Exploring Imitation Learning for Autonomous Driving with Feedback Synthesizer and Differentiable Rasterization

Jinyun Zhou, Rui Wang, Xu Liu, Yifei Jiang, Shu Jiang, Jiaming Tao, Jinghao Miao, Shiyu Song[1]

Abstract— We present a learning-based planner that aims to robustly drive a vehicle by mimicking human drivers' driving behavior. We leverage a mid-to-mid approach that allows us to manipulate the input to our imitation learning network freely. With that in mind, we propose a novel feedback synthesizer for data augmentation. It allows our agent to gain more driving experience in various previously unseen environments that are likely to encounter, thus improving overall performance. This is in contrast to prior works that rely purely on random synthesizers. Furthermore, rather than completely commit to imitating, we introduce task losses that penalize undesirable behaviors, such as collision, off-road, and so on. Unlike prior works, this is done by introducing a differentiable vehicle rasterizer that directly converts the waypoints output by the network into images. This effectively avoids the usage of heavy-weight ConvLSTM networks, therefore, yields a faster model inference time. About the network architecture, we exploit an attention mechanism that allows the network to reason critical objects in the scene and produce better interpretable attention heatmaps. To further enhance the safety and robustness of the network, we add an optional optimization-based post-processing planner improving the driving comfort. We comprehensively validate our method's effectiveness in different scenarios that are specifically created for evaluating self-driving vehicles. Results demonstrate that our learning-based planner achieves high intelligence and can handle complex situations. Detailed ablation and visualization analysis are included to further demonstrate each of our proposed modules' effectiveness in our method.

I. INTRODUCTION

In recent years, autonomous driving technologies have seen dramatic advances in self-driving vehicles' commercial and experimental operations. Due to complex road topologies across cities and dynamic interaction among vehicles or pedestrians, traditional planning approaches that involve heavy manual tuning are believed to be not cost-effective or scalable [1]. An alternative is imitation learning, a data-driven approach that mimics expert agents' driving behavior and leverages recent advances in supervised learning.

Imitation learning suffers from a long bothering issue, distributional shift [2], [3]. Variation in input state distribution between training and testing sets causes accumulated errors in the learned policy output, which further leads the agent to an unseen environment where the system fails ultimately. The key solution to this compounding problem is to increase the demonstrated data as much as possible. Inspired by [4],

The authors are with Baidu Autonomous Driving Technology Department, {jinyunzhou, ruiwang, liuxu21, jiangyifei, shujiang, taojiaming, miaojinghao, songshiyu}@baidu.com.

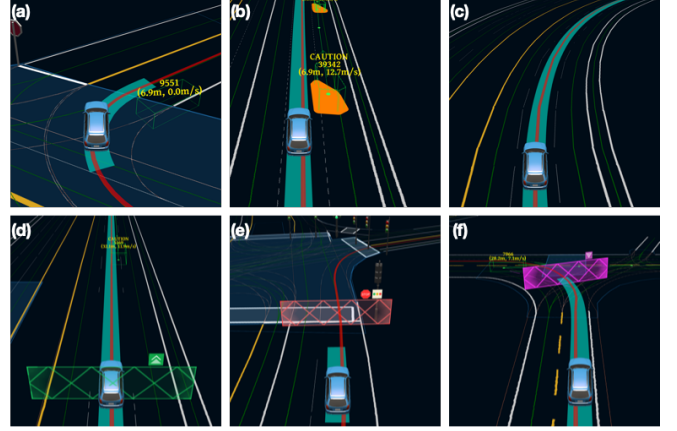[1]Author to whom correspondence should be addressed, E-mail: songshiyu@baidu.com

Fig. 1: Demonstration of the test scenarios that our learning-based planner (M2) successfully passed: (a) passing a static obstacle; (b) overtaking a dynamic obstacle; (c) cruising on a curved lane; (d) following a leading vehicle; (e) stopping for a red traffic light; (f) yielding to a vehicle.

[5], [6], [7], our work adopts a mid-to-mid approach where our system's input is constructed by building a top-down image representation of the environment that incorporates both static and dynamic information from our HD Map and perception system. This gives us large freedom to manipulate the input to an imitation learning network [5]. Therefore, it helps us handily increase the demonstrated data using appropriate data augmentation techniques and neatly avoid technical difficulties in building a high fidelity raw sensor data simulator. Following the philosophy of DAgger [2], we introduce a feedback synthesizer that generates and perturbs on-policy data based on the current policy. Then we train the next policy on the aggregate of collected datasets. The feedback synthesizer addresses the distributional shift issue, thus improving the overall performance shown in Section IV.

Furthermore, pure imitation learning does not explicitly specify important goals and constraints involved in the task, which yields undesirable behaviors inevitably. We introduce task losses to help regulate these unwanted behaviors, yielding better causal inference and scene compliance. Inspired by [8], The task losses are implemented by directly projecting the output trajectories into top-down images using a differentiable vehicle rasterizer. They effectively penalize behaviors, such as obstacle collision, traffic rule violation, and so on, by building losses between rasterized images and specific object masks. Inspired by recent works [9], our output trajectories are produced by a trajectory decoding module that includes an LSTM [10] and a kinematic layer that assures our output trajectories' feasibility. On the whole, these designs help us

avoid using the heavy AgentRNN network in Chauffeurnet [5], which functions similarly to a Convolutional-LSTM network [11]. It is demonstrated that our proposed method achieves a much shorter network inference time comparing to [5], as shown in Section IV-E. Moreover, to further improve the performance, similar to recent works [12], [13], we introduce a spatial attention module in our network design. The heatmap estimated by the attention module provides us a tool for visualizing and reasoning critical objects in the environment and reveals a causal relationship between them and the ego vehicle, as shown in Figure 10.

At last, although a standard vehicle controller can directly use the trajectory waypoints generated by our network to drive the vehicle following them, we propose to add an optional post-processing planner as a gatekeeper which manages to interpret them as high-level decision guidance and composes a new trajectory that offers better comfort. This architecture can be extended in the future, addressing the paramount safety concern caused by the fact that a neural network is typically considered a black box and hard to interpret and analyze.

Our models are trained with 400 hours of human driving data. We evaluated our system using 70 autonomous driving test scenarios (ADS) that are specifically created for evaluating the fundamental driving capabilities of a self-driving vehicle. We show that our learning-based planner (M2) trained via imitation learning achieves 70.0% ADS pass rate and can intelligently handle different challenging scenarios, including overtaking a dynamic vehicle, stopping for a red traffic light, and so on, as shown in Figure 1.

II. RELATED WORKS

1) Imitation Learning: Imitation Learning for motion planning was first introduced in the pioneering work [14] where it directly learns a policy that maps sensor data to steering angle and acceleration. In recent years, there is an extensive literature [15], [16], [17], [18], [19], [3], [12] that follows this end-to-end philosophy. Alternatively, our work adopts a mid-to-mid approach [5], [6] which allows us to augment data handily and go beyond pure imitation by having task-specific losses.

2) Loss and Differentiable Rasterization: Imitation learning for motion planning typically applies a loss between inferred and ground truth trajectories [17], [19]. Therefore, the ideas of avoiding collisions or off-road situations are implicit and don't generalize well. Some prior works [20], [5] propose to introduce task-specific losses penalizing these unwanted behaviors. With that in mind, we introduce task losses in our work and achieve real-time inference based on a differentiable rasterizer. Wang et al. [8] leverage a differentiable rasterizer, and it allows gradients to flow from a discriminator to a generator, enhancing a trajectory prediction network powered by GANs [21]. Different from us, concurrent work [22] tackles a trajectory prediction task for better scene-compliant using similar rasterizers. General-purpose differentiable mesh renderers [23], [24] have also been employed to solve other computer vision tasks.

(a) Rasterized Scene (b) Agent Box (c) Past Agent Poses
(d) Obs Prediction (e) Obs History (f) HD Map
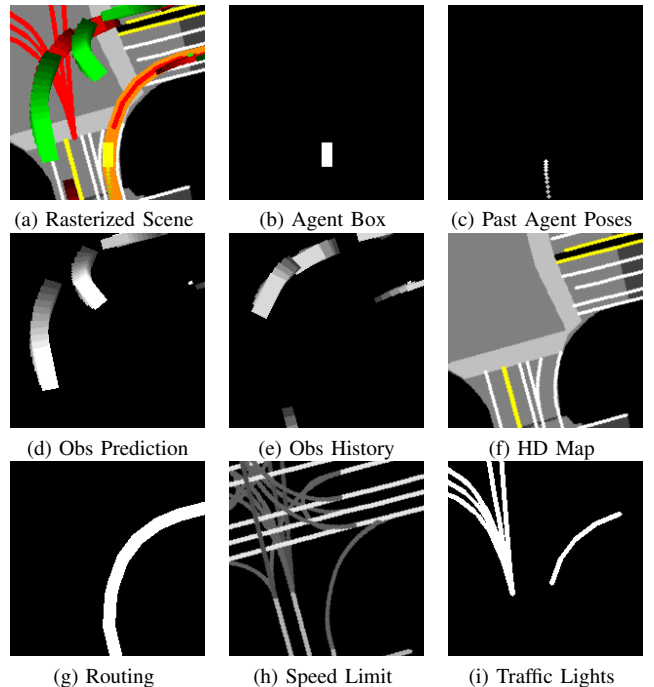(g) Routing (h) Speed Limit (i) Traffic Lights

Fig. 2: Scene rasterization in BEV as multi-channel image inputs for a scene shown in (a): (b) agent rasterized as a box; (c) past agent trajectory rasterized as a sequence of points with fading brightness (The oldest points are darkest); (d) prediction of obstacles rasterized as a sequence of boxes with fading brightness (The furthest boxes are brightest); (e) history of obstacles (The oldest boxes are darkest); (f) a color (3-channel) image rendered with surrounding road structures including lanes, intersections, crosswalks, etc; (g) the intended route rasterized in a constant white color; (h) lanes colored in proportion to known speed limit values; (i) traffic lights affected lanes colored in different grey levels that are corresponding to different states of traffic lights.

3) Attention: The attention mechanism has been used successfully in both computer vision [25], [26], [27] or natural language processing [28] tasks. Some works [29], [30], [31] employ the attention mechanism to improve the model's interpretability by providing spatial attention heatmaps highlighting image areas that the network attends to in their tasks. In this work, we introduce the Atrous Spatial Attentional Bottleneck from [13], providing easily interpretable attention heatmaps while also enhancing the network's performance.

4) Data Augmentation: DAgger [2] and its variants [32], [4], [3] propose to address the distributional shift issue by having more data with states that the agent is likely to encounter. In particular, they sample new states at each iteration based on the actions inferred from the current policy, let expert agents demonstrate the actions they would take under these new states, and train the next policy on the aggregate of collected datasets. They have been explored in the context of autonomous driving in prior works [32], [33], [7]. ChauffeurNet [5] introduces a random synthesizer that augments the demonstration data by synthesizing perturbations to the trajectories. In this work, we explore both ideas and propose a feedback synthesizer improving the overall performance.
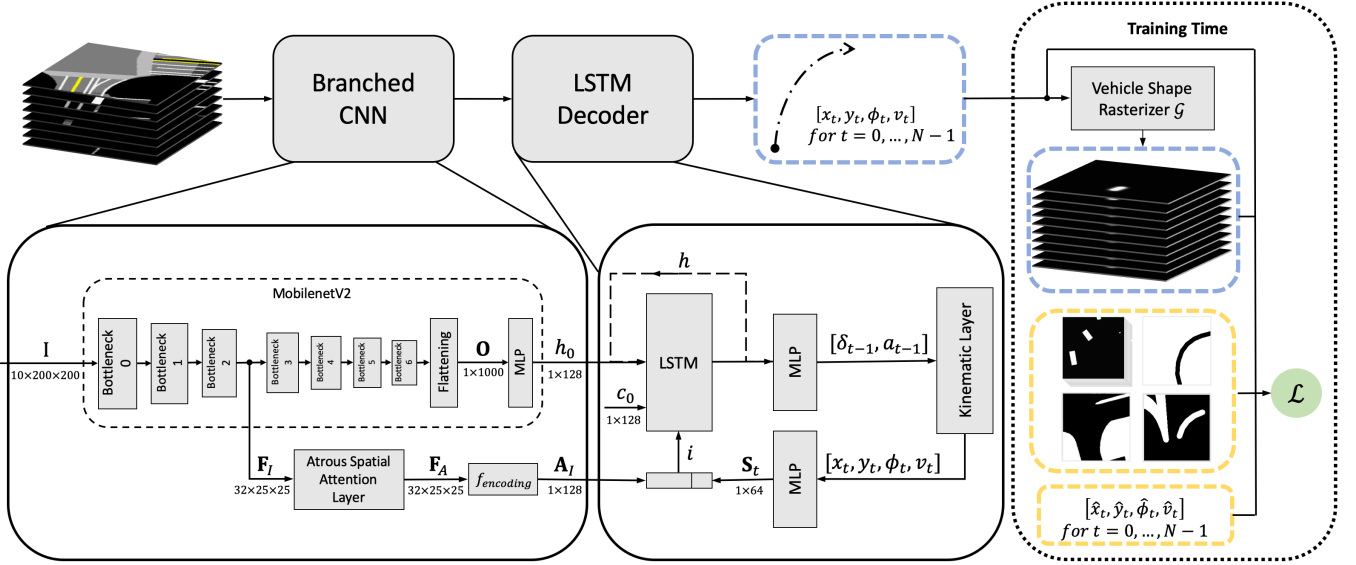
Fig. 3: The illustration of the network structure of the three main modules: (a) a CNN backbone with a branched spatial attention structure; (b) an LSTM trajectory decoder with a kinematical layer; (c) a vehicle shape differentiable rasterizer.

III. METHOD

A. Model Architecture

1) Model Input: We form a bird's eye view (BEV) representation with multiple channels by scene rasterization, similar to those used in both earlier planning [5], [6], [12] or prediction [34], [35], [36] works. The image size is $W \times H$ with ρ meters per pixel in resolution. All channels are ego-centered, which means an agent vehicle positioned at $\mathbf{x}_0 = [x_0, y_0]^T$ is always centered at image coordinate $\mathbf{p}_0 = [i_0, j_0]^T$ within the image and heading upward. The input $\mathcal{I}$ consists of several different channels, as shown in Figure 2. Both prediction or history of our ego-vehicle or other obstacles is sampled with a fixed time interval of Δt.

Besides the multi-channel images, vehicle speed v_0 is also an input, incorporated later by the kinematic motion model into the network (detailed in Section III-A.2).

2) Model Design: The model consists of three parts, a CNN backbone with a branch of spatial attention module that is stemmed from intermediate features of the CNN, an LSTM decoder taking the feature embeddings from the CNN as inputs and outputting planning trajectories in the time domain, a differentiable rasterizer module that is appended to the LSTM decoder during training rasterizing output trajectories into images and finally feeding them to our loss functions. The model architecture is shown in Figure 3.

For the CNN backbone, we use MobileNetV2 [37] for its well-balanced accuracy and inference speed. Then, its output features $\mathbf{F}_h$ are passed through a multilayer perceptron (MLP) layer, producing flattened feature output $\mathbf{h}_0$, which is the initial hidden state of our LSTM decoder. Also, to lighten the computation workload, intermediate features $\mathbf{F}_\mathcal{I}$ from the backbone instead of raw image channels are fed to our spatial attention module. Our attention implementation adopts the Atrous Spatial Attentional Bottleneck from [13], which provides better interpretable attention maps resulting in attentive features $\mathbf{F}_\mathcal{A}$. Following an encoding block

$f_{encoding}$ consisting of a sequence of 3×3 convolutions and average poolings, a vector $\mathbf{A}_\mathcal{I}$ is output by the spatial attention module. It constructs a part of the input $\mathbf{i}$ of the succeeding LSTM decoder. The cell state c_0 is initialized by the Glorot initialization [38]. Then, our LSTM decoder takes these inputs and generates a sequence of future vehicle steering angle and acceleration values (δ_{t-1}, a_{t-1}), where $t = 0, \dots, N-1$. Similar to prior works in the fields of vehicle control [39] or trajectory prediction [9], we add a kinematic model that is differentiable and acts as a network layer in the LSTM. Given the latest states $(x_{t-1}, y_{t-1}, \phi_{t-1}, v_{t-1})$, it converts (δ_{t-1}, a_{t-1}) to corresponding vehicle horizontal coordinates, heading angles, and velocities (x_t, y_t, ϕ_t, v_t), yielding kinematically feasible planning trajectories:

$$
\begin{aligned}
x_t &= v_{t-1} \cos(\phi_{t-1})\Delta t + x_{t-1}, \\
y_t &= v_{t-1} \sin(\phi_{t-1})\Delta t + y_{t-1}, \\
\phi_t &= v_{t-1} \frac{\tan(\delta_{t-1})}{L}\Delta t + \phi_{t-1}, \\
v_t &= a_{t-1}\Delta t + v_{t-1},
\end{aligned}
\tag{1}
$$

where L denotes the vehicle wheelbase length. The output vehicle waypoints and states (x_t, y_t, ϕ_t, v_t) are also passed through an MLP layer. Then the resulted embedding $\mathbf{S}_t$ are concatenated with aforementioned $\mathbf{A}_\mathcal{I}$ yielding the LSTM input $\mathbf{i}$ for the next LSTM iteration.

3) Differentiable Rasterizer: During training time, the output trajectory points are rasterized into N images. As shown in Figure 4, unlike [22], we represent our ego vehicle using three 2D Gaussian kernels instead of one, which better sketch its shape. Given the vehicle waypoints $\mathbf{s}_t = [x_t, y_t, \phi_t, v_t]^T$, vehicle length l, and width w, our rasterization function $\mathcal{G}$ rasterizes images as:

$$
\mathcal{G}_{i,j}(\mathbf{s}_t) = \max_{k=1,2,3}(\mathcal{N}(\mu^{(k)}, \mathbf{\Sigma}^{(k)})),
\tag{2}
$$

(a) Vehicle shape   (b) Rasterized shape   (c) Gaussian kernels
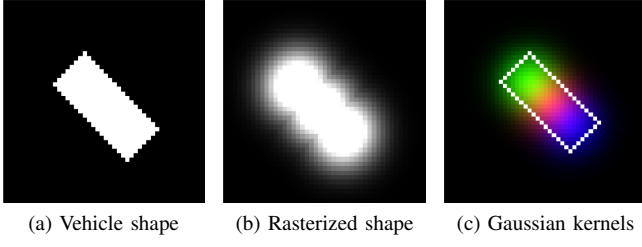
Fig. 4: Visualization of rasterized images: (a) vehicle shape that we aim to rasterize; (b) the rasterized images by our differentiable rasterizer; (c) visualization of the Gaussian kernels used by the rasterizer in different colors.

where each cell (i, j) in $\mathcal{G}$ is denoted as $\mathcal{G}_{i,j}$. Let $\mathbf{x}_t^{(k)} = [x_t^{(k)}, y_t^{(k)}]^T$ denote the center of the k-th Gaussian kernel of our vehicle representation. Then,

$$
\mu^{(k)} = \frac{1}{\rho}(\mathbf{x}_t^{(k)} - \mathbf{x}_0) + \mathbf{p}_0,
$$
$$
\mathbf{\Sigma}^{(k)} = \mathbf{R}(\phi_t)^T \mathrm{diag}(\sigma_l, \sigma_w)\mathbf{R}(\phi_t),
$$
(3)

where $(\sigma_l, \sigma_w) = (\frac{1}{3}\alpha l, \alpha w)$, α is a fixed positive scaling factor, and $\mathbf{R}(\phi)$ represents a rotation matrix constructed from the heading ϕ. Note that the Gaussian distributions are not truncated by the vehicle box-like shape allowing the model to learn how human drivers keep a safe distance to obstacles or road boundaries. Our experiments also demonstrate that this design choice helps us achieve better performance, as shown in Section IV-F.3.

4) Loss: Benefiting from the LSTM decoder and differentiable rasterizer, our trajectory imitation loss is simple and can be expressed in analytic form as:

$$
\mathcal{L}_{\mathrm{imit}} = \sum_{t=0}^{N-1} \lambda \|\mathbf{s}_t - \hat{\mathbf{s}}_t\|_2,
$$
(4)

where $\|\cdot\|_2$ is the L2 norm, λ are the weights, and $\hat{\mathbf{s}}_t$ denotes the corresponding ground truth vehicle waypoints.

Besides the imitation loss, four task losses are introduced to prevent our vehicle from undesirable behaviors, such as obstacle collision, off-route, off-road, or traffic signal violation. $\mathcal{T}^{\mathrm{obs}}$, $\mathcal{T}^{\mathrm{route}}$, $\mathcal{T}^{\mathrm{road}}$, and $\mathcal{T}^{\mathrm{signal}}$ are corresponding binary masks, as shown in Figure 5. We assign ones to all the cells that our vehicle should avoid in our binary masks. The following losses are included:

$$
\mathcal{L}_{\mathrm{obs}} = \sum_{t=0}^{N-1} \frac{1}{WH} \sum_i \sum_j \mathcal{G}_{i,j} \mathcal{T}_{i,j}^{\mathrm{obs}},
$$
$$
\mathcal{L}_{\mathrm{route}} = \sum_{t=0}^{N-1} \frac{1}{WH} \sum_i \sum_j \mathcal{G}_{i,j} \mathcal{T}_{i,j}^{\mathrm{route}},
$$
$$
\mathcal{L}_{\mathrm{road}} = \sum_{t=0}^{N-1} \frac{1}{WH} \sum_i \sum_j \mathcal{G}_{i,j} \mathcal{T}_{i,j}^{\mathrm{road}},
$$
(5)
$$
\mathcal{L}_{\mathrm{signal}} = \sum_{t=0}^{N-1} \frac{1}{WH} \sum_i \sum_j \mathcal{G}_{i,j} \mathcal{T}_{i,j}^{\mathrm{signal}}.
$$

Overall, our total loss is given by:

$$
\mathcal{L} = \mathcal{L}_{imit} + \lambda_{\mathrm{task}}(\mathcal{L}_{\mathrm{obs}} + \mathcal{L}_{\mathrm{route}} + \mathcal{L}_{\mathrm{road}} + \mathcal{L}_{\mathrm{signal}}), \quad (6)
$$

(a) Obstacle mask   (b) Route mask
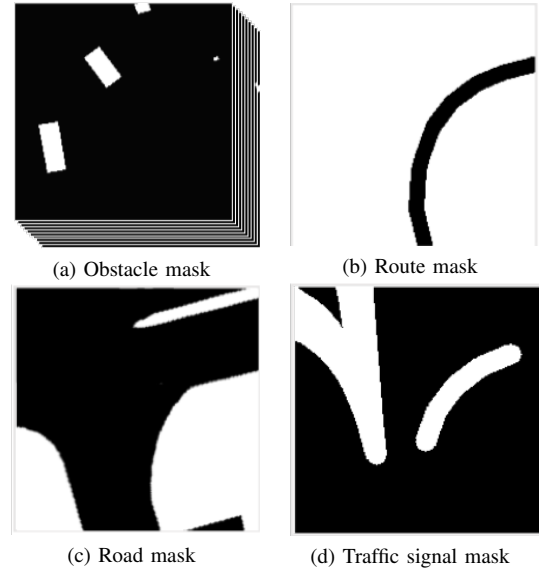
(c) Road mask   (d) Traffic signal mask

Fig. 5: Visualization of the binary masks we used in our task losses: (a) the obstacle masks that push our vehicle away from obstacles; (b) The route masks that keep our vehicle on the planned route by penalizing the behaviors that cause any portion of our vehicle that does not overlap with the planned route; (c) The road masks that keep our vehicle within drivable areas; (d) The masks that make our vehicle stop in front of traffic signals when they are red.

where λ_{task} is an empirical parameter. Like [5], we randomly drop out the imitation losses for some training samples to further benefit from the task losses.

B. Data Augmentation

To address the aforementioned distributional shift issue, we further introduce an iterative feedback synthesizer to enrich the training dataset besides the random horizontal perturbations [5]. Both data augmentation methods help our agent broaden its experience beyond the demonstrations.

1) Random Synthesizer: Similar to [5], our synthesizer perturbs trajectories randomly, creating off-road and collision scenarios. The start and end points of a trajectory are anchored, while we perturb one of the points on the trajectory and smooth across all other points. Only realistic perturbed trajectories are kept by thresholding on maximum curvature.

2) Feedback Synthesizer: The random synthesizer above achieves impressive performance improvement, as we shall see in Section IV. However, it fails to address one important issue due to its random nature, the bias in probability distributions of the synthesized states. To this end, following the principle of DAgger [2], we propose an iterative feedback synthesizer that generates the states that our agent is more likely to encounter by sampling the data based on the current policy. Thus, we seek to improve our policy by adding these states and their corresponding actions into the training dataset. If we denote our policy model at i-th iteration as π_i, given any start point t_0 in a trajectory, we can sample future T steps using π_i. This constructs a few new states that our agent is likely to encounter. To demonstrate how our agent should react to these states that are deviated from our optimal behavior, we propose to fit a smooth trajectory
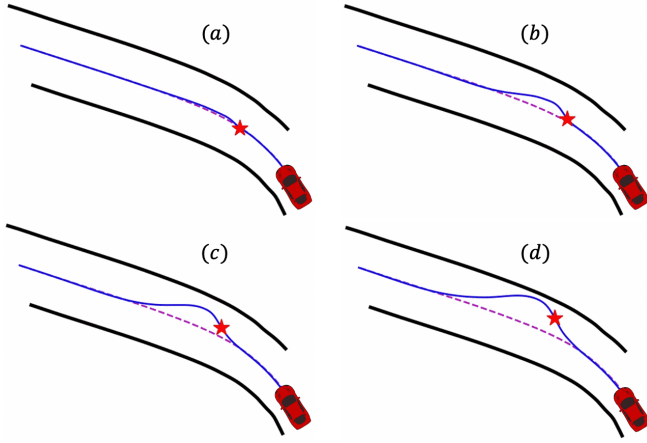
Fig. 6: The illustration of our feedback synthesizer. The "star" marks where we construct a new state by sampling an action using the current policy. We fit a smooth trajectory with necessary kinematic constraints after the "star" point, therefore avoiding any expert agents' dependence. (a) - (d) demonstrate the trajectories generated when $T = 2, 4, 6, 8$.

with necessary kinematic constraints. Figure 6 illustrates the deviated trajectories given different T values from 2 to 8. A complete step-by-step overview is detailed in Algorithm 1. The whole process is fully automatic and doesn't rely on the participation of any expert agent. Note that some synthesized samples may be discarded when the solver fails to compute feasible and smooth trajectories.

Algorithm 1 Feedback Synthesizer based Iterative Training

1: Input $\mathcal{D} = \mathcal{D}_0$. ▷ trajectories from human drivers
2: Train policy π_0 on $\mathcal{D}$.
3: **for** $i = 1$ to K **do**
4:     Sample T-step trajectories using π_{i-1} from $\mathcal{D}_0$.
5:     Get dataset $D_i = \{(s, a)\}$ of visited states by π_{i-1} and actions generated by smoothing.
6:     Aggregate datasets $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$
7:     Train policy π_i on $\mathcal{D}$.
8: **end for**

C. Post-processing Planner

This section introduces our post-processing planner, a new architecture that aims to offer better comfort and improve driving safety as a gatekeeper. The implementation has been released in Apollo 6.0 at `https://github.com/ApolloAuto/apollo/tree/master/modules/planning`.

The traditional path planning task is typically formulated as a quadratic optimization problem where lane boundaries, collision, traffic lights, and other driving conditions are formulated as bounding constraints [40]. Our new architecture is a joint optimization framework based on cues from both the kinematics and dynamics constraints and the above learning-based planner's output waypoints, as shown in Figure 7a. In the optimization framework, the bounding constraints remain unchanged. Meanwhile, the output waypoints of the learning-based planner are taken as its optimization objective. Finally,

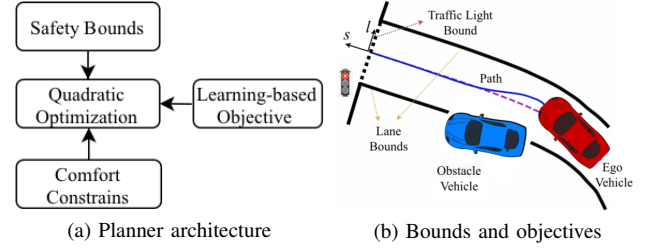(a) Planner architecture   (b) Bounds and objectives

Fig. 7: The illustration of our post-processing planner design. (a) We formulate a joint optimization problem by incorporating safety and comfort bounds into the framework while taking the learning-based planner's trajectory output as the optimization objective. (b) An example illustrates how we build safety and comfort bounds and the optimization objective.

the joint optimization framework produces eventual path trajectories that are fed to a control module.

Figure 7b illustrates an example on a real road. Our vehicle is approaching an intersection that generates a bound in front because of traffic signals. Lane boundaries and dynamic obstacles on the road also compose safety bounds in our optimization problem. Smoothness constraints ensure driving comfort. As opposed to the centerline (the default optimization objective in a traditional planner), the learning-based planner's waypoints guide our ego vehicle to keep a safe distance from the obstacle and successfully overtake it, which is similar to human driving behaviors. The final path, generated by the post-processing planner, follows our learning-based planner's high-level decision to overtake the obstacle vehicle while ensuring safety and comfort.

IV. EXPERIMENTS

A. Implementation Details

We have a squared BEV image with $W = 200$, $H = 200$, and $\rho = 0.2m/\text{pixel}$. In the image, our ego-vehicle is placed at $i_0 = 100$, $j_0 = 160$ in image coordinates. For the training data, we use a perception system in Apollo [41] that processes raw sensor measurements and produces object detection, tracking, and prediction results. Two seconds of history or predicted trajectories are rasterized in the corresponding image channels. Our learning-based planning module outputs trajectories with a time horizon as $N = 10$ and $\Delta t = 0.2s$. Our models were trained using the Adam optimizer with an initial learning rate of 0.0003.

B. Dataset and Augmented Data

Unlike prior works [17], [19], [3] that exploit a driving simulator and collect simulated driving data, we recruited our self-driving test vehicles and collected a total of 400 hours' driving data demonstrated by human-drivers in southern San Francisco bay area.

After necessary data pre-processing, we keep about 250k frames as our original training data before any data augmentation, denoted as D_o. Then, additional 400k frames are generated by our random synthesizer, denoted as D_r. When the feedback steps $T = 5$, our feedback synthesizer generates about 465k frames in one iteration, and they are denoted as D_f. To keep the training procedure efficient, we let the

TABLE I: Comparisons of different model configurations. The baseline model (M0) with pure imitation learning losses only achieves 15.7% ADS pass rate, while our proposed learning-based planner (M2) achieves 70.0% ADS pass rate. We further improve our ADS pass rate and comfort score by having an optional optimization-based post-processing planner (M3).

| Model	Description	Training Loss	Datasets (D_o, D_r, D_f)	Pass Rate	Comfort Score
M0	CNN backbone + LSTM decoder + imitation loss	$\mathcal{L}_{imit}$	✓, ×, ×	15.71%	0.1048
$M0a$	M0 + differentiable rasterizer and task losses + random synthesizer	$\mathcal{L}_{imit}+\mathcal{L}_{task}$	✓, ✓, ×	54.28%	0.0228
$M0b$	M0 + spatial attention branch	$\mathcal{L}_{imit}$	✓, ×, ×	10.00%	0.1105
M1	$M0a + M0b$	$\mathcal{L}_{imit}+\mathcal{L}_{task}$	✓, ✓, ×	55.71%	0.0214
$M1a$	M1 with attention applied directly on raw image rather than intermediates $\mathbf{F}_{\mathcal{I}}$	$\mathcal{L}_{imit}+\mathcal{L}_{task}$	✓, ✓, ×	42.85%	0.0070
$M1b$	M1 with attention module implemented inside the backbone, not as a branch	$\mathcal{L}_{imit}+\mathcal{L}_{task}$	✓, ✓, ×	51.43%	0.0103
$M1c$	M1 with truncated rasterization by vehicle shapes	$\mathcal{L}_{imit}+\mathcal{L}_{task}$	✓, ✓, ×	48.57%	0.0201
M2	M1 + feedback synthesizer	$\mathcal{L}_{imit}+\mathcal{L}_{task}$	✓, ✓, ✓	70.00%	0.0150
M3	M2 + post-processing planner	/	/	94.29%	0.1333
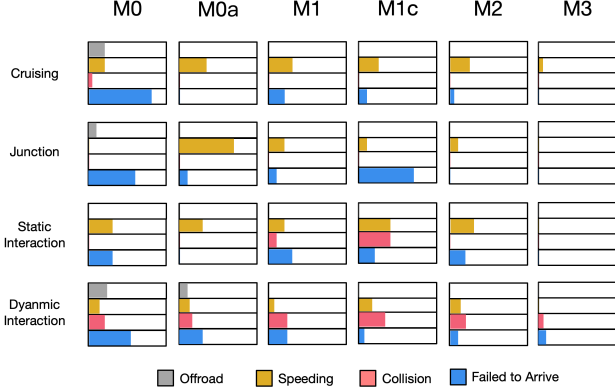
Fig. 8: Detailed analysis of failure reasons of different model configurations under different driving scenarios.

number of iterations $K = 1$ in the feedback synthesizer. Table I lists different configurations of training data when we evaluate different methods.

C. Evaluation Scenarios

To fully evaluate the proposed methods, we use the Apollo Dreamland simulator [41] and carefully created 70 ADS. Each ADS lasts about 30-90 seconds and includes one or more driving scenarios. The ADS are either handcrafted by our scene designers or taken from logs from real driving data (separate from the training data).

The driving scenarios in the 70 ADS can be roughly categorized into four types:

- **Cruising**: normal cruising in straight or curved roads without other traffic participants.
- **Junction**: junction related scenarios including left or right turns, U-turns, stop before a traffic signal, etc.
- **Static Interaction**: interaction with static obstacles, such as overtaking a stopped car.
- **Dynamic Interaction**: interaction with dynamic obstacles, for example, overtaking a slow vehicle.

They constitute a comprehensive driving test challenge specifically designed to evaluate the fundamental driving capabilities of a self-driving vehicle. Using these test scenarios, we carry out closed-loop evaluations where a control module executes our output waypoints, and the results are summarized in Table I.

D. Evaluation Metrics

Besides the pass or the success rate that prior works [5], [19] focus on, we propose to evaluate how comfortable the driving is and formulate a new comfort score. We assume that human drivers deliver a comfortable driving experience. The comfort score is calculated based on how similar our driving states are to human drivers. Particularly, we record the probabilities of a set of angular velocity (rad/s) and jerk (m/s^3) values (ω, j) in our human-driving data D_o. Given (ω, j) from certain driving data of an agent, we define the comfort score c as:

$$c = \frac{\sum_{i=1}^{n} P(\omega, j|D_o)}{n}, \qquad (7)$$

where $P(\omega, j|D_o)$ is the probabilities of values (ω, j) happened in human-driving data D_o. n is the number of frames, and (ω, j) are the corresponding values in our test driving data. Given (ω, j), when we look up the corresponding probabilities, ω and j are discretized into bins with sizes of 0.1 and 1.0, respectively.

Our ADS pass rate is based on a set of safety-related metrics, including collision, off-road, speeding, traffic-light violation, etc. When any of them is violated, we grade the entire ADS as failed. Of course, the agent has to reach the scheduled destination in a limited time to be considered successful, too.

E. Runtime Analysis

We evaluated our method's runtime performance with an Nvidia Titan-V GPU, Intel Core i7-9700K CPU, and 16GB Memory. The online inference time per frame is 10ms in rendering, 22 ms in model inference, and 15 ms (optional) in the post-processing planner. Note that our model inference time is much shorter than the prior work [5].

F. Comparison and Analysis

To demonstrate each of the above contributions' effectiveness, we show the ADS pass rate and comfort score with different methods in Table I. Also, the proportion of scenarios where we failed due to any violation of each particular rule in different driving scenarios using different methods are shown in Figure 8. Detailed analysis is discussed as follows.
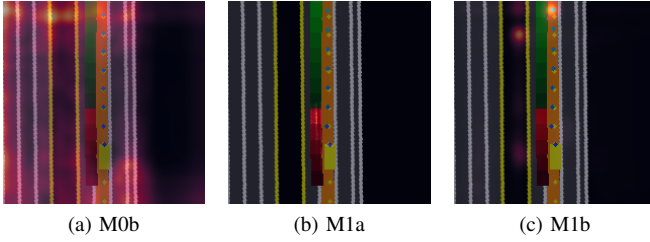
(a) M0b (b) M1a (c) M1b

Fig. 9: Comparisons of the attention heatmaps generated by different models in a dynamic interaction scenario. The corresponding heatmap from M1 can be found in Figure 10(a). Output and ground truth trajectories are illustrated as blue and yellow dots, respectively.
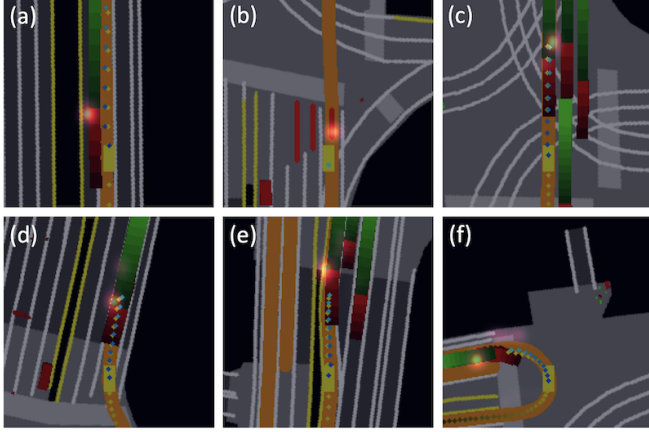

Fig. 10: Visualization of the attention heatmaps from M1 in different scenarios: (a) dynamic nudge; (b) stopping at a red light; (c) following a leading vehicle; (d) right turn; (e) yielding to a vehicle; (f) U-turn.

1) Task Losses: By comparing M0a with M0, it is observed that the numbers of scenarios where we failed due to off-road or collision reasons decrease. A similar trend can be found by observing the pass rate from M0b and M1. In particular, we add the differentiable rasterizer and task losses in our models M0a and M1. They play an important role in keeping us within drivable areas and avoiding obstacles.

2) Spatial Attention: The spatial attention in M0b gives a slightly worse pass rate with pure imitation losses comparing to M0. It implies that the attention can not improve the performance by itself without the important task losses. It makes sense since the task losses involve obstacles, traffic signals, and other traffic participants in explicit forms. They help the network locate the objects that it should pay attention to. In Figure 9a, we also find the attention heatmaps failed to converge in M0b. By comparing M1 with M0a, we find slight improvements in pass rate by having the spatial attention module. Also, we visualize the attention heatmaps from M1 in Figure 10, and they are easy to interpret and discover the attentive objects. To further verify our attention module design, we did experiments with M1a and M1b comparing with M1. M1a applies attention weights directly on raw image pixels resulting in higher computation complexity. In our experiment, M1a concluded with a lower ADS pass rate and less interpretable attention heatmaps, as shown in Figure 9b. M1b feeds the attention features back to the CNN backbone, also resulting in worse ADS pass rate and wrongly

focused attention heatmaps as shown in Figure 9c. Our design's inspiration is that keeping original CNN features in a separate branch is good for providing more meaningful features for downstream modules, therefore better for the overall performance.

3) Differentiable Rasterizer: When rasterizing vehicle boxes, it's intuitive to truncate the Gaussian distributions by box boundaries ensuring a precise representation. However, while we test it in experiment M1c, the experimental results suggest otherwise. M1c's ADS pass rate drops slightly compared to M1, the model without the truncation step. More importantly, M1 has fewer failed scenarios, especially in the collision category, as shown in Figure 8. It implies the long tail in distributions helps with obstacle avoidance. Besides this, we also find that M1c has a higher failure rate in junction scenarios where our vehicles stopped before traffic lights but failed to restart when they turned green. A possible reason is that the overlap between the long tail in distributions and the traffic signal mask provides more hints to the model and helps it learn the stop's causality.

4) Feedback Synthesizer: As we mentioned, our feedback synthesizer addresses the fundamental issue with imitation learning, the distributional shift. As a result, M2 improved the overall performance comparing to M1. Specifically, M2 achieves 70.0% ADS pass rate, about a 25% improvement over M1. The number of failure ADS decreases across almost all different scenarios or failure reasons, as shown in Figure 8.

5) Post-processing Planner: Our post-processing planner (M3) is designed as a gatekeeper with safety constraints. It's not surprising that it achieves a higher ADS pass rate, 94.29%. Better yet, the post-processing planner also significantly improves the comfort score with its smoothness constraints in the joint optimization framework. Note that we find pure imitation learning methods (M0 or M0b) without any data augmentation achieve similar comfort scores to M3. It seems that data augmentation hurts driving comfort, and our newly proposed comfort metric reveals this. We also evaluated the Post-processing Planner without the trajectory output from learning-based planner but using lane center-line as guidance (Traditional Planner). It has 88.57% pass rate and 0.1342 comfort score. Compared with the Post-processing Planner, the Traditional Planner has lower pass rate, as we expect that learning-based planner's high-level decision does perform better in complex scenarios.

6) Failure Modes: From Figure 8, it is easy to tell that speeding issues have already become a major problem for our learning-based planner, M2. These results seem to suggest that further improvements may be possible by adding a speeding loss. Another important failure reason is non-arrival. It happens when our ego vehicle gets stuck somewhere or has a too low speed. It probably means that we could extend our work and encourage our vehicle to keep moving forward at a steady speed by having a new loss which could be similar to the reward in reinforcement learning.

For our post-processing planner M3, safety constraints were implemented. We examined two failure scenarios that

are related to collision issues. Our vehicle is hit by a car behind it in both scenarios, which is a rear-end collision. Also note that, due to the limitation of our test environment, all other vehicles in the ADS only move forward by following scheduled routes and are not intelligent to avoid our ego-vehicle. This increases the chance of rear-end collisions, especially when our vehicle is slower than expected.

V. CONCLUSION AND FUTURE WORK

We have presented an imitation learning based planner designed for autonomous driving applications. We demonstrate that the overall ADS pass rate can be improved using necessary data augmentation techniques, including a novel feedback data synthesizer. Benefiting from the differentiable rasterizer, our learning-based planner runs inference in real-time, yielding output either directly used by a downstream control module or further fed into an optional post-processing planner. Task losses and spatial attention are introduced to help our system reason critical traffic participants on the road. Therefore, the full exploitation of the above modules enables our system to avoid collisions or off-road, smoothly drive through intersections with traffic signals, and even overtake slow vehicles by dynamically interacting with them. In the future, we plan to collect more real-road driving data, further strengthening our imitation learning capabilities. Also, the introduction of reinforcement learning maybe another direction to improve the overall performance to a higher level.

ACKNOWLEDGMENT

REFERENCES

[1] A. Sadat, M. Ren, A. Pokrovsky, Y.-C. Lin, E. Yumer, and R. Urtasun, "Jointly learnable behavior and trajectory planning for self-driving vehicles," in *IROS*, 2019, pp. 3949–3956.
[2] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *AISTATS*, 2011, pp. 627–635.
[3] A. Prakash, A. Behl, E. Ohn-Bar, K. Chitta, and A. Geiger, "Exploring data aggregation in policy learning for vision-based urban autonomous driving," in *CVPR*, 2020, pp. 11 763–11 773.
[4] V. Blukis, N. Brukhim, A. Bennett, R. Knepper, and Y. Artzi, "Following high-level navigation instructions on a simulated quadcopter with imitation learning," in *RSS*, June 2018.
[5] M. Bansal, A. Krizhevsky, and A. Ogale, "ChauffeurNet: Learning to drive by imitating the best and synthesizing the worst," in *RSS*, 2019.
[6] A. Bühler, A. Gaidon, A. Cramariuc, R. Ambrus, G. Rosman, and W. Burgard, "Driving through ghosts: Behavioral cloning with false positives," in *IROS*, 2020.
[7] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl, "Learning by cheating," in *CoRL*, 2020, pp. 66–75.
[8] E. Wang, H. Cui, S. Yalamanchi, M. Moorthy, and N. Djuric, "Improving movement predictions of traffic actors in bird's-eye view models using gans and differentiable trajectory rasterization," in *KDD*, 2020.
[9] H. Cui, T. Nguyen, F.-C. Chou, T.-H. Lin, J. Schneider, D. Bradley, and N. Djuric, "Deep kinematic models for kinematically feasible vehicle trajectory predictions," in *ICRA*, 2020, pp. 10 563–10 569.
[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[11] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *NeurIPS*, vol. 28, 2015, pp. 802–810.
[12] S. Hecker, D. Dai, A. Liniger, and L. Van Gool, "Learning accurate and human-like driving using semantic maps and attention," in *IROS*, 2020, pp. 2346–2353.
[13] J. Kim and M. Bansal, "Attentional bottleneck: Towards an interpretable deep driving network," in *CVPR Workshops*, 2020.
[14] D. A. Pomerleau, "ALVINN: an autonomous land vehicle in a neural network," in *NeurIPS*, 1988, pp. 305–313.
[15] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv1604.07316*, 2016.
[16] S. Hecker, D. Dai, and L. Van Gool, "End-to-end learning of driving models with surround-view cameras and route planners," in *ECCV*, 2018, pp. 435–453.
[17] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," in *ICRA*, 2018.
[18] A. Bewley, J. Rigley, Y. Liu, J. Hawke, R. Shen, V.-D. Lam, and A. Kendall, "Learning to drive from simulation without real world labels," in *ICRA*, 2019, pp. 4818–4824.
[19] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, "Exploring the limitations of behavior cloning for autonomous driving," in *ICCV*, 2019, pp. 9329–9338.
[20] M. Bhardwaj, B. Boots, and M. Mukadam, "Differentiable gaussian process motion planning," in *ICRA*, 2020.
[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *NeurIPS*, vol. 27, 2014.
[22] H. Cui, H. Shajari, S. Yalamanchi, and N. Djuric, "Ellipse loss for scene-compliant motion prediction," *arXiv2011.03139*, 2020.
[23] H. Kato, Y. Ushiku, and T. Harada, "Neural 3D mesh renderer," in *CVPR*, 2018, pp. 3907–3916.
[24] S. Liu, T. Li, W. Chen, and H. Li, "Soft rasterizer: A differentiable renderer for image-based 3D reasoning," in *ICCV*, 2019.
[25] F. Wang, M. Jiang, C. Qian, S. Yang, C. Li, H. Zhang, X. Wang, and X. Tang, "Residual attention network for image classification," in *CVPR*, 2017, pp. 3156–3164.
[26] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *CVPR*, 2018, pp. 7132–7141.
[27] W. Lu, G. Wan, Y. Zhou, X. Fu, P. Yuan, and S. Song, "DeepVCP: An end-to-end deep neural network for point cloud registration," in *ICCV*, 2019, pp. 12–21.
[28] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, vol. 30, 2017.
[29] J. Kim and J. Canny, "Interpretable learning for self-driving cars by visualizing causal attention," in *ICCV*, 2017, pp. 2942–2950.
[30] J. Kim, T. Misu, Y.-T. Chen, A. Tawari, and J. Canny, "Grounding human-to-vehicle advice for self-driving vehicles," in *CVPR*, 2019.
[31] Y. Zhou, G. Wan, S. Hou, L. Yu, G. Wang, X. Rui, and S. Song, "DA4AD: End-to-end deep attention-based visual localization for autonomous driving," in *ECCV*, 2020, pp. 271–289.
[32] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end simulated driving," in *AAAI*, vol. 31, no. 1, 2017.
[33] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile autonomous driving using end-to-end deep imitation learning," in *RSS*, 2018.
[34] S. Casas, W. Luo, and R. Urtasun, "IntentNet: Learning to predict intention from raw sensor data," in *CoRL*, 2018, pp. 947–956.
[35] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, "MultiPath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction," in *CoRL*, 2020, pp. 86–99.
[36] N. Djuric, V. Radosavljevic, H. Cui, T. Nguyen, F.-C. Chou, T.-H. Lin, N. Singh, and J. Schneider, "Uncertainty-aware short-term motion prediction of traffic actors for autonomous driving," in *WACV*, 2020.
[37] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018, pp. 4510–4520.
[38] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010, pp. 249–256.
[39] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *IV*, 2015, pp. 1094–1099.
[40] Y. Zhang, H. Sun, J. Zhou, J. Pan, J. Hu, and J. Miao, "Optimal vehicle path planning using quadratic optimization for baidu apollo open platform," in *IV*, 2020, pp. 978–984.
[41] "Baidu Apollo open platform," http://apollo.auto/.