Learning from All Vehicles

Dian Chen Philipp Krähenbühl
UT Austin

Abstract

In this paper, we present a system to train driving policies from experiences collected not just from the ego-vehicle, but all vehicles that it observes. This system uses the behaviors of other agents to create more diverse driving scenarios without collecting additional data. The main difficulty in learning from other vehicles is that there is no sensor information. We use a set of supervisory tasks to learn an intermediate representation that is invariant to the viewpoint of the controlling vehicle. This not only provides a richer signal at training time but also allows more complex reasoning during inference. Learning how all vehicles drive helps predict their behavior at test time and can avoid collisions. We evaluate this system in closed-loop driving simulations. Our system outperforms all prior methods on the public CARLA Leaderboard by a wide margin, improving driving score by 25 and route completion rate by 24 points.

1. Introduction

Autonomous driving has been one of the most anticipated technologies since the advent of modern-day artificial intelligence. However, even after decades of exploration, we have yet to see self-driving cars deployed at scale. One main reason is the generalization. The world and its drivers are more diverse than current planning approaches can handle. Hand-designed classical planning [3, 16, 29, 45] does not generalize gracefully to unseen or unfamiliar scenarios. Learning based methods [4, 9, 11, 14, 37] fare better, but suffer from a long tail of driving scenarios. The majority of driving data consist of easy and uninteresting behaviors. After all, humans drive thousands of hours before observing a traffic accident [43], especially when driving an expensive autonomous test vehicle. How do we tame the long-tail of driving scenes? While many approaches rely on carefully crafted safety-critical scenarios in simulation [33, 36, 42], or collect massive data in the real world [4, 41], in this paper we focus on an orthogonal direction.

We observe that, although many of us have not experienced traffic accidents ourselves, everyone has at least observed several accidents throughout our driving career. The same applies to safety-critical driving scenarios: While the data-collecting ego-vehicle might not experience accident-prone situations itself, it is likely its driving logs contain states that are interesting or safety-critical, but experienced by *other vehicles*. Training on other vehicles' trajectories helps not only with sample efficiency, but also greatly increase the chance that the model sees interesting scenarios. Moreover, knowing other vehicles' future trajectories helps the ego-vehicle avoid collisions.

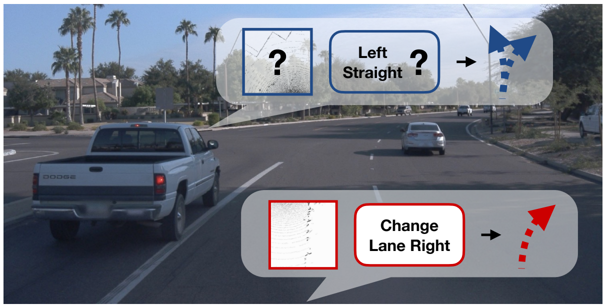


Figure 1. We present LAV, a mapless, learning-based end-to-end driving system. LAV takes as input multi-modal sensor readings and learns from all nearby vehicles in the scene for both perception and planning. At test time, LAV predicts multi-modal future trajectories for all detected vehicles, including the ego-vehicle. Picture credit – Waymo open dataset [41].

The main challenge with training on all vehicles lies in the partial observability of other vehicles. Unlike the ego-vehicle, other vehicles have only partially observed motion trajectories, exposing no control commands or higher-level goals. This makes direct training [10, 11, 12, 14, 38] on other vehicles' traces close to impossible. More importantly, other vehicles have no accessible sensors. To learn from other vehicles, a model has to infer their surrounding state using the ego-vehicle's sensors.

Our framework, **L**earning from **A**ll **V**ehicles (LAV), handles the partial observability of both perception and motion in one joint recognition, prediction, and planning stack. We decouple the partial observability challenge of perception and action using a privileged distillation approach [11]. LAV first learns a perception model that outputs a viewpoint invariant representation using auxiliary supervision from 3D detection and segmentation tasks. By definition, this auxiliary task does not distinguish between the ego-vehicle and other vehicles in the scene and thus learns a viewpoint invariant representation. It handles the partial observability of sensors. In parallel, LAV learns a privileged motion planner [11]. Instead of predicting steering and acceleration, which are only available for the ego-vehicle, we use future waypoints to represent the motion plan. We use ground-truth computer-vision labels as inputs to the privileged motion planner. Computer-vision labels ensure viewpoint invariance, waypoints provide an invariant representation of motion. The privileged motion planner predicts trajectories of all nearby vehicles and infers their high-level commands. Finally, we combine the two models in a joint framework using privileged distillation [11]. This final distillation learns a motion prediction model from all vehicles using the viewpoint invariant vision features of the perception model. The distilled policy drives from raw sensor inputs alone.

We validate our method in the CARLA driving simulator [17]. At the time of submission, our method ranks first on the CARLA public leaderboard[1]. It attains a $\mathbf{61.85}$ driving score and a $\mathbf{94.46}$ route completion rate. Both are the highest among all methods and outperform the prior state-of-the-art method by a wide margin, increasing driving score and route completion rate by **25** and **24** points respectively. Our method has also won the 2021 CARLA Autonomous Driving challenge[2]. Code available at https://github.com/dotchen/LAV.

2. Related Work

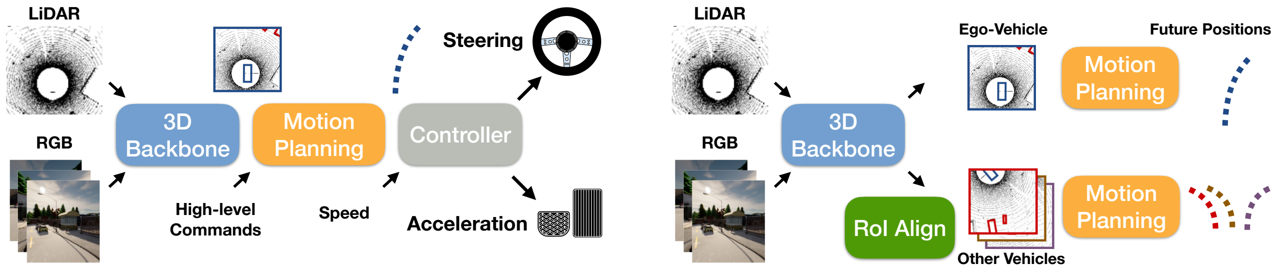
Perception for autonomous driving is driven by advances in visual understanding and recognition. The perception system of a self-driving vehicle understands the scene by inferring its nearby objects and surrounding road structures. A typical perception system takes as input LiDAR scans and performs object detection and tracking [19, 27, 47, 52, 54]. Liang et al. [32], Vora et al. [46] fuse RGB camera and LiDAR scans for richer semantic information. For roads, perception systems are categorized based on whether they require pre-recorded HD-Map: map-based systems localize themselves in the pre-recoded maps [30, 35, 51]; mapless systems either perform online mapping [6, 21, 22], or they implicitly predict road-related affordances [9, 40, 44]. Bansal et al. [4], Zeng et al. [48] represent the perception outputs as bird's-eye-viewed (BEV) spatial grids; more recently, Gao et al. [20], Li et al. [31] represent perception outputs in a parameterized vector space for a more compact representation. Our approach takes multi-modal sensor data as input and performs online mapping and object detection. However, we do not directly use the predicted map to perform classical planning. Instead, we learn a planner from data using imitation learning. This planner uses every vehicle it encounters on the road as a supervisory signal to enhance the diversity of the training data.

Behavior prediction focuses on forecasting the future state of driving scenes. In autonomous driving, a behavior predictor takes as either the input representations obtained from perception or raw sensor data; it predicts trajectories of the dynamic objects in the driving scene. Luo et al. [34], Zeng et al. [48] predict single, deterministic future trajectories of the detected vehicles. Casas et al. [5], Zhao et al. [50] model multi-modal future trajectories by using conditional models. Chai et al. [7] predicts trajectories as Gaussian mixtures to represent uncertainty in the euclidean space. Cui et al. [15], Lee et al. [28] use latent variables and VAEs to model actor and scene specific uncertainties. Recently, Casas et al. [6], Hu et al. [23], Kamenev et al. [25] combine perception and behavior prediction by directly predicting the occupancy maps. Our approach is highly related to the task of behavior prediction, as it also trains on all nearby vehicles' trajectories. Our approach consists of a behavior predictor. In particular, it applies a conditional motion planner on all nearby vehicles, including the ego-vehicle.

Learning-based motion planning uses imitation learning or reinforcement learning to plan future trajectories. Pioneered by Pomerleau [37], imitation learning for autonomous driving regresses sensor inputs to controls by imitating the recorded expert trajectories. Codevilla et al. [14] use conditional branching and high-level commands to extend imitative models for urban driving. Zeng et al. [48] use imitation learning to train a cost volume predictor for planning; Chen et al. [9], Sauer et al. [40] predict actions from the learned affordances. Chen et al. [11] uses on-policy distillation to handle distribution shift as well as to provide stronger imitative supervision signals. Reinforcement learning, on the other hand, trains policies from a user-defined reward function. Kendall et al. [26] train a lane following driving policy using DDPG; Toromanoff et al. [44] use distributed Rainbow-IQN to train an urban driving policy with competitive performance. Recently, Chen et al. [10] use model-based reinforcement learning and distillation to train a driving policy in an offline manner. Our approach builds upon Chen et al. [11] and trains a motion planner using imitation learning and distillation. However, unlike most prior methods, we train motion planning on data from all nearby vehicles in addition to the ego-vehicle.

Our idea of training the ego motion planner using data from all vehicles is closely related to Filos et al. [18] and Zhang and Ohn-Bar [49]. Filos et al. [18] extends offline reinforcement learning to learn from other agents' behaviors. Zhang and Ohn-Bar [49] train a privileged imitation learning policy that learns from other vehicles in a scene. Their policy

[1] https://leaderboard.carla.org/leaderboard/
[2] https://ml4ad.github.io/



(a) Overview of the inference pipeline.

(b) Overview of the training pipeline for the motion planning module.

Figure 2. Overview of the agent's pipeline. A 3D Backbone fuses LiDAR measurements and semantic segmentation from RGB cameras to produce a 2D spatial feature map. This shared feature map serves as an input to a motion planner. At inference time (a), we use the central crop to predict the ego-vehicles trajectory. At training time (b), we additionally use ground-truth detections of nearby vehicles to train a motion planner from all visible vehicles. Detection results use rotated regions of interest (RoIs) of the shared feature map. Finally, at inference time, a controller aggregates multiple motion predictions into a single steering and acceleration command.

side-steps partial observability by training a policy that acts only on the ground truth state of the simulator. It assumes perfect perception or access to other agents' sensors. LAV, on the other hand, operates on raw sensor inputs and learns a viewpoint invariant intermediate representation.

3. Learning from All Vehicles

We aim to build a deterministic driving model π that at each timestep t maps sensor readings, high-level navigational command, and vehicle state to raw control command a_t. We opt for an end-to-end differentiable three-stage modular pipeline: A perception module, a motion planner, and a low-level controller. See Figure 2a for an overview.

The perception module is trained from massive labeled supervision with two goals in mind: To create a robust and generalizable representation of the surrounding world, and to build vehicle-invariant features that help supervise the motion planner. Section 3.1 describes the overall architecture and training setup of the perception module. It maps raw sensor readings to a map-view feature representation.

The motion planner uses the map-view features of the perception model to produce a series of waypoints describing the future trajectory of the vehicles. Motion planners commonly use supervision from just the ego-vehicle for this prediction [11]. This supervision is quite sparse and provides the motion planner with just a single series of labels per collected data point. In our framework, we learn motion planning from all vehicles that surround the ego-vehicle. This is possible because our perception system produces vehicle-invariant features as inputs; it is also because the outputs of the motion planner, the future trajectories, can be easily obtained from ground truth driving logs. Figure 2b shows an overview of the motion planner training. Section 3.2 describes the motion planner and its training setup.

Finally, a low-level controller converts motion plans into actual steering and acceleration commands that are executed on the vehicle. At test time, the low-level controller considers other vehicles' motion plans to make emergency stop decisions. Section 3.3 describes the controller.

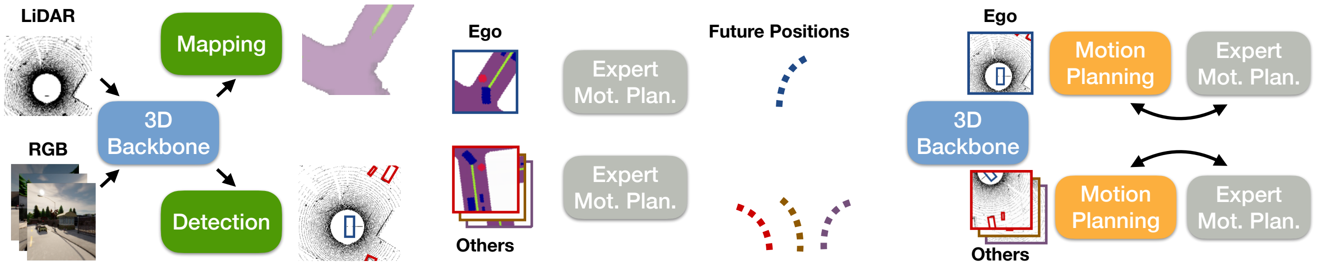
3.1. A vehicle-independent perception model

The core objective of any perception module is to build an intermediate representation that readily generalizes from training to test conditions. In our setup, a secondary goal is to build input features to the motion planner that are indistinguishable between the current vehicle and nearby vehicles. The closer the output representations of the ego-vehicle and other vehicles are, the better motion plans transfer between those vehicles. Here, we opt for a metric map-based output representation. In a metric map, rotated ROI pooling extracts fixed-sized feature representations for training vehicles.

Specifically, we use three RGB cameras $\mathbf{I}_t = \{I_t^1, I_t^2, I_t^3\}$ surrounding the vehicle and one LiDAR sensor L_t as an input. We combine the color and LiDAR inputs using point-painting [46] from RGB inputs and a light-weight CenterPoint [47] with PointPillars [27] 3D backbone. The backbone provides us with a map-view feature representation $f \in \mathbb{R}^{W \times H \times C}$ of width W and height H with C channels.

We train the backbone network using a combination of semantic segmentation and detection losses. See Figure 3a for an overview. For every pixel in map-view, we predict a road mask, solid and broken lane boundaries. We use a binary classifier, and binary cross-entropy loss, as road and lane-marking can overlap. In addition, we train a CenterPoint-style detector [47] for pedestrians and vehicles. Most importantly, we explicitly label the ego-vehicle in this detector. This minimizes the feature distance between ego-vehicle and other vehicles and enables better transfer. We pre-train the perception model using fully labeled data and use rotation augmentations around the ego-vehicle to increase the robustness of the learned model.

Supervised pre-training has two advantages. It generalizes better to unseen test conditions. It also learns a similar feature representation for all vehicles. This feature representation is next used in the motion planner.



(a) Perception training. (b) Privileged motion models training. (c) Final distillation.

Figure 3. Overview of our training pipeline. **(a)** We train a 3D perception model using detection and semantic mapping as the supervision signal. Both tasks help learn a viewpoint-invariant spatial representation. Detection additionally predicts other vehicles' poses which we use to forecast their future trajectories at inference. The perception module produces a vehicle-independent feature representation used in motion planning. **(b)** In parallel, we train a motion planner over ground truth perception. We train the model using traces from all nearby vehicles using their future trajectory as supervision. **(c)** Finally, we combine the models learned in **(a)** and **(b)** using distillation. This model learns how all vehicles plan in an end-to-end manner using only the ego-vehicles sensor inputs.

3.2. Learning to plan motion from all vehicles

The motion planner uses the output of the perception system to predict a series of future waypoints describing positions the vehicles should steer towards. Here, we propose a novel two-stage motion planner that combines geometric GPS targets and discrete high-level commands. We use a standard RNN formulation [28, 38] to predict $n = 10$ future waypoints $y_1, \ldots, y_n \in \mathbb{R}^2$. The motion planner uses a high-level command c and intermediate GNSS coordinate goal $g \in \mathbb{R}^2$ to perform different driving maneuvers. In CARLA, GNSS goals are sampled every 50-100 meters and contain a measurement error of around one meter. Possible high-level commands c include *turn-left*, *turn-right*, *go-straight*, *follow-lane*, *change-lane-to-left*, *change-lane-to-right*.

Let $M(\hat{f}, c) :\to \mathbb{R}^{n\times 2}$ be the motion planner conditioned on high-level command c and warped features $\hat{f}$ for the Region of Interest (RoI) at the location and orientation of the vehicle in question. For all vehicles, we observe their future trajectory to obtain supervision for future waypoints y. For the ego-vehicle, the simulator provides a ground truth high-level command $\hat{c}$ and provides sufficient supervision to train the motion planner

$$\mathcal{L}_M^{ego} = \mathrm{E}_{\hat{f},y,\hat{c}}\left[\|y - M(\hat{f},\hat{c})\|_1\right]. \tag{1}$$

However, other vehicles do not expose their high-level commands. While it may be possible to infer this command from future trajectories alone, any rule-based inference will be ambiguous and noisy. We instead allow the model to infer the high-level command directly and optimize the plan for the most fitting high-level command.

$$\mathcal{L}_M^{other} = \mathrm{E}_{\hat{f},y}\left[\min_c \|y - M(\hat{f},c)\|_1\right]. \tag{2}$$

At training time we optimize both losses $\mathcal{L}_M^{ego} + \mathcal{L}_M^{other}$ jointly.

The resulting motion planner M finds good coarse trajectories for a wide range of traffic scenarios. It learns to plan for all vehicles it sees. However, the resulting motion plan may be noisy as high-level commands c are ambiguous.

In a second stage, we refine the motion plan using an additional RNN-based motion planning network $M'(\hat{f}, g, \tilde{y}) \in \mathbb{R}^{n\times 2}$. The motion refinement network uses the same ROI-warped feature $\hat{f}$, the previously predicted motion plan $\tilde{y}$, and the more fine-grain GNSS goal g as input. We normalize g in the ego-vehicle's coordinate. It then produces a delta to the original trajectory as output. Since GNSS goals are only available for the ego-vehicle, we train the refinement $\hat{M}$ only on ego-vehicle trajectories

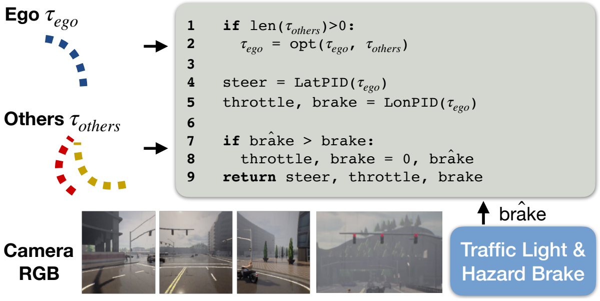
$$\mathcal{L}_M^{refine} = \mathrm{E}_{\hat{f},y,\tilde{y},\hat{g}}\left[\|\tilde{y} + M'(\hat{f},\hat{g},\tilde{y}) - y\|_1\right]. \tag{3}$$

During both training and testing, we roll out the same refinement network multiple times to recursively refine the predicted trajectory. The above loss then applies to each step of the rollout.

In practice, we learn the motion planner in a privileged distillation framework [11]. See Figure 3b and Figure 3c for an overview. We first learn motion planning on ground truth trajectories and ground-truth perception outputs and regions of interest using the losses (1)-(3). We then use the privileged motion planner to supervise a motion planner that uses the inferred perception outputs. During this second stage, we supervise predictions on all high-level commands which leads to a richer supervisory signal [11]. We additionally distill a high-level command classifier for other vehicles which we use later in the vehicle-aware controller. This stage trains end-to-end by backpropagating gradients from motion prediction and planning to the perception backbone, allowing perception models to attend to low-level details in the scene. We keep the pre-training perception loss in the previous stage as an auxilliary supervision to regularize the features.

3.3. Vehicle-aware control

The controller translates a motion plan into actual driving commands. We use two PID controllers for latitudinal (steering) and longitudinal (acceleration) control. Both PID controllers use basic statistics of the refined motion plan as an input to produce a continuous output command. The longitudinal PID controller additionally uses the current speed as an input to compute acceleration. We overwrite braking using a separate neural network classifier B in case of traffic light and hazard stoppages. The classifier uses the same image inputs as the perception module plus one additional camera with telephoto lenses to capture distant traffic lights. The classifier learns the braking behavior of the data-collecting ego-vehicle using recorded brake actions. Finally, we reuse the motion plans learned from other vehicles to detect potential collisions and perform hazard stops. Specifically, we use the 3D detections of the backbone to find all nearby vehicles. For each, we use the motion planner M to produce future trajectories over each high-level command. We use all motion plans above the high-level command likelihood threshold to check for collisions with the ego-vehicle's motion plan.



```
if len(τ_others)>0:
  τ_ego = opt(τ_ego, τ_others)

steer = LatPID(τ_ego)
throttle, brake = LonPID(τ_ego)

if bra^ke > brake:
  throttle, brake = 0, bra^ke
return steer, throttle, brake
```

Figure 4. Overview of our controller logic. The controller considers all vehicles and their predicted multi-modal future trajectories. It additionally uses an image-only brake predictor to handle traffic sign and hazard stoppages.

Perception. We use PointPillars [27] with PointPainting [46] as our multi-modal 3D perception backbone P_B. In particular, given RGB images captured from three frontal facing camera $\{I_t^0, I_t^1, I_t^2\}$ with extrinsic matrices $E = \{E^0, E^1, E^2\}$, we use a ERFNet [39] to compute their semantic segmentation scores $\mathbf{S}_t = \{S_t^0, S_t^1, S_t^2\}$. We use five semantic classes: "background", "vehicles", "roads", "lane markings" and "pedestrians". For each LiDAR point $l \in L_t$, we use PointPainting [46] to concatenate its corresponding semantic classes using the segmentation scores: $l_t^s = \text{PointPaint}(\mathbf{S}_t, \mathcal{T}_E, l_t)$. $\mathcal{T}_E$ is the perspective transform function.

For PointPillars, we use FC-64-64 with BatchNorm [24] as its PointNet. We create pillars for LiDAR points for $x \in [-10\text{m}, 70\text{m}]$ and $y \in [-40\text{m}, 40\text{m}]$. Each pillar represents a $0.25\text{m} \times 0.25\text{m}$ spatial region. We use the default 2D CNNs with multi-scale features to obtain the spatial features $\phi_t \in \mathbb{R}^{192 \times 160 \times 160}$ with $0.5\times$ resolution of the original pillars. Unlike the original PointPillars which directly builds dense pillars specified by the hyperparameters, we sparsely represent the pillars. We also use a sparse PointNet to process the corresponding sparse pillar features. This allows us to process all pillars efficiently both in space and time.

We use a branching architecture for the detection and mapping heads. We use a simplified one-stage CenterPoint [47, 53] formulation for BEV object detection. In particular, we predict two "centerness" maps, one for vehicles and one for pedestrians; we also predict an orientation and bounding-box maps that are both class-agnostic. For mapping, we predict a BEV semantic map for roads, solid lane markings and broken lane markings. Each map is generated using a separate 3×3 convolution followed by a 3×3 up-convolution with stride 2, all from the shared backbone P_B. At test time, we use a 2D max pooling layer as a simplified version of NMS.

We additionally train a binary brake classifier that takes as input all the four camera RGB images. We feed the telephoto lens image and the stitched other three images to a ResNet-18 followed by a global average pooling layer. This gives us fixed-sized embeddings of $z_1, z_2 \in \mathbb{R}^{512}$. We concatenate z_1, z_2 and feed it to a linear layer to predict the binary brake.

Prediction and Planning. Given the ego-vehicle and a list of vehicle detection, we use differentiable warping to crop a rotated region of interest (RoI) $\hat{f}^i$ for each vehicle location and yaw angle. A CNN followed by global average pooling takes as input the rotated RoI features and returns a fixed-sized embedding z^i for each vehicle i. z^i is shared among M and M'. The motion planner M uses a separate GRU [13] for each high-level command. The GRU is rolled out n times to produce an offset between consecutive waypoints. The refinement motion planner M' uses two forms of recursions and rollouts: rollouts along waypoint and rollouts along refinement iterations. It predicts an offset from the prior motion plan for each iteration. The refinement motion plan relies on just a single GRU unit that takes the GNSS goal g as an additional input. Both motion planners use a linear layer to transform GRU states into the desired outputs.

Control. The controller C takes as input refined ego-trajectory $\tau = M'(\hat{f}, \tilde{y}, \hat{g})$. See Figure 4 for an overview. If predicted trajectory τ leads to a collision with other traffic participants, we adjust it. For now we perform a hard stop using a hard-coded braking logic. If the predicted trajectory is collision free, we follow it directly. We use two PID controllers for latitudinal and longitudinal control respectively. For latitudinal control, we use the 5-th point in τ_5 as the aim

Rank	Method	Driving Score	Route Completion	Infraction Score
1	**LAV**	**61.85**	**94.46**	0.64
2	GRIAD [8]	36.79	61.85	0.60
3	TransFuser+ [2]	34.58	69.84	0.56
4	Rails [10]	31.37	57.65	0.56
5	IARL [44]	24.98	46.97	0.52
6	NEAT [12]	21.83	41.71	0.65
7	TransFuser [38]	16.93	51.82	0.42
8	LBC [11]	8.94	17.54	**0.73**

Table 1. Comparison of the driving score (main metric), route completion and infraction score on the public CARLA leaderboard [1] (accessed March 2022). All three metrics are higher the better. Detailed infraction numbers reported in the supplement for reference.

point to compute the steering error. For longitudinal control, we use the difference between target speed inferred from $\|\tau_{k+1} - \tau_k\|$ and the current speed v_t to compute acceleration. We use $K_P = 1.0, K_I = 0.5, K_D = 0.2$ for the latitudinal PID controller, and we use $K_P = 5.0, K_I = 0.5, K_D = 1.0$ for the longitudinal PID controller. We overwrite the brake control with the predicted brake score if it is larger than the brake computed from the longitudinal controller.

4. Experiments

We evaluate our method on the CARLA simulator [17] using closed-loop driving. We compare our approach with the state-of-the-art methods on the public leaderboard, and we perform ablation study on the effect of our design choices locally. For our online leaderboard submission, we train on all the 8 publicly available towns using a dataset of 400K frames, collected with the CARLA behavior agent under randomized weathers. For ablations, we only train on 4 out of the 8 towns, resulting in a dataset of 186K frames. We test on two other unseen towns. Details of the dataset statistics are provided in the supplement for reference.

4.1. Comparison with state-of-the-art

Table 1 compares our method with prior state-of-the-art methods on the CARLA public leaderboard [1]. The CARLA leaderboard evaluates autonomous driving systems under unseen and partially adversarial conditions. Vehicles are tasked to complete a set of predefined routes in new towns. For each route, the simulator adds dangerous scenarios such as suddenly crossing pedestrians or aggressive lane-changing vehicles. These scenarios are modeled after the NHTSA typology [1]. The leaderboard measures how far self-driving vehicles proceed along a route within a fixed time budget, and how often they cause traffic infractions. In Table 1, we list three key metrics of the leaderboard: Driving Score, Route Completion, and Infraction Score. Route Completion measures the distance percentage an agent is able to complete; Infraction Score measures how often an agent drives without causing infractions; Driving Score measures route completion rate weighted by infractions per route. Driving Score and Route Completion are the two main metrics of comparison. A vehicle standing perfectly still will receive an infraction score of 1. All three metrics are higher the better. We refer readers to the official leaderboard [1] for a more detailed description of the metrics.

We compare to the top entries on the leaderboard. GRIAD [8] is the prior state-of-the-art. Rails [10] is a model-based reinforcement-learning method that trains vision-based driving policies from offline driving logs. IARL [44] is based on state-of-the-art model-free reinforcement-learning with distributed training. NEAT [12] uses imitation-learning with attention and implicit functions. Transfuser [38] uses imitation-learning with attention-based sensor fusion. LBC [11] relies on knowledge distillation with imitation learning. LBC is the closest comparison to our approach, as we also use knowledge distillation and imitation learning as a supervisory signal. However, LAV additionally uses other observed vehicles to train the control policy.

LAV ranks first on the leaderboard, and it outperforms the prior leading entry by a wide margin. It achieves $\mathbf{61.85}$ driving score, the highest among all methods, and $\mathbf{25}$ points higher than on the previous leading method GRIAD. It also achieves a $\mathbf{94.46}$ Route Completion, the highest among all methods and $\mathbf{24}$ points higher than the next best method, and $\mathbf{32}$ points higher than the previous state-of-the-art. Moreover, previous top methods, such as Rails and IARL, require 1M and 40M frames to train the policies. Our method uses only 400K training frames. Our approach has a relatively high infraction score; however, we note that higher Route Completion naturally leads to more infractions. A vehicle that drives slowly or stands still causes fewer infractions but struggles to complete its routes. See LBC for example.

4.2. Ablation study

We answer few important questions on our design choices. We again evaluate on Driving Score, Route Completion and Infraction Score. However, we cannot use the online leaderboard directly for these additional experiments. We instead use a local setup with similar characteristics to the Leaderboard. In particular, we train on 4 out of 8 towns (Town01, Town03, Town04 and Town06), and evaluate on 2 unseen towns (Town02 and Town05). We select 4 representative routes, 2 from each town, and we evaluate each route with 4 weathers: “Clear Noon”, “Cloudy Sunset”, “Soft Rain Dawn” and “Hard Rain Night”. We evaluate each setup for 3 runs and report the mean and standard deviation. This results in 48 trials for each model. All ablated models use a similar but slightly outdated setup as our leaderboard entry. The only differences are: 1. the ablated models use U-Net as the

	Driving Score	Route Completion	Infraction Score	Vehicle Collisions	Pedestrian Collisions	Layout Collisions	Red light Violations
LAV	**45.20** $\pm$ 6.35	**91.55** $\pm$ 5.61	0.49 $\pm$ 0.06	**0.92** $\pm$ 0.42	0.00 $\pm$ 0.00	**0.33** $\pm$ 0.50	0.28 $\pm$ 0.28
Ego-vehicle only	38.56 $\pm$ 1.86	84.76 $\pm$ 5.12	0.46 $\pm$ 0.02	1.17 $\pm$ 0.50	0.00 $\pm$ 0.00	1.82 $\pm$ 0.06	0.34 $\pm$ 0.20
No distillation	28.23 $\pm$ 2.27	81.05 $\pm$ 6.04	0.36 $\pm$ 0.04	2.08 $\pm$ 0.34	0.00 $\pm$ 0.00	7.87 $\pm$ 0.15	**0.21** $\pm$ 0.04

Table 2. Driving performance ablation of the key components of our approach on test towns. Infractions are measured as number of occurrences per kilometer traveled. Mean and standard deviation are computed over three runs.

semantic segmentation backbone 2. the ablated models use FC-32-32 in PointPillars and 3. slightly different controller hyperparameters.

Table 2 studies the effects of our key design choices. We compare to two variants of our approach: 1) One that only trains on ego-vehicle data, 2) One that does not perform privileged distillation. We find that training on other vehicles' trajectories and viewpoints results in lower performance on both route completion and infraction. The performance degradation is smaller than expected likely because our auxiliary perception supervision makes the motion models generalize well to distribution shifts, caused by both test time errors and viewpoints changes. Not using privileged distillation results in a larger performance drop. Without distillation, the motion models need to train from both noisy inputs and labels, thus tackling a much harder learning problem. Our full model achieves the highest scores in all three metrics.

Table 3 studies the degree to which training on other vehicles' experiences affect the driving performance. We evaluate three standards, where we only train vehicles within 5, 15 and 25 meters within the ego-vehicle, with 15 meters being our default value. $\leq$ 5m and $\leq$ 15m performs equally well, while $\leq$ 25m is slightly worse. We think this is due to the fact that vehicles at range have too different of an appearance in the sensor inputs which the auxiliary supervision is unable to correct for. The LiDAR sensor in CARLA mimics the Velodyne-64 rays model, which produces at most a few dozen measurements for cars at a distance of 25m.

Table 4 studies different perception training schemes. We compare the default staged training scheme (staged) with two variants of joint training: 1) No perception pre-training (No Pretrain.), and joint perception and motion training (Joint). The first variant only optimizes the distillation loss, and the latter optimize perception and distillation simultaneously. Both variants do not freeze the 3D backbone. As expected, models without perception training perform poorly. They suffer from the distribution shifts caused by viewpoint changes. Joint training also performs worse than staged straining, because solving perception and planning simultaneously is harder than solving them in a disentangled manner, as also observed by Chen et al. [11].

Vehicles Range	Driving Score	Route Completion	Infraction Score
$\leq$ 5m	**46.06** $\pm$ 1.70	88.77 $\pm$ 1.01	0.51 $\pm$ 0.02
$\leq$ 15m	45.20 $\pm$ 6.35	**91.55** $\pm$ 5.61	0.49 $\pm$ 0.06
$\leq$ 25m	37.42 $\pm$ 3.09	89.56 $\pm$ 5.61	**0.61** $\pm$ 0.12

Table 3. Driving performance in test towns of models trained with different range of other vehicles. All models are the same except for other vehicles' maximum range used during training.

Perception Training	Driving Score	Route Completion	Infraction Score
No Pretrain.	8.47 $\pm$ 0.83	9.34 $\pm$ 0.35	**0.90** $\pm$ 0.07
Joint	28.36 $\pm$ 2.11	79.58 $\pm$ 4.99	0.34 $\pm$ 0.02
Staged	**45.20** $\pm$ 6.35	**91.55** $\pm$ 5.61	0.49 $\pm$ 0.06

Table 4. Driving performance in test towns of models with different perception training scheme. All models are the same except for perception training.

Refinement Iteration	Driving Score	Route Completion	Infraction Score
$K = 0$	12.69 $\pm$ 2.86	35.85 $\pm$ 2.91	0.42 $\pm$ 0.03
$K = 1$	21.30 $\pm$ 1.10	85.90 $\pm$ 2.46	0.25 $\pm$ 0.01
$K = 5$	**45.20** $\pm$ 6.35	**91.55** $\pm$ 5.61	**0.49** $\pm$ 0.06

Table 5. Driving performance ablation on the effect of motion refinement. All models are the same except for number of refinement iterations.

Table 5 studies the effect of our iterative refinement module. $K = 0$ means we directly use the trajectories predicted by the motion planner M_f to drive. The default option $K = 5$ performs the best, showing the benefits of iterative motion refinement. Iterative refinement allows the model to elastically figure out what residuals to learn. It also naturally combines the semantic information from the high-level command and the geometric information from the goals.

Detailed infraction numbers for Table 3, Table 4 and Table 5 are provided in the supplement for reference.

4.3. Qualitative analysis

Figure 5 provides a qualitative analysis of our system. It shows the combined input images, LiDAR point cloud,

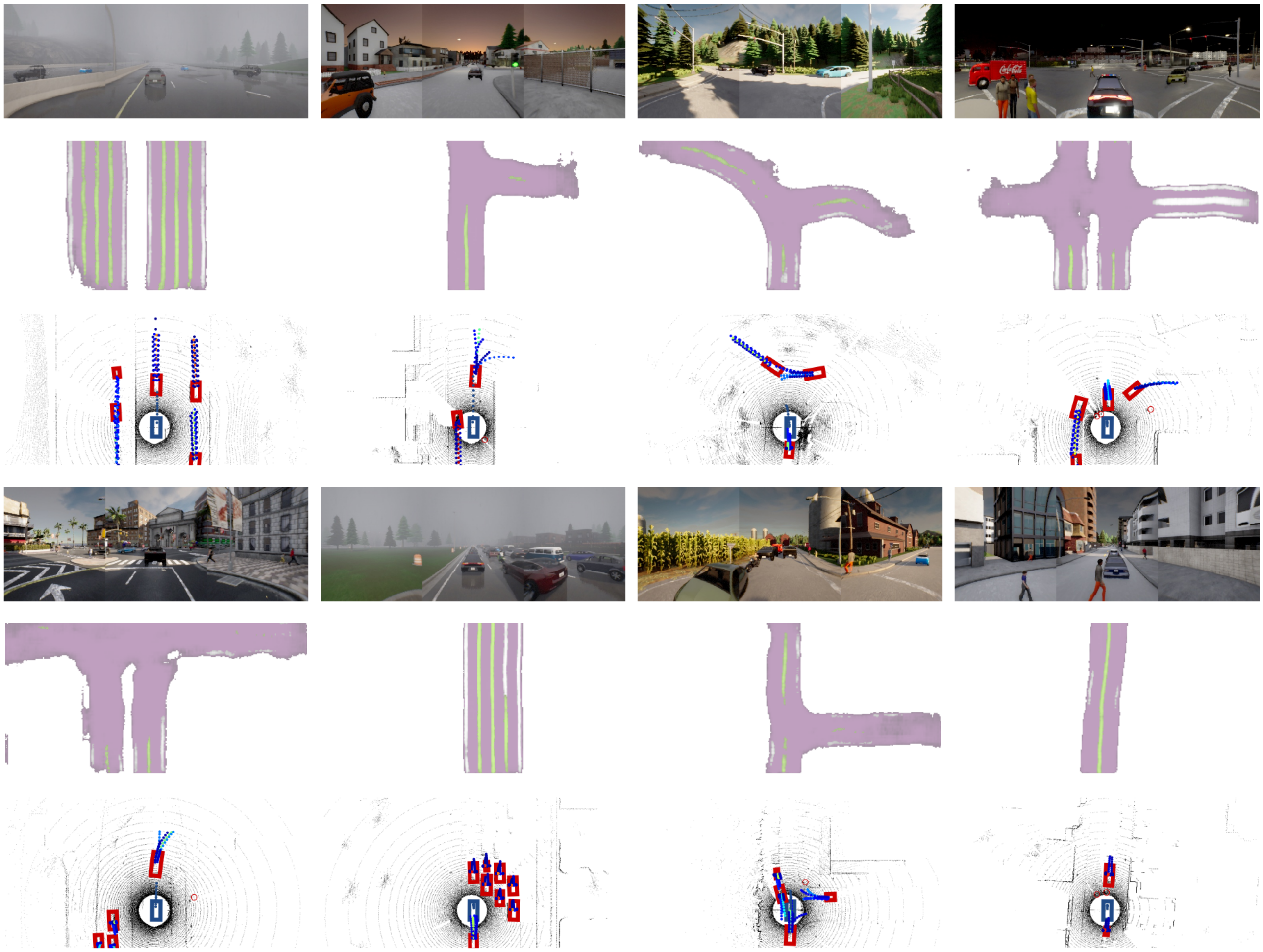


Figure 5. Visualizations of the outputs from our system. Each row visualizes RGB camera inputs, predicted road geometries, and detection and motion predictions respectively. Detection and motion prediction are used during inference; mapping is used for training only. For mapping, we predict **road**, **broken** and **solid** (white) lane markings. For detection, we predict pedestrians' and vehicles' poses and bounding boxes. We forecast multi-modal future trajectories with their corresponding likelihoods. Best viewed on screen.

the auxiliary segmentation predictions, detections, and predicted plans for all vehicles in the scene. The ego-vehicles plan uses the provided high-level command, while all other vehicles predict a distribution over possible future plans. Note how all vehicles predict a reasonable and consistent set of future plans aligning well with the inferred map-view representation of the road and the potential other vehicles.

5. Discussion

In this paper, we present a mapless, end-to-end driving system that trains from the experiences of all nearby vehicles. Our system achieves state-of-the-art performance in closed-loop driving simulation, and it outperforms prior leading methods by a wide margin. **Limitations and potential negative social impacts:** Our approach is trained and evaluated in simulation alone and still incurs traffic infractions. If directly deployed in the real world, it would most likely result in traffic accidents (negative social impacts). On the technical side, our current behavior predictor instantiated by the conditional motion planner does not consider multi-modality beyond the high-level commands. Extending our work with a probabilistic formulation will strengthen its ability in handling the diverse behaviors of both the ego vehicle and the other road users. Improving the motion predictor beyond its raster representation is also an exciting direction.

Acknowlegments

We thank Tianwei Yin for his help on pillar generation codes, Xingyi Zhou and Jeffrey Zhang for feedback on writing and figures. We thank TACC for providing part of our computing resources. This work was supported by the NSF Institute for Foundations of Machine Learning and NSF award #1845485.

References

[1] Carla autonomous driving leaderboard (accessed november 2021). https://leaderboard.carla.org/leaderboard/, 2021. 6

[2] Expert drivers for autonomous driving. url-https://kait0.github.io/files/master_thesis_bernhard_jaeger.pdf, 2021. 6

[3] Andrew Bacha, Cheryl Bauman, Ruel Faruque, Michael Fleming, Chris Terwelp, Charles Reinholtz, Dennis Hong, Al Wicks, Thomas Alberi, David Anderson, et al. Odin: Team victortango's entry in the darpa urban challenge. *Journal of field Robotics*, 2008. 1

[4] Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *RSS*, 2019. 1, 2

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

[6] Sergio Casas, Abbas Sadat, and Raquel Urtasun. Mp3: A unified model to map, perceive, predict and plan. In *CVPR*, 2021. 2

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

[8] Raphael Chekroun, Marin Toromanoff, Sascha Hornauer, and Fabien Moutarde. GRI: general reinforced imitation and its application to vision-based autonomous driving. *arXiv*, 2021. 6

[9] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *ICCV*, 2015. 1, 2

[10] Dian Chen, Vladlen Koltun, and Philipp Krähenbühl. Learning to drive from a world on rails. In *ICCV*, 2021. 1, 2, 6

[11] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *CoRL*, 2019. 1, 2, 3, 4, 6, 7

[12] Kashyap Chitta, Aditya Prakash, and Andreas Geiger. Neat: Neural attention fields for end-to-end autonomous driving. In *ICCV*, 2021. 1, 6

[13] Kyunghyun Cho, Bart van Merriënboer Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *EMNLP*, 2016. 5

[14] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *ICRA*, 2018. 1, 2

[15] Alexander Cui, Sergio Casas, Abbas Sadat, Renjie Liao, and Raquel Urtasun. Lookout: Diverse multi-future prediction and planning for self-driving. In *ICCV*, 2021. 2

[16] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. In *STAIR*, 2008. 1

[17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *CoRL*, 2017. 2, 6

[18] Angelos Filos, Clare Lyle, Yarin Gal, Sergey Levine, Natasha Jaques, and Gregory Farquhar. Psiphi-learning: Reinforcement learning with demonstrations using successor features and inverse temporal difference learning. *arXiv preprint arXiv:2102.12560*, 2021. 2

[19] Davi Frossard, Simon Suo, Sergio Casas, James Tu, Rui Hu, and Raquel Urtasun. Strobe: Streaming object detection from lidar packets. In *CoRL*, 2020. 2

[20] Jiyang Gao, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li, and Cordelia Schmid. Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In *CVPR*, 2020. 2

[21] Noa Garnett, Rafi Cohen, Tomer Pe'er, Roee Lahav, and Dan Levi. 3d-lanenet: end-to-end 3d multiple lane detection. In *CVPR*, 2019. 2

[22] Yuliang Guo, Guang Chen, Peitao Zhao, Weide Zhang, Jinghao Miao, Jingao Wang, and Tae Eun Choe. Genlanenet: A generalized and scalable approach for 3d lane detection. In *ECCV*, 2020. 2

[23] Anthony Hu, Zak Murez, Nikhil Mohan, Sofía Dudas, Jeff Hawke, Vijay Badrinarayanan, Roberto Cipolla, and Alex Kendall. Fiery: Future instance prediction in bird's-eye view from surround monocular cameras. In *ICCV*, 2021. 2

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015. 5

[25] Alexey Kamenev, Lirui Wang, Ollin Boer Bohan, Ishwar Kulkarni, Bilal Kartal, Artem Molchanov, Stan Birchfield, David Nistér, and Nikolai Smolyanskiy. Predictionnet: Real-time joint probabilistic traffic prediction for planning, control, and simulation. *arXiv preprint arXiv:2109.11094*, 2021. 2

[26] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *ICRA*, 2019. 2

[27] 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. 2, 3, 5

[28] 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, 4

[29] John Leonard, Jonathan How, Seth Teller, Mitch Berger, Stefan Campbell, Gaston Fiore, Luke Fletcher, Emilio Frazzoli, Albert Huang, Sertac Karaman, et al.

A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 2008. 1

[30] Jesse Levinson, Michael Montemerlo, and Sebastian Thrun. Map-based precision vehicle localization in urban environments. In *RSS*, 2007. 2

[31] Qi Li, Yue Wang, Yilun Wang, and Hang Zhao. Hdmapnet: An online hd map construction and evaluation framework. In *CVPR Workshop*, 2021. 2

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

[33] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *ITSC*, 2018. 1

[34] 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. 2

[35] Wei-Chiu Ma, Ignacio Tartavull, Ioan Andrei Bârsan, Shenlong Wang, Min Bai, Gellert Mattyus, Namdar Homayounfar, Shrinidhi Kowshika Lakshmikanth, Andrei Pokrovsky, and Raquel Urtasun. Exploiting sparse semantic hd maps for self-driving vehicle localization. In *IROS*, 2019. 2

[36] Zhenghao Peng, Quanyi Li, Ka Ming Hui, Chunxiao Liu, and Bolei Zhou. Learning to simulate self-driven particles system with coordinated policy optimization. In *NeurIPS*, 2021. 1

[37] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *NeurIPS*, 1989. 1, 2

[38] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. In *CVPR*, 2021. 1, 4, 6

[39] Eduardo Romera, José M Alvarez, Luis M Bergasa, and Roberto Arroyo. Erfnet: Efficient residual factorized convnet for real-time semantic segmentation. *ITS*, 2017. 5

[40] Axel Sauer, Nikolay Savinov, and Andreas Geiger. Conditional affordance learning for driving in urban environments. In *CoRL*, 2018. 2

[41] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *CVPR*, 2020. 1

[42] Simon Suo, Sebastian Regalado, Sergio Casas, and Raquel Urtasun. Trafficsim: Learning to simulate realistic multi-agent behaviors. In *CVPR*, 2021. 1

[43] Brian Tefft. Rates of motor vehicle crashes, injuries and deaths in relation to driver age, united states, 2014-2015. In *AAA Foundation for Traffic Safety.*, 2017. 1

[44] Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. End-to-end model-free reinforcement learning for urban driving using implicit affordances. In *CVPR*, 2020. 2, 6

[45] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 2008. 1

[46] Sourabh Vora, Alex H Lang, Bassam Helou, and Oscar Beijbom. Pointpainting: Sequential fusion for 3d object detection. In *CVPR*, 2020. 2, 3, 5

[47] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *CVPR*, 2021. 2, 3, 5

[48] 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. 2

[49] Jimuyang Zhang and Eshed Ohn-Bar. Learning by watching. In *CVPR*, 2021. 2

[50] Hang Zhao, Jiyang Gao, Tian Lan, Chen Sun, Benjamin Sapp, Balakrishnan Varadarajan, Yue Shen, Yi Shen, Yuning Chai, Cordelia Schmid, et al. Tnt: Target-driven trajectory prediction. In *CoRL*, 2020. 2

[51] Shuran Zheng and Jinling Wang. High definition map-based vehicle localization for highly automated driving: Geometric analysis. In *ICL-GNSS*, 2017. 2

[52] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Tracking objects as points. In *ECCV*, 2020. 2

[53] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019. 5

[54] 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 *CoRL*, 2020. 2