ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst

Mayank Bansal
Waymo Research
Mountain View, CA, USA
mayban@waymo.com

Alex Krizhevsky†
akrizhevsky@gmail.com

Abhijit Ogale
Waymo Research
Mountain View, CA, USA
ogale@waymo.com

***Abstract*—Our goal is to train a policy for autonomous driving via imitation learning that is robust enough to drive a real vehicle. We find that standard behavior cloning is insufficient for handling complex driving scenarios, even when we leverage a perception system for preprocessing the input and a controller for executing the output on the car: 30 million examples are still not enough. We propose exposing the learner to synthesized data in the form of perturbations to the expert's driving, which creates interesting situations such as collisions and/or going off the road. Rather than purely imitating all data, we augment the imitation loss with additional losses that penalize undesirable events and encourage progress – the perturbations then provide an important signal for these losses and lead to robustness of the learned model. We show that the *ChauffeurNet* model can handle complex situations in simulation, and present ablation experiments that emphasize the importance of each of our proposed changes and show that the model is responding to the appropriate causal factors. Finally, we demonstrate the model driving a real car at our test facility.**

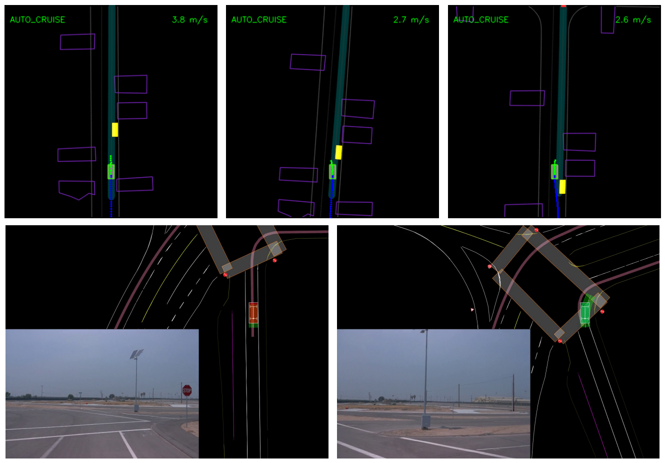


Fig. 1: *ChauffeurNet* driving in simulation (top row) and a real vehicle (bottom row). The cyan path depicts the input route, green box is the agent, blue dots are the agent's past positions and green dots are the predicted future positions which are used by the controller to drive the agent forward. The top row shows how ChauffeurNet nudges the agent to drive around a parked car (yellow box). The bottom row shows ChauffeurNet driving a real car: braking for the stop-sign (left image) and generating a trajectory to complete a natural right turn instead of just following the route (right image).

I. Introduction

Imitation learning is a promising approach for autonomous driving – not only for learning a policy for the autonomous car itself, but also for learning policies that mimic human driving behavior. Having access to such policies is crucial for both anticipating human behavior onboard, as well as testing the car's performance in simulation, prior to deployment. In this work, we introduce a system that pushes the boundaries of what imitation learning can achieve in the driving domain, up to the point that we can trust the learned policy to drive[1] a real vehicle (Fig. 1).

The key challenge in building a robust imitation learning system for driving has been lingering since the days of ALVINN (Pomerleau [19]): we need to run the system closed-loop, which means errors accumulate and induce a shift from the training distribution. This is a general problem with imitation learning (Ross et al. [21]), and in other domains it can be alleviated by actively exposing demonstrators to new states (Ross et al. [21], Laskey et al. [11]), or by combining imitation with reinforcement learning (Kuefler et al. [9]). Unfortunately, the former is not practical in driving, because we cannot ignore driver commands and drive the car in a different, possibly unsafe way. The latter requires simulating human behavior: producing such a simulator is one of the main goals of imitation learning in the first place.

Our goal was therefore to take the demonstrated data as far as possible, without relying on gathering new labels or rolling out policies in a simulator. We built our system from the ground up with this in mind, starting with the choice of input and output representations. Even though there is a lot of excitement for end-to-end approaches – which typically focus on learning to directly predict raw control outputs such as steering or braking after consuming raw sensor input such as camera or lidar data – we reduce sample complexity by opting for a mid-level representation. We use a perception system that processes raw sensor information and produces our input: a top-down representation of the environment and intended route, where objects such as vehicles are drawn as oriented 2D boxes along with a rendering of the road information and traffic light states. We present this mid-level input to a

†Work done while at Google Brain & Waymo.
[1]Please see https://sites.google.com/view/learn-to-drive/ for supplemental videos for this paper.

recurrent neural network (RNN), named *ChauffeurNet*, which then outputs a driving trajectory that is consumed by a controller which translates it to steering and acceleration.

Despite recent results that point to error accumulation no longer being an issue with good representations and high capacity models (Laskey et al. [10]), our first finding was that not even 30 million examples were sufficient for pure imitation learning. This is in spite of our mid-level input and output representations that remove the burden of perception and control. As an example, we found that this model would get stuck or collide with another vehicle parked on the side of a narrow street, when a nudge-and-pass behavior was viable.

The key to our approach for alleviating error accumulation is to augment our demonstrations by *synthesizing* perturbations to the trajectories, thereby exposing the learner to data that is immediately outside the demonstrations. The mid-level input representation is crucial for leveraging perturbations: even though perturbations have been applied at the image level before (Bojarski et al. [1, 2]) to correct steering errors, they fail to control acceleration or inform the learner of what to do near collisions. In contrast, we leverage the mid-level input and output representations to perturb the actual trajectory, and augment the imitation loss with auxiliary losses that penalize collisions, being off-road, etc.

We evaluate our system, as well as the relative importance of both loss augmentation and data augmentation, first in simulation. We then show how our final model successfully drives a car in the real world and is able to negotiate situations involving other agents, turns, stop signs, and traffic lights. Finally, it is important to note that there are highly interactive situations, such as merging, which go beyond the capabilities of purely offline-collected data. Our system is a stepping stone towards better simulation and initialization for RL approaches.

II. Related Work

Learned policies for real-world driving. Imitation learning policies for the task of autonomous driving date back to the seminal paper on ALVINN (Pomerleau [19]), which mapped camera and laser range map inputs to *steering* controls. More recently, Bojarski et al. [1, 2] demonstrated autonomous steering using camera input alone, but the real-world driving results were limited to simple tasks like lane and road following only, with a human driver taking over control when a lane change or a turning maneuver was required.

Policies tested in an open-loop setting. End-to-end prediction of both steering and speed/acceleration has been addressed in Codevilla et al. [4], which uses a single camera image along with high-level control commands as input, and by Hecker et al. [8] which uses 360-degree camera images along with desired route planner as inputs. However, the latter only evaluate their method in terms of a mean-squared error between the ground-truth and predicted speed and steering values 0.3 seconds in the future without any closed-loop control. Xu et al. [25] predict discrete or continuous actions from uncalibrated large-scale video data, but again only evaluate their results in an open-loop setting. In this paper, we show that evaluation in a closed-loop setting is crucial to validate the model's ability to deal with drift issues. In addition, none of the above approaches deal with dynamic obstacles or traffic controls.

Intermediate representations. Instead of directly predicting controls, Chen et al. [3] demonstrated a convolutional net that learns a mapping from an input image to an explicit, hand-crafted list of 13 affordance indicators such as the distance to the preceding car. These affordance indicators are then consumed by a hand-crafted rule-based controller that computes the desired acceleration and steering commands to drive a car in highway environments only. Sauer et al. [22] generalize this to a conditional affordance framework for urban environments and demonstrate results within the CARLA simulator (Dosovitskiy et al. [6]). Unlike our representation, the hand crafted features do not generalize to all driving scenarios, such as complex intersections. Using mid-level representations in a spirit similar to our own, Müller et al. [15] train a system in simulation using CARLA by training a driving policy from a scene segmentation network to output high-level control, thereby enabling transfer learning to the real world using a different segmentation network trained on real data. Pan et al. [17] also describes achieving transfer of an agent trained in simulation to the real world using a learned intermediate scene labeling representation. A convolutional network operating on a space-time volume of bird's eye-view representations is also employed by Luo et al. [14], Djuric et al. [5], Lee et al. [12] for tasks like 3D detection, tracking and motion forecasting.

Policies that work in closed-loop settings. To address the problem of drift in closed-loop control with a learned policy, methods typically resort to collecting labels for states that are outside of the demonstration. DAgger (Ross et al. [21]) iteratively rolls out the learner's current policies and asks experts to correct each state, and DART (Laskey et al. [11]) injects noise into the expert's controls at demonstration time to get to states that are outside the expert's policy. Bojarski et al. [1] addressed the drift issue by augmenting the training data with synthetically shifted and rotated camera views as input and the corresponding adjusted steering values as target. Codevilla et al. [4] also adopted a similar setup but found this kind of data augmentation to be not sufficiently robust. Instead, inspired by DART, they inject temporally correlated noise to the control signal during data generation and ask the expert to record the corrective control as the target, which limits their approach to data generated in a simulated environment (and to only 10% of the data due to the expert effort required). In our approach, we apply synthetic perturbations in the pose space directly and do not need expert input to generate the training labels. This enables us to present the network with both a large amount of real driving expert data as well as a large amount of synthetically perturbed data that explores a much more diverse set of drifts (translation, rotation, heading, speed), than what is possible with just the steering perturbation explored by Bojarski et al. [1]. Additionally, perturbation in the pose space enables us to generate synthetic collisions with other objects and the environment, and the corresponding losses help the network to learn to avoid these and generalize better. We have found this aspect of our work to be crucial to its robustness in dealing with dynamic obstacles – something which hasn't been addressed in either of the two approaches before.

Alternatives to imitation learning. Alternative approaches to imitation learning rely on access to a good simulator. These include reinforcement learning (Shalev-Shwartz et al. [23], Liang et al. [13]), hybrids of reinforcement and imitation learning (Kuefler et al. [9], Rhinehart et al. [20]), and motion planning (Paden et al. [16]). In this work, we focus on the ability to drive not only in isolation, but around and in interaction with other (human) agents, for which accurate simulation is particularly challenging. Our work helps improve our ability to simulate other agents via imitation learning.

III. Model Architecture

A. Input Output Representation

We begin by describing our top-down input representation that the network will process to output a drivable trajectory. At any time t, our agent (or vehicle) may be represented in a top-down coordinate system by $\mathbf{p}_t, \theta_t, s_t$, where $\mathbf{p}_t = (x_t, y_t)$ denotes the agent's location or pose, θ_t denotes the heading or orientation, and s_t denotes the speed. The top-down coordinate system is picked such that our agent's pose $\mathbf{p}_0$ at the current time $t = 0$ is always at a fixed location (u_0, v_0) within the image. For data augmentation purposes during training, the orientation of the coordinate system is randomly picked for each training example to be within an angular range of $\theta_0 \pm \Delta$, where θ_0 denotes the heading or orientation of our agent at time $t = 0$. The top-down view is represented by a set of images of size $W \times H$ pixels, at a ground sampling resolution of ϕ meters/pixel. Note that this view of the environment moves with the agent as it moves, so the agent always sees a fixed forward range, $R_{fwd} = (H - v_0)\phi$ of the world – similar to having an agent with sensors that see only up to R_{fwd} meters forward.

As shown in Fig. 2, the input to our model consists of several images of size $W \times H$ pixels rendered into this top-down coordinate system. (a) Roadmap: a color (3-channel) image with a rendering of various map features such as lanes, stop signs, cross-walks, curbs, etc. (b) Speed limit: a single channel image with lane centers colored in proportion to their known speed limit. (c) Traffic lights: a temporal sequence of grayscale images where each frame of the sequence represents the known state of the traffic lights at each past timestep. Within each frame, we color each lane center by a gray level with the brightest level for red lights, intermediate gray level for yellow lights, and a darker level for green or unknown lights[2]. (d) Route: the intended route along which we wish to drive, generated by a router (think of a Google Maps-style route). (e) Current agent box: this shows our agent's full bounding box at the current timestep $t = 0$. (f) Dynamic objects in the environment: a temporal sequence of images

[2]We employ an indexed representation for roadmap and traffic lights channels to reduce the number of input channels, and to allow extensibility of the input representation to express more roadmap features or more traffic light states without changing the model architecture.

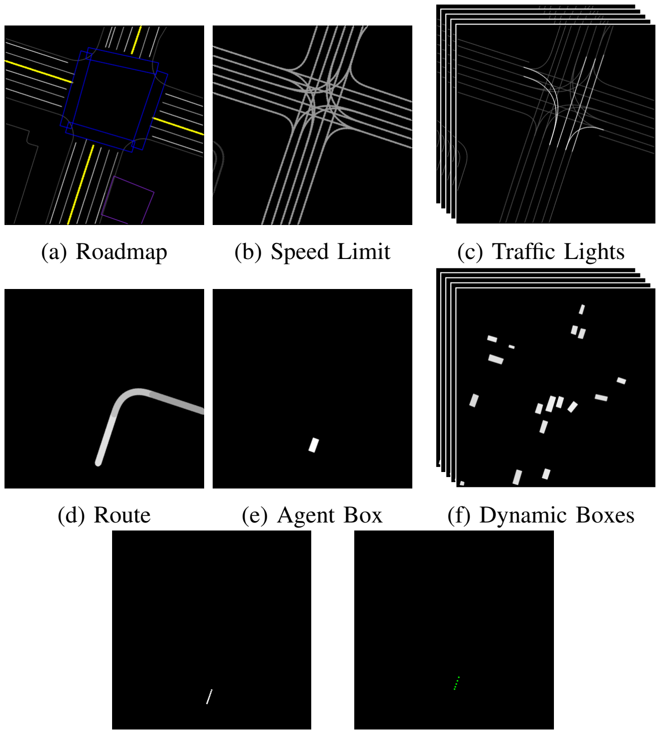

(a) Roadmap (b) Speed Limit (c) Traffic Lights
(d) Route (e) Agent Box (f) Dynamic Boxes
(g) Past Agent Poses (h) Future Agent Poses

Fig. 2: Driving model inputs (a-g) and output (h).

showing all the potential dynamic objects (vehicles, cyclists, pedestrians) rendered as oriented boxes. (g) Past agent poses: the past poses of our agent are rendered into a single grayscale image as a trail of points.

We use a fixed-time sampling of δt to sample any past or future temporal information, such as the traffic light state or dynamic object states in the above inputs. The traffic lights and dynamic objects are sampled over the past T_{scene} seconds, while the past agent poses are sampled over a potentially longer interval of T_{pose} seconds. This simple input representation, particularly the box representation of other dynamic objects, makes it easy to generate input data from simulation or create it from real-sensor logs using a standard perception system that detects and tracks objects. This enables testing and validation of models in closed-loop simulations before running them on a real car. This also allows the same model to be improved using simulated data to adequately explore rare situations such as collisions for which real-world data might be difficult to obtain. Using a top-down 2D view also means efficient convolutional inputs, and allows flexibility to represent metadata and spatial relationships in a human-readable format. Papers on testing frameworks such as Tian et al. [24], Pei et al. [18] show the brittleness of using raw sensor data (such as camera images or lidar point clouds) for learning to drive, and reinforce the approach of using an intermediate input representation.

If I denotes the set of all the inputs enumerated above, then the *ChauffeurNet* model recurrently predicts future poses of our agent conditioned on these input images I as shown by

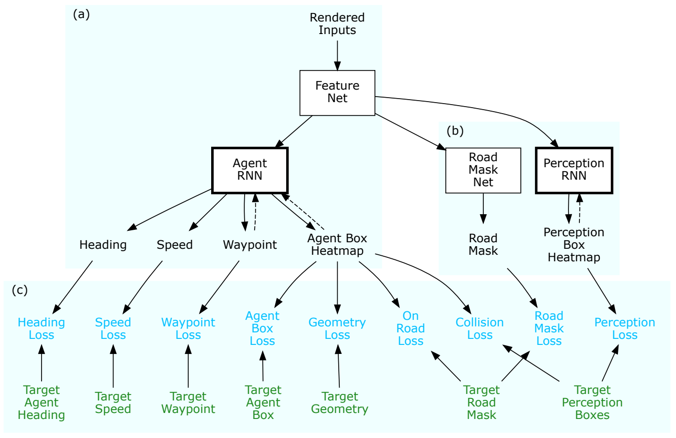


Fig. 3: Training the driving model. (a) The core *ChauffeurNet* model with a FeatureNet and an AgentRNN, (b) Co-trained road mask prediction net and PerceptionRNN, and (c) Training losses are shown in blue, and the green labels depict the ground-truth data. The dashed arrows represent the recurrent feedback of predictions from one iteration to the next.

the green dots in Fig. 2(h).

$$\mathbf{p}_{t+\delta t} = \text{ChauffeurNet}(I, \mathbf{p}_t) \tag{1}$$

In Eq. (1), current pose $\mathbf{p}_0$ is a known part of the input, and then the ChauffeurNet performs N iterations and outputs a future trajectory$\{\mathbf{p}_{\delta t}, \mathbf{p}_{2\delta t}, ..., \mathbf{p}_{N\delta t}\}$ along with other properties such as future speeds. This trajectory can be fed to a controls optimizer that computes detailed driving control (such as steering and braking commands) within the specific constraints imposed by the dynamics of the vehicle to be driven. Different types of vehicles may possibly utilize different control outputs to achieve the same driving trajectory, which argues against training a network to directly output low-level steering and acceleration control. Note, however, that having intermediate representations like ours does not preclude end-to-end optimization from sensors to controls.

B. Model Design

Broadly, the driving model is composed of several parts as shown in Fig. 3. The main *ChauffeurNet* model shown in part (a) of the figure consists of a convolutional feature network (*FeatureNet*) that consumes the input data to create a digested contextual feature representation that is shared by the other networks. These features are consumed by a recurrent agent network (*AgentRNN*) that iteratively predicts successive points in the driving trajectory. Each point at time t in the trajectory is characterized by its location $\mathbf{p}_t = (x_t, y_t)$, heading θ_t and speed s_t. The *AgentRNN* also predicts the bounding box of the vehicle as a spatial heatmap at each future timestep. In part (b) of the figure, we see that two other networks are co-trained using the same feature representation as an input. The Road Mask Network predicts the drivable areas of the field of view (on-road vs. off-road), while the recurrent perception network (*PerceptionRNN*) iteratively predicts a spatial heatmap for each timestep showing the future location of every other agent in

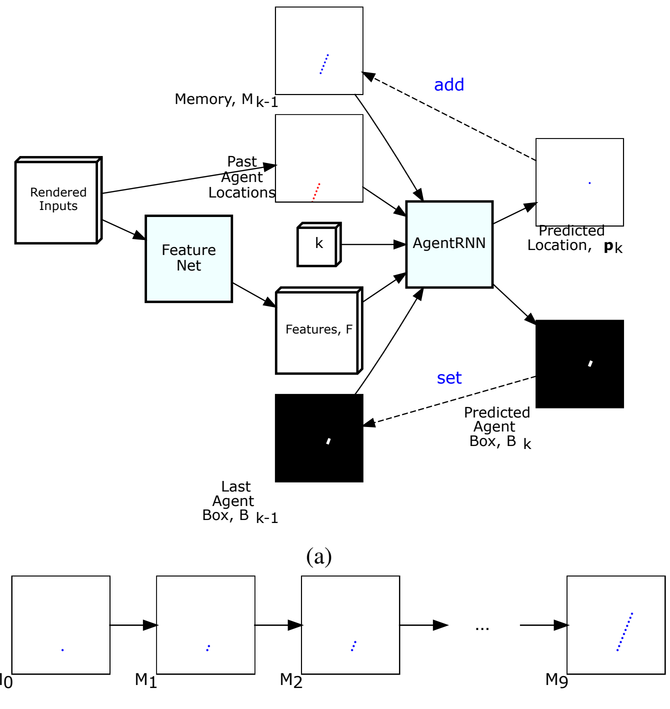


Fig. 4: (a) Schematic of ChauffeurNet. (b) Memory updates over multiple iterations.

the scene. We believe that doing well on these additional tasks using the same shared features as the main task improves generalization on the main task. Fig. 3(c) shows the various losses used in training the model, which we will discuss in detail below.

Fig. 4 illustrates the ChauffeurNet model in more detail. The rendered inputs shown in Fig. 2 are fed to a large-receptive field convolutional *FeatureNet* with skip connections, which outputs features F that capture the environmental context and the intent. These features are fed to the *AgentRNN* which predicts the next point $\mathbf{p}_k$ on the driving trajectory, and the agent bounding box heatmap B_k, conditioned on the features F from the FeatureNet, the iteration number $k \in \{1, \ldots, N\}$, the memory M_{k-1} of past predictions from the *AgentRNN*, and the agent bounding box heatmap B_{k-1} predicted in the previous iteration.

$$\mathbf{p}_k, B_k = \text{AgentRNN}(k, F, M_{k-1}, B_{k-1}) \tag{2}$$

The memory M_k is an additive memory consisting of a single channel image. At iteration k of the *AgentRNN*, the memory is incremented by 1 at the location $\mathbf{p}_k$ predicted by the *AgentRNN*, and this memory is then fed to the next iteration. The *AgentRNN* outputs a heatmap image over the next pose of the agent, and we use the arg-max operation to obtain the coarse pose prediction $\mathbf{p}_k$ from this heatmap. The *AgentRNN* then employs a shallow convolutional meta-prediction network with a fully-connected layer that predicts a sub-pixel refinement of the pose $\delta\mathbf{p}_k$ and also estimates the heading θ_k and the speed s_k. Note that the *AgentRNN* is unrolled at training time for a fixed number of iterations, and the losses described below are summed together over the unrolled iterations. This is possible because of the non-traditional RNN design where we employ an explicitly crafted memory model instead of a learned memory. For visual reference, see site: section G where we show various predictions and losses for a single example. In the next section, we show how to train the model above to imitate the expert.

IV. Imitating the Expert

A. Imitation Losses

1) Agent Position, Heading and Box Prediction: The *AgentRNN* produces three outputs at each iteration k: a probability distribution $P_k(x, y)$ over the spatial coordinates of the predicted waypoint obtained after a spatial *softmax*, a heatmap of the predicted agent box at that timestep $B_k(x, y)$ obtained after a per-pixel *sigmoid* activation that represents the probability that the agent occupies a particular pixel, and a regressed box heading output θ_k. Given ground-truth data for the above predicted quantities, we can define the corresponding losses for each iteration as:

$$\mathcal{L}_p = \mathcal{H}(P_k, P_k^{gt}) \tag{3}$$

$$\mathcal{L}_B = \frac{1}{WH} \sum_x \sum_y \mathcal{H}(B_k(x, y), B_k^{gt}(x, y)) \tag{4}$$

$$\mathcal{L}_\theta = \left\| \theta_k - \theta_k^{gt} \right\|_1 \tag{5}$$

where the superscript *gt* denotes the corresponding ground-truth values, and $\mathcal{H}(a, b)$ is the cross-entropy function. Note that P_k^{gt} is a binary image with only the pixel at the ground-truth target coordinate $\lfloor \mathbf{p}_k^{gt} \rfloor$ set to one.

2) Agent Meta Prediction: The meta prediction network performs regression on the features to generate a sub-pixel refinement $\delta\mathbf{p}_k$ of the coarse waypoint prediction as well as a speed estimate s_k at each iteration. We employ L_1 loss for both of these outputs:

$$\mathcal{L}_{p-subpixel} = \left\| \delta\mathbf{p}_k - \delta\mathbf{p}_k^{gt} \right\|_1 \tag{6}$$

$$\mathcal{L}_{speed} = \left\| s_k - s_k^{gt} \right\|_1 \tag{7}$$

where $\delta\mathbf{p}_k^{gt} = \mathbf{p}_k^{gt} - \lfloor \mathbf{p}_k^{gt} \rfloor$ is the fractional part of the ground-truth pose coordinates.

B. Past Motion Dropout

During training, the model is provided the past motion history as one of the inputs (Fig. 2(g)). Since the past motion history during training is from an expert demonstration, the net can learn to "cheat" by just extrapolating from the past rather than finding the underlying causes of the behavior. During closed-loop inference, this breaks down because the past history is from the net's own past predictions. For example, such a trained net may learn to only stop for a stop sign if it sees a deceleration in the past history, and will therefore never stop for a stop sign during closed-loop inference. To address this, we introduce a dropout on the past pose history, where for 50% of the examples, we keep only the current position (u_0, v_0) of the agent in the past agent poses channel of the input data. This forces the net to look at other cues in the environment to explain the future motion profile.

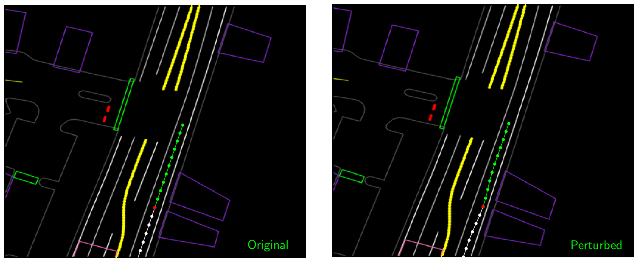


Fig. 5: Trajectory Perturbation: The current agent location (red point) in the original logged example is perturbed to generate the perturbed training example.

V. Beyond pure Imitation

In this section, we go beyond vanilla cloning of the expert's demonstrations in order to teach the model to arrest drift and avoid bad behavior such as collisions and off-road driving by synthesizing variations of the expert's behavior.

A. Synthesizing Perturbations

Running the model as a part of a closed-loop system over time can cause the input data to deviate from the training distribution. To prevent this, we train the model by adding some examples with realistic perturbations to the agent trajectories. The start and end of a trajectory are kept constant, while a perturbation is applied around the midpoint and smoothed across the other points. Quantitatively, we jitter the midpoint pose of the agent uniformly at random in the range $[-0.5, 0.5]$ meters in both axes, and perturb the heading by $[-\pi/3, \pi/3]$ radians. We then fit a smooth trajectory to the perturbed point and the original start and end points. As shown in Fig. 5, such training examples bring the car back to its original trajectory after a perturbation. We filter out some perturbed trajectories that are impractical by thresholding on maximum curvature. But we do allow the perturbed trajectories to collide with other agents or drive off-road, because the network can then experience and avoid such behaviors even though real examples of these cases are not present in the training data. In training, we give perturbed examples a weight of $1/10$ relative to the real examples, to avoid learning a propensity for perturbed driving.

B. Beyond the Imitation Loss

1) Collision Loss: Since our training data does not have any real collisions, the idea of avoiding collisions is implicit and will not generalize well. To alleviate this issue, we add a specialized loss that directly measures the overlap of the predicted agent box B_k with the ground-truth boxes of all the scene objects at each timestep.

$$\mathcal{L}_{collision} = \frac{1}{WH} \sum_x \sum_y B_k(x, y) \, . \, Obj_k^{gt}(x, y) \tag{8}$$

where B_k is the likelihood map for the output agent box prediction, and Obj_k^{gt} is a binary mask with ones at all

T_{scene}	T_{pose}	δt	N	Δ
1.0 s	8.0 s	0.2s	10	25°
W	H	u_0	v_0	ϕ
400 px	400 px	200 px	320 px	0.2 m/px

TABLE I: Parameter values for the experiments in this paper.

Rendering	FeatureNet	AgentRNN (N=10)	PerceptionRNN (N=10)	**Overall**
8 ms	6.5 ms	145 ms	35 ms	**160 ms**

TABLE II: Run-time performance on a NVIDIA P100 GPU.

pixels occupied by other dynamic objects (other vehicles, pedestrians, etc.) in the scene at timestep k. At any time during training, if the model makes a poor prediction that leads to a collision, the overlap loss would influence the gradients to correct the mistake. However, this loss would be effective only during the initial training rounds when the model hasn't learned to predict close to the ground-truth locations due to the absence of real collisions in the ground truth data. This issue is alleviated by the addition of trajectory perturbation data, where artificial collisions within those examples allow this loss to be effective throughout training without the need for online exploration like in reinforcement learning settings.

2) On Road Loss: Trajectory perturbations also create synthetic cases where the car veers off the road or climbs a curb or median because of the perturbation. To train the network to avoid hitting such hard road edges, we add a specialized loss that measures overlap of the predicted agent box B_k in each timestep with a binary mask $Road^{gt}$ denoting the road and non-road regions within the field-of-view.

$$\mathcal{L}_{onroad} = \frac{1}{WH}\sum_x\sum_y B_k(x,y)\ .\ (1 - Road^{gt}(x,y)) \quad (9)$$

3) Geometry Loss: We would like to explicitly constrain the agent to follow the target geometry independent of the speed profile. We model this target geometry by fitting a smooth curve to the target waypoints and rendering this curve as a binary image in the top-down coordinate system. The thickness of this curve is set to be equal to the width of the agent. We express this loss similar to the collision loss by measuring the overlap of the predicted agent box with the binary target geometry image $Geom^{gt}$. Any portion of the box that does not overlap with the target geometry curve is added as a penalty to the loss function.

$$\mathcal{L}_{geom} = \frac{1}{WH}\sum_x\sum_y B_k(x,y)\ .\ (1 - Geom^{gt}(x,y)) \quad (10)$$

4) Auxiliary losses: Similar to our own agent's trajectory, the motion of other agents may also be predicted by a recurrent network. Correspondingly, we add a recurrent perception network *PerceptionRNN* that uses as input the shared features F created by the *FeatureNet* and its own predictions Obj_{k-1} from the previous iteration, and predicts a heatmap Obj_k at each iteration. $Obj_k(x,y)$ denotes the probability that location (x,y) is occupied by a dynamic object at time k. For iteration $k=0$, the PerceptionRNN is fed the ground truth objects at the current time.

$$\mathcal{L}_{objects} = \frac{1}{WH}\sum_x\sum_y \mathcal{H}(Obj_k(x,y), Obj_k^{gt}(x,y)) \quad (11)$$

Co-training a *PerceptionRNN* to predict the future of other agents by sharing the same feature representation F used by the *PerceptionRNN* is likely to induce the feature network to learn better features that are suited to both tasks. Several videos of predicted trajectories from *PerceptionRNN* on logged data are shown on the supplemental site: section F.

We also co-train to predict a binary road/non-road mask by adding a small network of convolutional layers to the output of the feature net F. We add a cross-entropy loss to the predicted road mask output $Road(x,y)$ which compares it to the ground-truth road mask $Road^{gt}$.

$$\mathcal{L}_{road} = \frac{1}{WH}\sum_x\sum_y \mathcal{H}(Road(x,y), Road^{gt}(x,y)) \quad (12)$$

Model	**Description**	w_{imit}	w_{env}
$\mathcal{M}_0$	Imitation with Past Dropout	1.0	0.0
$\mathcal{M}_1$	$\mathcal{M}_0$ + Traj Perturbation	1.0	0.0
$\mathcal{M}_2$	$\mathcal{M}_1$ + Environment Losses	1.0	1.0
$\mathcal{M}_3$	$\mathcal{M}_2$ with less imitation	0.5	1.0
$\mathcal{M}_4$	$\mathcal{M}_2$ with Imitation Dropout	*Dropout prob = 0.5 (see V-C).*	

TABLE III: Model configuration for the model ablation tests.

C. Imitation dropout

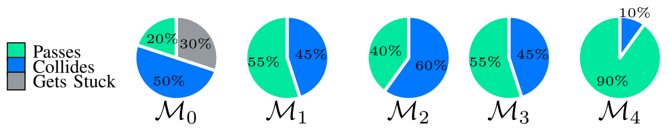
Overall, our losses may be grouped into two sub-groups: the imitation losses $\mathcal{L}_{imit} = \{\mathcal{L}_p, \mathcal{L}_B, \mathcal{L}_\theta, \mathcal{L}_{p-subpixel}, \mathcal{L}_{speed}\}$ and the environment losses $\mathcal{L}_{env} = \{\mathcal{L}_{collision}, \mathcal{L}_{onroad}, \mathcal{L}_{geom}, \mathcal{L}_{objects}, \mathcal{L}_{road}\}$. The imitation losses cause the model to imitate the expert's demonstrations, while the environment losses discourage undesirable behavior such as collisions. To further increase the effectiveness of the environment losses, we experimented with randomly dropping out the imitation losses for a random subset of training examples. We refer to this as "imitation dropout". In the experiments, we show that imitation dropout yields a better driving model than simply under-weighting the imitation losses. During imitation dropout, the weight on the imitation losses w_{imit} is randomly chosen to be either 0 or 1 with a certain probability for each training example. The overall loss is given by:

$$\mathcal{L} = w_{imit}\sum_{\ell\in\mathcal{L}_{imit}}\ell + w_{env}\sum_{\ell\in\mathcal{L}_{env}}\ell \quad (13)$$

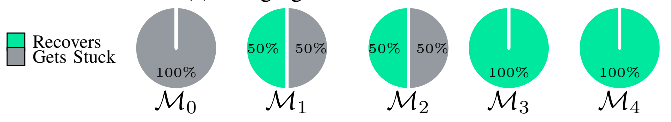
VI. Experiments

A. Data

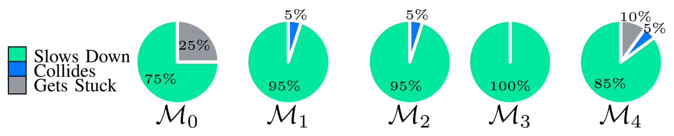
The training data to train our model was obtained by randomly sampling segments of real-world expert driving and removing segments where the car was stationary for long periods of time. Our input field of view is $80m \times 80m$ $(W\phi = 80)$ and with the agent positioned at (u_0, v_0), we get an effective forward sensing range of $R_{fwd} = 64m$. Therefore, for the experiments in this work we also removed any segments of highway driving given the longer sensing



(a) Nudging for a Parked Car



(b) Trajectory Perturbation



(c) Slowing for a Slow Car

Fig. 6: Model ablation test results on three scenario types. For videos, please see [supplemental site: section C].

range requirement that entails. Our dataset contains approximately 26 million examples which amount to about 60 days of continuous driving. As discussed in Section III, the vertical-axis of the top-down coordinate system for each training example is randomly oriented within a range of $\Delta = \pm 25°$ of our agent's current heading, in order to avoid a bias for driving along the vertical axis.The rendering orientation is set to the agent heading ($\Delta = 0$) during inference. Data about the prior map of the environment (roadmap) and the speed-limits along the lanes is collected apriori. For the dynamic scene entities like objects and traffic-lights, we employ a separate perception system based on laser and camera data similar to existing works in the literature (Yang et al. [26], Fairfield and Urmson [7]). Table I lists the parameter values used for all the experiments in this paper. The model runs on a NVIDIA Tesla P100 GPU in 160ms with the detailed breakdown in Table II.

B. Models

We train and test not only our final model, but a sequence of models that introduce the ingredients we describe one by one on top of behavior cloning. We start with $\mathcal{M}_0$, which does behavior cloning with past motion dropout to prevent using the history to cheat. $\mathcal{M}_1$ adds perturbations without modifying the losses. $\mathcal{M}_2$ further adds our environment losses $\mathcal{L}_{env}$ in Section V-B. $\mathcal{M}_3$ and $\mathcal{M}_4$ address the fact that we do not want to imitate bad behavior – $\mathcal{M}_3$ is a baseline approach, where we simply decrease the weight on the imitation loss, while $\mathcal{M}_4$ uses our imitation dropout approach with a dropout probability of 0.5. Table III lists the configuration for each of these models.

C. Closed Loop Evaluation

To evaluate our learned model on a specific scenario, we replay the segment through the simulation until a buffer period of $\max(T_{pose}, T_{scene})$ has passed. This allows us to generate the first rendered snapshot of the model input using all the replayed messages until now. The model is evaluated on this input, and the fitted controls are passed to the vehicle simulator that emulates the dynamics of the vehicle thus moving the simulated agent to its next pose. At this point, the simulated pose might be different from the logged pose, but our input representation allows us to correctly render the new input for the model relative to the new pose. This process is repeated until the end of the segment, and we evaluate scenario specific metrics like stopping for a stop-sign during the simulation. Since the model is being used to drive the agent forward, this is a *closed-loop* evaluation setup.

1) Model Ablation Tests: Here, we present results from experiments using the various models in the closed-loop simulation setup. We first evaluated all the models on simple situations such as stopping for stop-signs and red traffic lights, and lane following along straight and curved roads by creating 20 scenarios for each situation, and found that *all the models worked well in these simple cases*. Therefore, we will focus below on specific complex situations that highlight the differences between these models.

a) Nudging around a parked car: To set up this scenario, we place the agent at an arbitrary distance from a stop-sign on an undivided two-way street and then place a parked vehicle on the right shoulder between the the agent and the stop-sign. We pick 4 separate locations with both straight and curved roads then vary the starting speed of the agent between 5 different values to create a total of 20 scenarios. We then observe if the agent would stop and get stuck behind, collide with the parked car, or correctly pass around the parked car, and report the aggregate performance in Fig. 6a. We find that other than $\mathcal{M}_4$, all other models cause the agent to collide with the parked vehicle about half the time. The baseline $\mathcal{M}_0$ model can also get stuck behind the parked vehicle in some of the scenarios. Model $\mathcal{M}_4$ nudges around the parked vehicle and then brings the agent back to the lane center. This can be attributed to the model's ability to learn to avoid collisions and nudge around objects because of training with the collision loss and trajectory perturbation. Comparing models $\mathcal{M}_3$ & $\mathcal{M}_4$, it is apparent that "imitation dropout" was more effective at learning the right behavior than only re-weighting the imitation losses. Note that by varying the agent starting speed, we create scenarios where the agent approaches the parked car at a very high relative speed and thus does not have enough time to nudge around the car given the dynamic constraints. A 10% collision rate for $\mathcal{M}_4$ is thus not a measure of the absolute performance of the model since we do not have a perfect driver which could have performed well at all the scenarios here. But in relative terms, this model performs the best.

b) Recovering from a trajectory perturbation: We place the agent approaching a curved road and vary the starting position and the starting speed of the agent to generate a total of 20 scenario variations. Each variation puts the agent at a different amount of offset from the lane center with a different heading error relative to the lane. We then measure how well the various models are at recovering from the lane departure. Fig. 6b presents the results aggregated across these scenarios and shows the contrast between the baseline model

$\mathcal{M}_0$ which is not able to recover in any of the situations and the models $\mathcal{M}_3$ & $\mathcal{M}_4$ which handle all deviations well. All models trained with perturbation data are able to handle 50% of the scenarios which have a lower starting speed. At a higher starting speed, $\mathcal{M}_3$ & $\mathcal{M}_4$ do better than $\mathcal{M}_1$ & $\mathcal{M}_2$ because they place a higher emphasis on the imagination losses.

c) Slowing down for a slow car: To set up this scenario, we place the agent on a straight road at varying initial speeds and place another car ahead with a varying but slower constant speed, generating a total of 20 scenario variations, to evaluate the ability to slow for and then follow the car ahead. From Fig. 6c, we see that some models slow down to zero speed and get stuck. For the variation with the largest relative speed, there isn't enough time for most models to stop the agent in time, thus leading to a collision. For these cases, model $\mathcal{M}_3$ which uses imitation loss re-weighting works better than the model $\mathcal{M}_4$ which uses imitation dropout. $\mathcal{M}_4$ has trouble in two situations due to being over aggressive in trying to maneuver around the slow car and then grazes the left edge of the road. This happens in the two extreme variations where the relative speed between the two cars is the highest.

2) Input Ablation Tests: With input ablation tests, we want to test the final $\mathcal{M}_4$ model's ability to identify the correct causal factors behind specific behaviors, by testing the model's behavior in the presence or absence of the correct causal factor while holding other conditions constant. In simulation, we have evaluated our model on 20 scenarios with and without stop-signs rendered, and 20 scenarios with and without other vehicles in the scene rendered. The model exhibits the correct behavior in all scenarios, thus confirming that it has learned to respond to the correct features for a stop-sign and a stopped vehicle. Please see the site: section B for examples.

3) Logged Data Simulated Driving: For this evaluation, we take logs from our real-driving test data (separate from training data), and use our trained network to drive the car using the vehicle simulator keeping dynamic objects, traffic-light states etc. the same as in the logs. Some example videos are shown on the site: section E and they illustrate the ability of the model in dealing with multiple dynamic objects and road controls.

4) Real World Driving: We have also evaluated this model on our self-driving car by replacing the existing planner module with the learned model $\mathcal{M}_4$ and have replicated the driving behaviors observed in simulation. The videos of several of these runs are available on the supplemental site: section D and they illustrate not only the smoothness of the network's driving ability, but also its ability to deal with stop-signs and turns and to drive for long durations in full closed-loop control without deviating from the trajectory.

D. Open Loop Evaluation

In an open-loop evaluation, we take test examples of expert driving data and for each example, compute the L_2 distance error between the predicted and ground-truth waypoints. Unlike the closed-loop setting, the predictions are not used to drive the agent forward and thus the network never sees its own predictions as input. Fig. 7 shows the L_2 distance metric

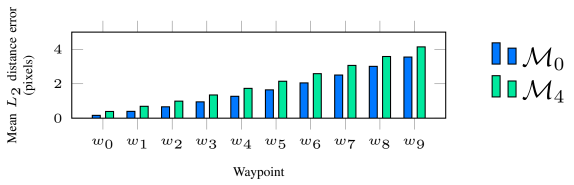


Fig. 7: Prediction Error for models $\mathcal{M}_0$ and $\mathcal{M}_4$ on unperturbed evaluation data (1 pixel = 0.2m).

in this open-loop evaluation setting for models $\mathcal{M}_0$ and $\mathcal{M}_4$ on a test set of 10,000 examples. These results show that model $\mathcal{M}_0$ makes fewer errors than the full model $\mathcal{M}_4$, but we know from closed-loop testing that $\mathcal{M}_4$ is a far better driver than $\mathcal{M}_0$. This shows how open-loop evaluations such as Fig. 7 are insufficient indicators of real driving performance, and closed-loop evaluations are critical while assessing such driving models.

E. Failure Modes

At our ground resolution of 20 cm/pixel, the agent currently sees 64 m in front and 40 m on the sides and this limits the model's ability to perform merges on T-junctions and turns from a high-speed road. Specific situations like U-turns and cul-de-sacs are also not currently handled, and will require sampling enough training data. The model occasionally gets stuck in some low speed nudging situations. It sometimes outputs turn geometries that make the specific turn infeasible (e.g. large turning radius). We also see some cases where the model gets over aggressive in novel and rare situations for example by trying to pass a slow moving vehicle. We believe that adequate simulated exploration may be needed for highly interactive or rare situations.

VII. Discussion

In this paper, we presented our experience with what it took to get imitation learning to perform well in real-world driving. We found that key to its success is synthesizing interesting situations around the expert's behavior and augmenting appropriate losses that discourage undesirable behavior. This constrained exploration is what allowed us to avoid collisions and off-road driving even though such examples were not explicitly present in the expert's demonstrations. To support it, and to best leverage the expert data, we used mid-level input and output representations which allow easy mixing of real and simulated data and alleviate the burdens of learning perception and control. With these ingredients, we got a model good enough to drive a real car. That said, the model is not yet fully competitive with motion planning approaches but we feel that this is a good step forward for machine learned driving models. There is room for improvement: comparing to end-to-end approaches, and investigating alternatives to imitation dropout are among them. But most importantly, we believe that augmenting the expert demonstrations with a thorough exploration of rare and difficult scenarios in simulation, perhaps within a reinforcement learning framework, will be the key to improving the performance of these models especially for highly interactive scenarios.

Acknowledgments

We would like to thank Andrew Barton-Sweeney for help running the model on the car, Aleksandar Gabrovski for help with the simulation, Anca Dragan and Dragomir Anguelov for reviewing the paper and suggesting several improvements.

References

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[2] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint arXiv:1704.07911*, 2017.

[3] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[4] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[5] Nemanja Djuric, Vladan Radosavljevic, Henggang Cui, Thi Nguyen, Fang-Chieh Chou, Tsung-Han Lin, and Jeff Schneider. Motion prediction of traffic actors for autonomous driving using deep convolutional networks. *arXiv preprint arXiv:1808.05819*, 2018.

[6] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[7] Nathaniel Fairfield and Chris Urmson. Traffic light mapping and detection. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5421–5426. IEEE, 2011.

[8] Simon Hecker, Dengxin Dai, and Luc Van Gool. Learning driving models with a surround-view camera system and a route planner. *arXiv preprint arXiv:1803.10158*, 2018.

[9] Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. Imitating driver behavior with generative adversarial networks. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 204–211. IEEE, 2017.

[10] Michael Laskey, Caleb Chuck, Jonathan Lee, Jeffrey Mahler, Sanjay Krishnan, Kevin Jamieson, Anca Dragan, and Ken Goldberg. Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 358–365. IEEE, 2017.

[11] Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. Dart: Noise injection for robust imitation learning. *arXiv preprint arXiv:1703.09327*, 2017.

[12] Donghan Lee, Youngwook Paul Kwon, Sara McMains, and J Karl Hedrick. Convolution neural network-based lane change intention prediction of surrounding vehicles for acc. In *Intelligent Transportation Systems (ITSC), 2017 IEEE 20th International Conference on*, pages 1–6. IEEE, 2017.

[13] Xiaodan Liang, Tairui Wang, Luona Yang, and Eric Xing. Cirl: Controllable imitative reinforcement learning for vision-based self-driving. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 584–599, 2018.

[14] 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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2018.

[15] Matthias Müller, Alexey Dosovitskiy, Bernard Ghanem, and Vladen Koltun. Driving policy transfer via modularity and abstraction. *arXiv preprint arXiv:1804.09364*, 2018.

[16] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on intelligent vehicles*, 1 (1):33–55, 2016.

[17] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

[18] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.

[19] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[20] Nicholas Rhinehart, Rowan McAllister, and Sergey Levine. Deep imitative models for flexible inference, planning, and control. *arXiv preprint arXiv:1810.06544*, 2018.

[21] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635, 2011.

[22] Axel Sauer, Nikolay Savinov, and Andreas Geiger. Conditional affordance learning for driving in urban environments. *arXiv preprint arXiv:1806.06498*, 2018.

[23] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv preprint arXiv:1610.03295*, 2016.

[24] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray.

Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering*, pages 303–314. ACM, 2018.

[25] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2174–2182, 2017.

[26] Bin Yang, Ming Liang, and Raquel Urtasun. Hdnet: Exploiting hd maps for 3d object detection. In *Conference on Robot Learning*, pages 146–155, 2018.