4 `Conv2D` layers each with 128 filters. The detection header outputs $(6+1)*C$ channels as dense detection estimations, which correspond to $(x, y, w, l, \sin\theta, \cos\theta, \text{score\_logit})$ for each object category.

B. Implementation Details on nuScenes

We use the same network architecture on both nuScenes and ATG4D datasets. On nuScenes, following the dataset rule imposed by the creators of the dataset, we aggregate 10 sweeps of LiDAR point clouds (1 current and 9 previous) corresponding to 0.5 seconds of past history. We consider the point clouds within a region of $[-50, 50] \times [-50, 50] \times [-3, 5]$ meters around the ego car. We use a voxel size of $0.15625 \times 0.15625 \times 0.25$ meters, leading to a voxel grid size of $640 \times 640 \times 320$ as input.

We apply frame-level data augmentation during training. Specifically, labels at non-key frames are linearly interpolated from labels at adjacent key frames. For each frame, we apply random scaling ($0.95 \sim 1.05$ for all 3 axes), translation (-1 $\sim$ 1 meters for XY axes and -0.2 $\sim$ 0.2 meter for Z axis), rotation (-45 $\sim$ 45 degrees along Z axis) and flipping (along X axis) to both 3D LIDAR point clouds and 3D object labels.

The model is trained on the car class, and we ignore labels that have 0 LiDAR point inside the box or outside the 50 meters range with respect to the ego car. During training we define positive samples as voxels with IoU (assuming ground-truth size and orientation) larger than 0.9, and define negative samples as smaller than 0.4 IoU. We use Adam optimizer and train with batch size of 8 for 4 epochs. The initial learning rate is 0.001, and is decayed by 0.1 at 2.8 and 3.6 epochs respectively.

C. Fine-Grained Evaluation of Motion Forecasting

While in the main paper we evaluate prediction errors (ADE and FDE) of motion forecasting at fixed object recall rates, here we provide more fine-grained evaluation of FDE (at 1s, 2s and 3s respectively) for all recall rates. We show evaluation results in Figure 7, where we observe that the proposed PnPNet not only achieves higher object detection recall, but also outperforms the baseline in prediction consistently at all recall rates.
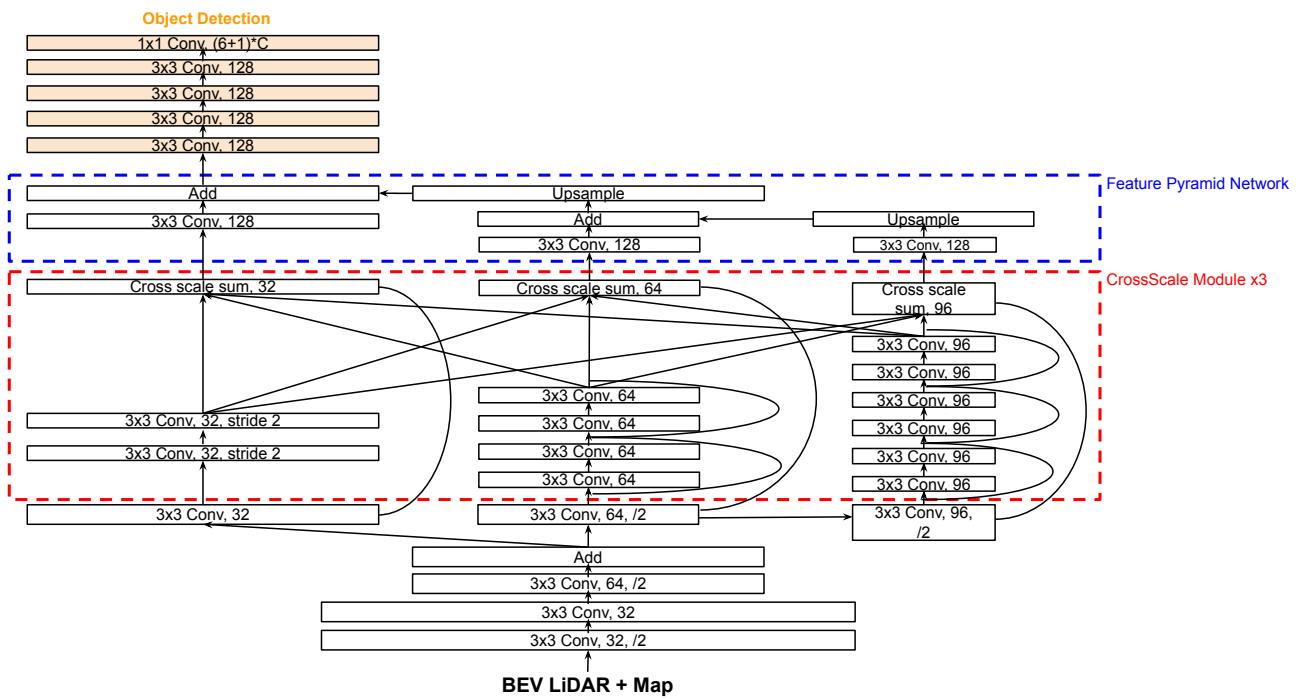
Object Detection

| 1x1 Conv, (6+1)*C |
| 3x3 Conv, 128 |
| 3x3 Conv, 128 |
| 3x3 Conv, 128 |
| 3x3 Conv, 128 |

Feature Pyramid Network

Add	Upsample
3x3 Conv, 128	Add	Upsample
	3x3 Conv, 128	3x3 Conv, 128

CrossScale Module x3

| Cross scale sum, 32 | Cross scale sum, 64 | Cross scale sum, 96 |

		3x3 Conv, 96
		3x3 Conv, 96
	3x3 Conv, 64	3x3 Conv, 96
3x3 Conv, 32, stride 2	3x3 Conv, 64	3x3 Conv, 96
3x3 Conv, 32, stride 2	3x3 Conv, 64	3x3 Conv, 96
	3x3 Conv, 64	3x3 Conv, 96

| 3x3 Conv, 32 | 3x3 Conv, 64, /2 | 3x3 Conv, 96, /2 |

| Add |
| 3x3 Conv, 64, /2 |
| 3x3 Conv, 32 |
| 3x3 Conv, 32, /2 |

BEV LiDAR + Map

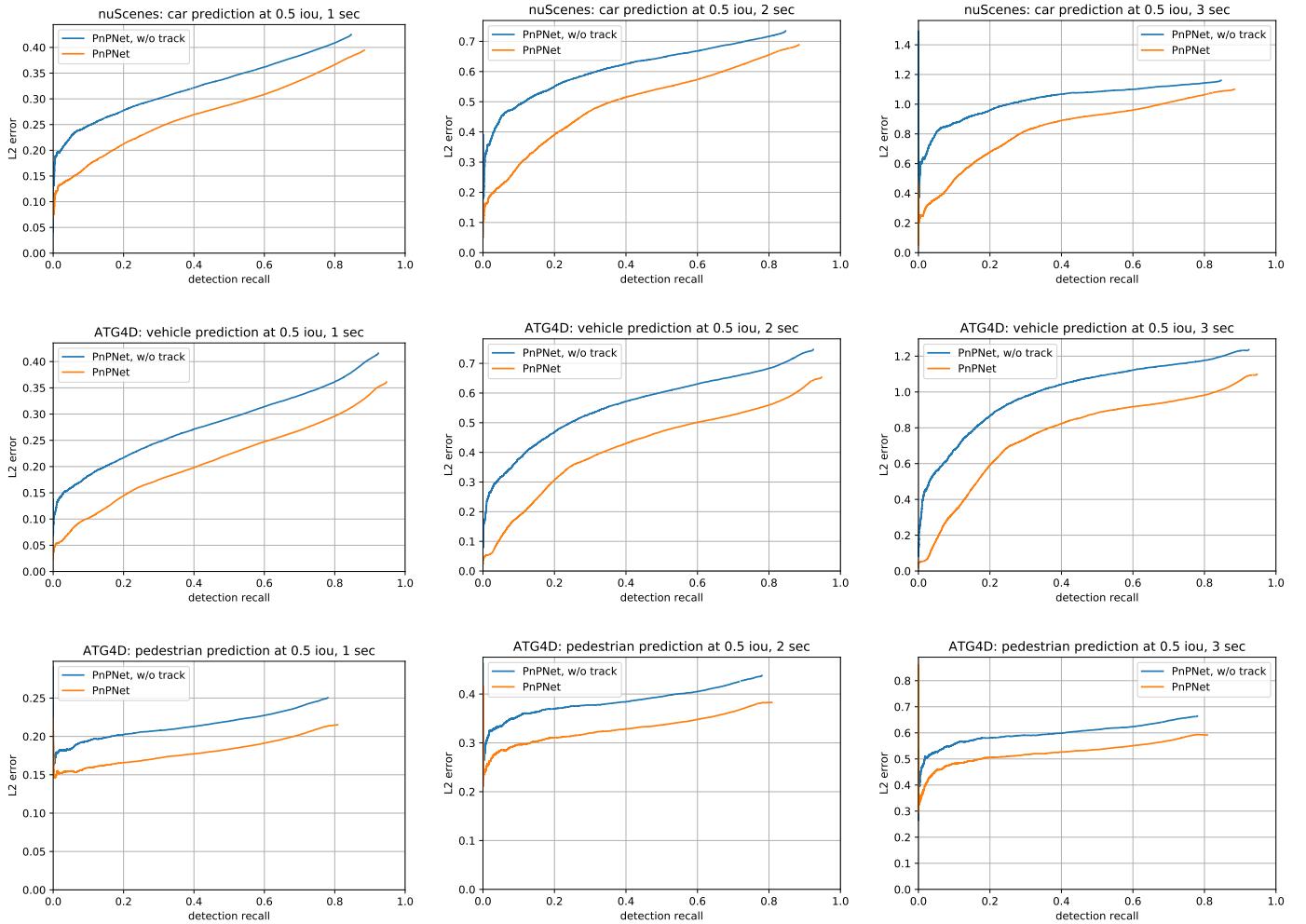Figure 6. **Architecture of the backbone network.**

Figure 7. **Final Displacement Error (FDE) at 1s, 2s, 3s prediction for all recall rates on nuScenes and ATG4D datasets.**