Imitation Human Behavior Model for Interactive Autonomous Driving Simulation

Chenran Li[1], Akio Kodaira[1], Yatong Bai[1]

***Abstract*— Traditionally, evaluations for motion planning algorithms are preformed by letting the planner react to replayed data environments or algorithm-driven environments. However, these evaluation approaches have major issues: The replayed data environments cannot react to the actions of the subject planner, and the subject planner always favors actions that yields to other agents as a result. On the other hand, the algorithm-driven agent may not accurately represent human behavior and can cause the state transitional distribution in the simulated environment to be incorrect. On the other hand, algorithm-driven environments may not accurately represent human behaviors and can cause incorrect state transitional distributions. To solve this problem and enable interactive autonomous driving simulation, we implement a neural network policy based on the imitation learning framework to mimic the human behaviors. We also establish a simulator for closed-loop simulation and propose an "A* planning"-based pseudo-expert to provide expert demonstration for DAgger, a data augmentation technique for imitation learning. We validate our proposed framework in reactive simulation environments. The experimental results show that our framework can improve the performance of the human behavior model for autonomous driving.**

I. INTRODUCTION

Recently, the autonomous driving technology has been attracting huge attention. Along with the rapid development of the autonomous driving, a number of planning and control algorithms have been developed [1]–[4]. Since, the real world evaluation experiment for those developed planning algorithms is difficult to be carried out, it is essential to develop an accurate evaluation system using the simulator. Traditionally, the literature evaluates the planning algorithms by letting them react with the replayed data environments and algorithm-driven environments. However, there are two major issues associated with these evaluation approaches. On the one hand, the replayed data environments could not react to the planning algorithms' actions. As a result, the subject planners always favor actions of yielding to other agents. On the other hand, the behavior of the algorithm-driven environments that use model based planner [2] may not accurately represent the real environment. From the subject planner's perspective, this mismatch may cause the state transitional distribution in the simulated environment to be incorrect. To address these problems, a reactive model that can reproduce human behavior is required.

In this paper, we develop an imitation human behavior model refined by DAgger with planner based pseudo-expert. The contributions of this paper are threefold:

- We implement an imitation human behavior model based on Behavioral Cloning that can be improved by applying DAgger iteratively. We use the real traffic data provided by INTERACTION dataset [5] to train the human behavior model.
- We use an A* planner as a pseudo-expert and make the data augmentation process of DAgger fully automated and efficient.
- We validate the performance of the proposed method using reactive close-loop simulation on Intersection Scenario and Roundabout Scenario. The experimental results show the effectiveness of the proposed method.

A. Related work

1) Behavior cloning: One common approach to develop a human behavior reactive model for autonomous driving is behavior cloning (BC) [6]–[9]. BC, an instance of imitation learning algorithms, directly maps an observation to an action. BC can combine perception and control into an end-to-end trainable model. However, previous policies built upon BC are not robust in practical environment because of the inherent distribution shift problem [10], [11].

2) DAgger: One solution for the distributional shift problem of BC is using data aggregation algorithms such as DAgger. DAgger solves the distribution shift problem by collecting expert demonstration, and adding them to the training dataset for iterative model re-training. In autonomous driving field, for example, DAgger is used to solve the off-road control problem [12] and the urban-driving problem [13].

3) Planning: When a human expert provides the demonstration data for DAgger, the sample inefficiency problem of DAgger hinders its wide application in the field of autonomous driving. Specifically, it requires too much human effort to collect demonstration data for each specific driving scenario. An efficient way to perform DAgger is to use model-based planning algorithms developed for autonomous driving as a pseudo-expert. While model-based planners may not act the same as humans, they are able to achieve properties such as smoothness, mild acceleration, and safety in general. Therefore, model-based planners can act as pseudo-experts when a human-behavioral model fails to generate appropriate trajectories. We use an A* planner in this paper as the pseudo-expert to carry out DAgger efficiently.

4) Prediction: Prediction refers to the task of predicting the surrounding agents' future trajectories given the state history of the environment. As rule-based methods struggle to infer the sub-optimality of human behavior, many learning-based methods are proposed. The task of prediction is a re-

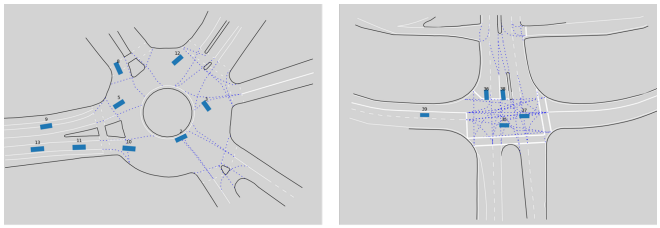
[1] Department of Mechanical Engineering, University of California, Berkeley, California, USA chenran_li, akio.kodaira, yatong_bai@berkeley.edu

laxed task of BC since both tasks require the same input and produce the same output. Therefore, an existing prediction model can be used as the starting point for constructing a BC model. Recently, the work [14] has introduced a novel neural network architecture for autonomous driving tasks. The inputs of neural network are the past trajectories of all agents in the environment and the map topology represented as a graph. In the prediction setting, the architecture in [14] achieves the best prediction accuracy among recent works [15]–[18]. The details of this architecture will be introduced in Section III-B. Despite its success in prediction setting, this architecture can still be non-robust in planning setting due to the distributional shift problem.

The paper is organized as follows: Section II introduces simulation construction. Section III describes the policy implementation based on the novel neural network. The implementations of failure identification for the BC model and the design of the DAgger process and the underlying pseudo-expert will be explained in Section IV and Section V, respectively. The results including simulations will be given in Section VI. Finally, Section VII presents the conclusions.

II. Simulation Construction

In this work, we design a neural network BC policy (NN policy) that imitates human behaviors by learning from real traffic data provided by the INTERACTION dataset [5]. We select two typical interaction scenarios shown in Figure 1. We train the NN policy over these two scenarios.



(a) Roundabout Scenario (b) Intersection Scenario

Fig. 1. Selected Scenarios form the INTERACTION dataset

The simulation is designed based on the same dataset. We first select the interaction group and the interaction period. We use the start frame of the interaction period as the initial frame and launch the simulation until the end of the selected interaction period. A vehicle selected randomly in the interaction group (referred to as the ego vehicle) is controlled by the neural network policy, while other vehicles in the interaction group are controlled by an expert planner. Vehicles outside the interaction group can also have mild influences on the interaction group. These non-interaction-group vehicles are controlled by the intelligent driver model (IDM). Therefore, the simulation will be a highly interactive environment, where every agents in the scenario will react to the ego vehicles action.

Notation	Definition	Length
$S_{\text{pla},i}$	i^{th} candidate NN trajectory	H_{pla}
$S_{\text{smo},i}$	i^{th} smoothed candidate NN trajectory	H_{pla}
$S^{\star}_{\text{smo}}$	Selected smoothed NN trajectory	H_{pla}
S_{ctrl}	Executed trajectory	H_{pla}
S_{exp}	Expert future trajectory	H_{pla}
S_{past}	Past trajectory	H_{past}

TABLE I
NOTATIONS FOR TRAJECTORIES.

III. Policy Implementation

A. Notations for datasets and trajectories

The NN policy refers to denote the control policy powered by the neural network. For an agent in the environment, we define the trajectory notations $S_{\text{pla},i}$, $S_{\text{smo},i}$, $S^{\star}_{\text{smo}}$, S_{ctrl}, S_{exp}, S_{past} as in Table I, where $i = 1,\dots,K$, and K is the number of candidate trajectories generated by the neural network. K is a hyperparameter of the NN policy and is set to 6 in this paper. Note that S_{ctrl} can be generated by either the NN policy or some other policies. The expert trajectory S_{exp} can either come from a training dataset or a planner algorithm. The notations H_{pla}, H_{past} denote the lengths of the corresponding trajectories. In Section III-C, we discuss how $S_{\text{smo},i}$, $S^{\star}_{\text{smo}}$, and S_{ctrl} are computed.

When the NN policy is trained or evaluated with a dataset, the superscipt $^{(j)}$ denotes the trajectory corresponds to the j^{th} data. When the NN policy is applied in a real-time environment, the superscript $^{(t)}$ denotes the trajectory is generated at time step t. For a trajectory $S^{(j)}$, the notation $s^{(j,h)}$ denotes the h^{th} coordinate in $S^{(j)}$.

The set $\mathscr{D} := \{(S^{(j)}_{\text{past}}, S^{(j)}_{\text{exp}})\}, j = 1,2,\cdots,n$ defines a dataset containing n ground-truth trajectories. For each $S^{(j)}_{\text{past}}$, the NN policy generates a set of predicted future trajectories $S^{(j)}_{\text{pla},i}$ with $i = 1,2,\cdots,K$. Moreover, the neural network assigns each $S^{(j)}_{\text{pla},i}$ with a confidence level, defined as a probability measure $P^{(j)}_i$ satisfying $\sum_{i=1}^{K} P^{(j)}_i = 1$ for all j. The output trajectories are sorted by confidence in a decreasing order (i. e. $P^{(j)}_i \geq P^{(j)}_{i+1}$ for all i). We represent the NN policy as $f_\theta : S_{\text{past}} \to \{(S_{\text{pla},i}, P)\}_{i=1}^{K}$, where θ denotes the trainable parameters in the underlying neural network.

The following two metrics are used to evaluate a candidate trajectory $S^{(j)}_{\text{pla},i}$ with respect to an expert trajectory $S^{(j)}_{\text{exp}}$:

- Average distance error (ADE):

$$ADE\left(S^{(j)}_{\text{pla},i}, S^{(j)}_{\text{exp}}\right) = \frac{1}{H_{\text{pla}}} \sum_{h=1}^{H_{\text{pla}}} \left\| s^{(j,h)}_{\text{pla},i} - s^{(j,h)}_{\text{exp}} \right\|_2^2 \quad (1)$$

- Final distance error (FDE):

$$FDE\left(S^{(j)}_{\text{pla},i}, S^{(j)}_{\text{exp}}\right) = \left\| s^{(j,H_{\text{pla}})}_{\text{pla},i} - s^{(j,H_{\text{pla}})}_{\text{exp}} \right\|_2^2 \quad (2)$$

The common metrics below are used for evaluating a model f_θ:

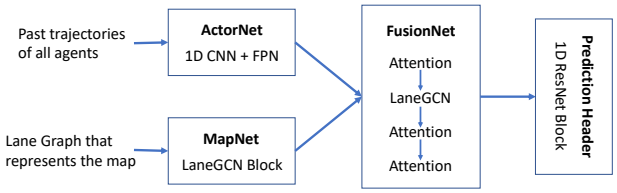


Fig. 2. The architecture of the NN policy.

- ADE and FDE: We overload the notations for model evaluation as

$$
\begin{aligned}
ADE(f_\theta, \mathscr{D}) &= \frac{1}{n}\sum_{j=1}^{n} \min_{i=1,\dots,K} ADE\left(S_{\text{pla},i}^{(j)}, S_{\text{exp}}^{(j)}\right) \\
FDE(f_\theta, \mathscr{D}) &= \frac{1}{n}\sum_{j=1}^{n} \min_{i=1,\dots,K} FDE\left(S_{\text{pla},i}^{(j)}, S_{\text{exp}}^{(j)}\right) \\
\text{where } &\left\{(S_{\text{pla},i}^{(j)}, P_i^{(j)})\right\}_{i=1}^{K} = f_\theta\left(S_{\text{past}}^{(j)}\right), \quad j = 1,\dots,n, \\
&\left\{S_{\text{past}}^{(j)}, S_{\text{exp}}^{(j)}\right\}_{j=1}^{n} \sim \mathscr{D}.
\end{aligned} \tag{3}
$$

  Note that when $K = 1$, the ADE and the FDE across the dataset is the ADE and the FDE between the ground-truth and the most likely trajectory.
- Missing Rate (MR): the percentage of the cases when FDE is greater than 2 meters.

B. Neural network architecture

The paper [14] has proposed a novel neural network architecture and has demonstrated the superior performance of such an architecture for prediction tasks on the Argoverse dataset [19]. In this paper, we adapt this architecture for both prediction and control tasks on the INTERACTION dataset [5]. INTERACTION consists data of interactive scenarios in complex environments, and is thus a more suitable evaluation testbed for an interactive policy. INTERACTION is recorded from real-world experiences. Specifically, the training data for the NN policy is the training sets of the "roundabout" environment and the "unsignaled intersection" environment in INTERACTION, consisting of 101,681 input-target pairs in total. The validation data is from the same environments, consisting of 17,853 pairs in total. The training inputs are the trajectories of all agents over the past 10 time steps ($H_{\text{past}} = 10$), and the targets are the trajectories of the controlled agent in the future 30 time steps ($H_{\text{pla}} = 30$). The neural network generates candidate trajectories of the ego vehicle and a confidence level for each candidate trajectory. As shown in Figure 2, The neural network consists of the following four subnetworks: a MapNet, an ActorNet, a FusionNet, and a prediction header. In each subnetwork, layer normalization and a rectified linear unit (ReLU) activation function and is applied after each layer.

MapNet is a LaneGCN module, a modified graph convolutional network (GCN) tailored specifically for autonomous driving applications. LaneGCN extracts features from the map graph. A traditional GCN also encodes connectivities but is invariant to orientations: the directional information between two nodes will be lost. LaneGCN encodes directional information by incorporating four graphs, representing the connectivities between nodes in the front, rear, left, and right directions, respectively. The center points of lane sections are represented as the graph nodes, and the connectivities between the lane sections along a specific direction are represented as the graph edges in the corresponding graph.

ActorNet consists of several one-dimensional convolutional (Conv1d) blocks and feature pyramid network (FPN) blocks. The inputs to the ActorNet are the trajectories of the past H_{past} time steps of all agents in the environment. ActorNet is compatible with an arbitrary H_{past}. The Conv1d and Attention blocks extract features with a fixed dimension of 128 (this dimension is a hyperparameter of LaneGCN) from the past trajectories.

FusionNet consists of several attention blocks. The inputs to the FusionNet are the features extracted by the MapNet and the ActorNet. The FusionNet consists of another LaneGCN block and several attention blocks, and allows the map information to interact with the agent states.

The prediction header is a residual block that takes the features encoded by the FusionNet as inputs and outputs the K most probable future trajectories of all agents in the environment over the next H_{pla} timesteps. The header also provides the confidence of each candidate trajectory. The trajectories $\{S_{\text{pla},i}\}_{i=1}^{K}$ corresponding to the controlled agent are the outputs of the entire policy. As in [14], during the training phase, we optimize the NN policy to mimic the target expert trajectories S_{exp} in the dataset. We first define a regression loss function and a classification loss function as below for each j:

$$
\begin{aligned}
\ell_{\text{cls}}\left(\{P_i^{(j)}\}_{i=1}^{K}\right) &:= \frac{1}{K-1}\sum_{i\neq\hat{i}} \max\left\{0, P_i^{(j)} - P_{\hat{i}_j}^{(j)} + \varepsilon\right\}, \\
\ell_{\text{reg}}\left(\{S_{\text{pla},i}^{(j)}\}_{i=1}^{K}, S_{\text{exp}}^{(j)}\right) &:= \frac{1}{H_{\text{pla}}}\sum_{h=1}^{H_{\text{pla}}} d\left(s_{\text{pla},\hat{i}_j}^{(j,h)} - s_{\text{exp}}^{(j,h)}\right)
\end{aligned}
$$

where $\hat{i}_j$ is defined as $\arg\min_{i=1,\dots,K} FDE\left(S_{\text{pla},i}^{(j)}, S_{\text{exp}}^{(j)}\right)$. Note that ℓ_{cls} is the max-margin loss, and ε is a preset margin hyperparameter. Furthermore, for a vector p, the function $d(p)$ is defined as the following:

$$
d(p) := \begin{cases} \frac{1}{2}\|p\|_2^2 & \text{if } \|p\|_1 < 1, \\ \|p\|_1 - \frac{1}{2} & \text{otherwise,} \end{cases}
$$

which is the smoothed ℓ_1 loss.

The training formulation of the NN policy is the optimization problem

$$
\begin{aligned}
&\min_\theta \frac{1}{n}\sum_{j=1}^{n} \ell_{\text{reg}}\left(\{S_{\text{pla},i}^{(j)}\}_{i=1}^{K}, S_{\text{exp}}^{(j)}\right) + \frac{1}{n}\sum_{j=1}^{n} \ell_{\text{cls}}\left(\{P_i^{(j)}\}_{i=1}^{K}\right), \\
&\text{where } \left\{(S_{\text{pla},i}^{(j)}, P_i^{(j)})\right\}_{i=1}^{K} = f_\theta\left(S_{\text{past}}^{(j)}\right), \quad j = 1,\dots,n.
\end{aligned}
$$

C. From neural network to closed-loop controls

In the controls setting, at each time step t, a control trajectory $S_{\text{ctrl}}^{(t)}$ is generated, and its first coordinate decides

the location of the controlled agent at time step $t+1$. We assume to have a perfect actuator that allows the controlled agent to precisely follow $S^{(t)}_{\text{ctrl}}$. Since the NN policy takes the trajectories over the past H_{past} time steps as the input, the first H_{past} time steps must be controlled by an expert. Therefore, the control policy uses

$$S^{(t)}_{\text{ctrl}} = S^{(t)}_{\text{exp}}, \quad t = 1, \dots, H_{\text{past}}, \tag{4}$$

where S_{exp} is produced by an expert planning algorithm.

Starting from $t = H_{\text{past}} + 1$, the neural network trajectories become available. To ensure policy stability, we smooth the neural network trajectories with the previous executed control trajectory. In later parts of this paper, we will show that such a smoothing procedure alleviates the distributional shift problem inherent to imitation learning, and is crucial to the performance of the NN policy. At each time step t, the NN policy generates the candidate trajectories $\{S^{(t)}_{\text{pla},i}\}_{i=1}^{K}$, where K is the number of the candidate trajectories. The smoothed trajectory is generated using the following mixing recursion:

$$S^{(t)}_{\text{smo},i} = \alpha S^{(t)}_{\text{pla},i} + (1-\alpha) S^{(t-1)}_{\text{ctrl}}, \quad \forall t > H_{\text{past}}, \quad \forall i, \tag{5}$$

where $\alpha \in [0,1]$ is a hyperparameter that controls the strength of the smoothing operation. The mixing procedure of the trajectories is specifically defined as the following: Since $S^{(t-1)}_{\text{ctrl}}$ is generated at time step $t-1$, it should be shifted backward by one step. Therefore, $s^{(t,h)}_{\text{smo},i}$ is a convex combination of $s^{(t,h)}_{\text{pla},i}$ and $s^{(t-1,h+1)}_{\text{ctrl}}$.[1]

Next, the best smoothed candidate trajectory is selected via the discrete optimization problem:

$$S^{\star(t)}_{\text{smo}} = \min_{S^{(t)} \in \{S^{(t)}_{\text{smo},i}\}_{i=1}^{K}} c(S^{(t)}), \quad \forall t > H_{\text{past}},$$
$$\text{where } c(S^{(t)}) := \sum_{h=1}^{H_{\text{pla}}} \left\| s^{(t,h)} - s^{(t,h)}_{\text{exp}} \right\|_2^2.$$

After training on the INTERACTION dataset, the NN policy learns to produce trajectories that resemble the target trajectories in the dataset. However, LaneGCN is not perfect when trained with only offline data, because it does not have the knowledge about how its trajectories interact with the environment. To gain such knowledge and improve closed-loop performance, the NN policy needs to learn from closed-loop data, where its trajectories affect future states. To ensure safety and stability throughout this procedure, an expert takes over the control task when the NN policy generates an poor trajectory. A trajectory is considered poor if it is dynamically infeasible or differs noticeably from the reference. Specifically,

$$S^{(t)}_{\text{ctrl}} = \begin{cases} S^{\star(t)}_{\text{smo}} & \text{if } \mathbb{I}\big(A(S^{\star(t)}_{\text{smo}})\big) = 0, \\ S^{(t)}_{\text{exp}} & \text{if } \mathbb{I}\big(A(S^{\star(t)}_{\text{smo}})\big) > 0, \end{cases} \quad \forall t > H_{\text{past}}, \tag{6}$$

[1]For example, the first step of $S^{(t)}_{\text{smo},i}$ should be a convex combination of the fist step of $S^{(t)}_{\text{pla},i}$ and the second step of $S^{(t-1)}_{\text{ctrl}}$.

where $A(S)$ is an event that indicates that the trajectory S is poor, and $\mathbb{I}(\cdot)$ is the indicator function. In the next section, we will discuss the design of the event $A(S)$.

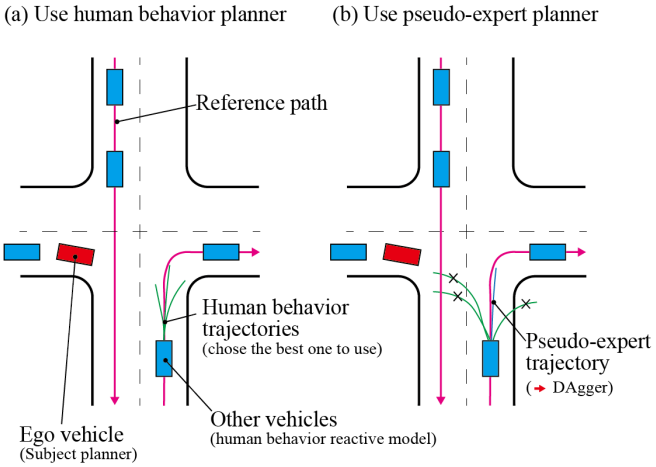


Fig. 3. When a smoothed neural network trajectory has high quality (the complement of A happens), it is used to control the vehicle. When A happens, the vehicle is controlled by a pseudo-expert trajectory generated by a non-learning-based algorithm, and this pseudo-expert trajectory is then added to the training set to perform DAgger.

IV. Failure Identification

To evaluate the trajectory generated by policy and apply DAgger, the simulation need to identify the failure of the policy. In this section, we propose a two-layer processes for failure identification. First, as implemented in [20], a classifier predicts whether a trajectory will deviate too much from the pseudo-expert trajectory. This classifier is trained on the training set of INTERACTION. Then, when a trajectory is fed into the simulator, a rule-based error threshold identifies potential failures in term of maintaining close loop stability. When a failure is identified, the simulator switches to the pseudo-expert trajectory.

A. Failure Classifier

The work [20] has shown that a dedicated classifier can achieve a higher successful rate of identifying poor trajectories, compared with methods such as ensemble. In this work, we use a classifier with a similar architecture as the NN policy, with the detailed architecture illustrated in Figure 4. Compared to the NN policy, the classifier differs in the following:

- An additional ActorNet extracts features from the neural network trajectory produced at the current time step.
- The classification header has one 1-dimensional residual block with a single output.

The failure classifier is trained after the training of the NN policy has completed. The classifier performs binary classification, where 1 denotes that the neural network trajectory is poor, and 0 denotes that the complement. The training inputs for the classifier are the past trajectories of all agents in the environment, as well as the neural network trajectories of the controlled agent. The training label corresponding

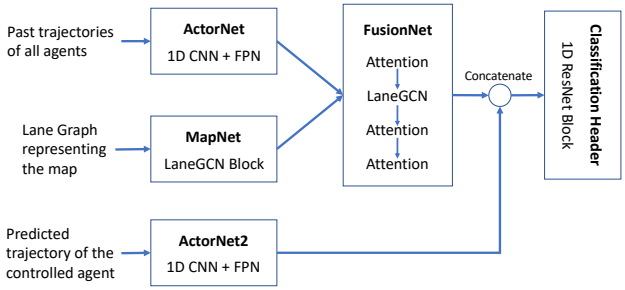


Fig. 4. The architecture of the failure classifier.

to a neural network trajectory $S^{(j)}_{\text{pla},i}$ for the classifier is $\mathbb{I}\big(FDE(S^{(j)}_{\text{pla},i}, S^{(j)}_{\text{exp}}) > 1 \text{ meter}\big)$. The classifier is trained using the binary cross-entropy loss, with the optimizer chosen to be Adam.

The failure classifier trained on the training set of the INTERACTION dataset achieves 100% evaluation accuracy on the validation set.

B. Error Indicator

The rule based error threshold is designed for detecting the closed loop unstable trajectory and collision on line boundary.

The error indicator function for collision on line boundary is defined as

$$\mathbb{I}_{\text{Bou}}(S^{(t)}_{\text{smo},i}) = \begin{cases} \infty & \text{if } \exists h \text{ s. t. } \left\| s^{(t,h)}_{\text{smo},i} - s^{(t,h)}_{\text{ref},i} \right\|_2 > W \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

where $s^{(t,h)}_{\text{ref},i}$ denotes the coordinate of the closest point to $s^{(t,h)}_{\text{smo},i}$ on the reference path and W is the feasible width of current line.

The error indicator function for closed-loop instablility is defined as

$$\mathbb{I}_{\text{Clo}}(S^{(t)}_{\text{smo},i}) = \begin{cases} \infty & \text{if } \exists h \text{ s. t. } \left\| s^{(t,h)}_{\text{smo},i} - s^{(t-1,h+1)}_{\text{ctrl}} \right\|_2 > \text{Tol}, \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

where Tol is the distance tolerance threshold.

When any of the above indicator functions is ∞, the input trajectory will be signed as a failed trajectory. When all candidate trajectories are marked as failed, the simulator switches to expert trajectory and the expert planner takes over.

V. Pseudo-Expert for Imitation

DAgger is a powerful method that tackles the distribution shift problem of imitation learning. It augments the training set with data collected in the online setting. Unfortunately, for autonomous driving tasks, it is prohibitively hard for human to relabel the whole trajectory in a online setting. To this end, we propose a pseudo-expert based on the A* planning algorithm.

A. Path Correction

As the vehicle may not drive on the defined reference path in general, a path planning method is needed for generating a smoothed correction path back to the reference path before motion planning. Therefore, we combine the pure pursuit controller [4] and the kinematic bicycle model to ensure the smoothness and the kinematic feasibility of the trajectories. This method has been evaluated in [1].

The kinematic bicycle model can be written as

$$\begin{aligned} \dot{x} &= v\cos(\psi+\beta) \\ \dot{y} &= v\sin(\psi+\beta) \\ \dot{\psi} &= \frac{v}{l_r}\sin(\beta) \\ \beta &= \tan^{-1}\left(\frac{l_r}{l_f+l_r}\tan(\delta_f)\right) \end{aligned} \tag{9}$$

where x and y are the coordinates of the center of mass, ψ is the inertial heading, v is the speed of the vehicle, l_r and l_f represent the distance from the center of the mass of the vehicle to the front and rear axles, respectively. Moreover, β is the angle between the current velocity of the center of mass and the longitudinal axis of the car, δ_f is the front steering angle.

With the δ_f provided by pure pursuit controller, by giving a small constant speed v, we can generate a smoothed path by updating the kinematic bicycle model.

B. A* Rough Motion Planning

The search space is defined as a three-dimension space including time, velocity, and longitudinal distance. A single node n_i in the search space can be represented as (s_i, v_i, t_i), where s_i is the coordinate of the vehicle on its reference path, v_i is the longitudinal velocity of the vehicle, t_i is the time step. Therefore, the beginning node and goal node can be represented as $(s_0, v_0, 0)$ and (s_g, v_g, t_g) respectively. Note here, since normally, when driving on the reference path, it is not necessary to have a certain position as the goal, the goal state can be relaxed as (s_H, v_H, H), where H is the planning horizon, that is any state at the end of the planning horizon.

The transition between nodes is defined as applying the acceleration a along the reference path. The next node of n_i, denoted as n_{i+1} is defined by

$$n_{i+1} = (s_i + \frac{1}{2}a\Delta t^2, v_i + a\Delta t, t_i + \Delta t) \tag{10}$$

Here Δt is the minimum time interval. we discretized acceleration as $a \in \{-2,-1,0,1,2\} m/s^2$. Such discretization was used in [2] which well covered normal human actions. The cost function C for the transition between nodes is defined as

$$\begin{aligned} C(n_i, a, n_j) = w_1 a^2 + w_2 \kappa(s_{i+1}) v_{i+1}^2 \\ + w_3 (v_{i+1} - v_d)^2 + \mathbb{I}_{\text{Col}}(n_{i+1}) \end{aligned} \tag{11}$$

where $\kappa(s_{i+1})$ is the curvature of the reference path at s_{i+1}, v_d is the desired speed, $\mathbb{I}_{\text{Col}}(n_{i+1})$ is a indicator function that is ∞ if collision happens at n_{i+1} and 0 otherwise. Therefore, the fist and second terms in cost function penalizes large

longitudinal and lateral acceleration for comfort. The third term encourage the vehicle to follow an appropriate speed.

C. Trajectory Smoothing

To reduce the sampling time and smooth previous searched rough motion, a smoothing process is needed. We formulate it as an optimization problem. The optimization variables are $\tilde{s}_1, \tilde{s}_2, ..., \tilde{s}_M$ where the the current minimum time interval Δt_s is defined by $\Delta t = n\Delta t_s$. Therefore, the smoothed velocity $\tilde{v}$, acceleration $\tilde{a}$ and jerk $\tilde{j}$ at each time step can be approximated by

$$\begin{aligned} \tilde{v}(k\Delta t_s) &= \frac{\tilde{s}_k - \tilde{s}_{k-1}}{\Delta t_s} \\ \tilde{a}(k\Delta t_s) &= \frac{\tilde{v}_k - \tilde{v}_{k-1}}{\Delta t_s} \\ \tilde{j}(k\Delta t_s) &= \frac{\tilde{a}_k - \tilde{a}_{k-1}}{\Delta t_s} \end{aligned} \tag{12}$$

which are linear function of $\tilde{s}_k$. Therefore the optimization can be formulated as a quadratic programming as following

$$\begin{aligned} &\min_{s_1,...,s_M} \sum_{t=k\Delta t_s} (\tilde{w}_1 \tilde{a}(t)^2 + \tilde{w}_2 \tilde{j}(t)^2) + \tilde{w}_3 \sum_{t=k\Delta t} (\tilde{s}(t) - s(t))^2 \\ &s.t. \quad A_s s \leq b_s \end{aligned} \tag{13}$$

where the $A_s s \leq b$ are constructed via linear hard constraints, such as longitudinal acceleration $a_{min} < \tilde{a} < a_{max}$ and the collision avoidance constraints derived from $I_{\text{Col}}(n_{i+1})$. With the smoothed the trajectory points, We then apply the cubic spline to generate the function of expert trajectory $S_{\text{Exp}}(t)$.

VI. Experiments

To evaluate the proposed method, several open-loop and closed-loop experiments were conducted and are discussed in this section. We first conducted several open-loop model evaluation experiments which evaluate the model as a predictor to predict the human drivers' future trajectory. Then, we use the model as the planner for the robot vehicle in our simulator. We then collect the data from the simulator and apply the DAgger and evaluate the DAgger model in the same pipeline.

A. Open-loop model evaluation

To evaluate the ability of the model to produce the human behavior like trajectory, we first conducted the open-loop experiments. In these experiments, we use the validation set which we split from the original dataset to initial the simulation. The vehicles in the simulation were driven by data. We then used the model to generate the trajectories and compared them with the data.

Table II shows the result of the open loop evaluation of the proposed method and the baseline which is the current first on the leader board. As it can be seen in the table, the model used in this work can generate similar trajectories as human drivers. Meanwhile, after applying DAgger with pseudo-expert, every metric didn't change much, which means even we are not using human experts the model doesn't lose its generality on reproducing human behavior.

Model	MinADE	MinFDE	MR
baseline	0.2020	0.5902	0.0284
Before DAgger (K = 6)	0.1595	0.4397	0.0149
DAgger Iteration 1 (K = 6)	0.1615	0.4476	0.0160
DAgger Iteration 2 (K = 6)	0.1635	0.4513	0.0173
DAgger Iteration 3 (K = 6)	0.1620	0.4494	0.0160
DAgger Iteration 4 (K = 6)	0.1614	0.4457	0.0162
Before DAgger (K = 1)	0.3474	1.1307	0.1436
DAgger Iteration 1 (K = 1)	0.3549	1.1519	0.1482
DAgger Iteration 2 (K = 1)	0.3536	1.1480	0.1498
DAgger Iteration 3 (K = 1)	0.3512	1.1429	0.1433
DAgger Iteration 4 (K = 1)	0.3509	1.1389	0.1462

TABLE II

EXPERIMENT RESULTS WITH LANEGCN-ACTOR-FUSIONNET NN POLICY.

B. Close-loop experiment

In this subsection, the proposed method was evaluated in the closed-loop simulation. We did the experiment in four different settings that are model without closed-loop smoothing; DAggered model without closed-loop smoothing; model with closed-loop smoothing; DAggered model with closed-loop smoothing. We compared the number of failure frames of each setting and a detailed discussion is provided.

In these experiments, we set the time step as 0.1 second. The total simulation length of the intersection scenario and the roundabout scenario are 52 and 64 steps. At each step, the simulator will call the proposed framework to generate the planning trajectory.

Model	Smoothing	Number of failure case
Original	False	49
DAgger	False	47
Original	True	8
DAgger	True	0

TABLE III

CLOSE-LOOP SIMULATION ON INTERSECTION SCENARIO.

As it can be seen in Table III, the original model without the smoothing layer failed in the most of steps. After DAgger, the model without smoothing didn't receive considerable improvement. Meanwhile, after applying the smoothing process, the performance received a dramatic improvement. When the smoothing process was used, the DAggered model can help more.

We further looked into this phenomenon by plotting the planned trajectory at each step.

Figure 5 and Figure 6 show the closed-loop simulation of the original model without the smoothing layer. As it can be seen, the model can generate good trajectories at first update. However, as soon as the new state counted into the input, the model failed immediately.

Figure 7 and Figure 8 show the closed loop simulation of the original model with the smoothing layer. As it can be seen, with the smoothing layer, the model can keep generate good trajectories.

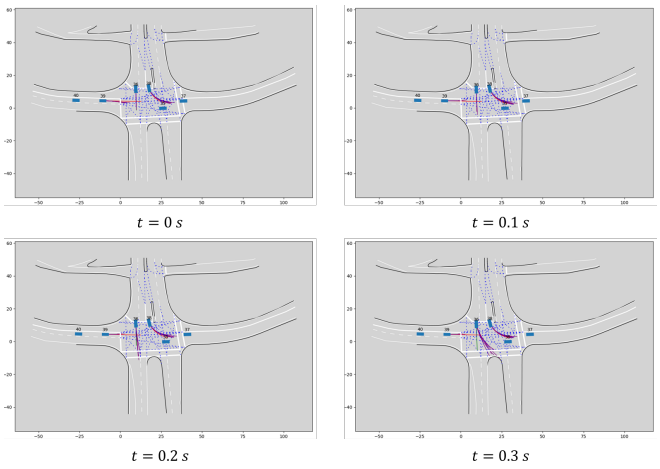


Fig. 5. Close-loop Simulation on Intersection Scenario by Original Model Without Smoothing

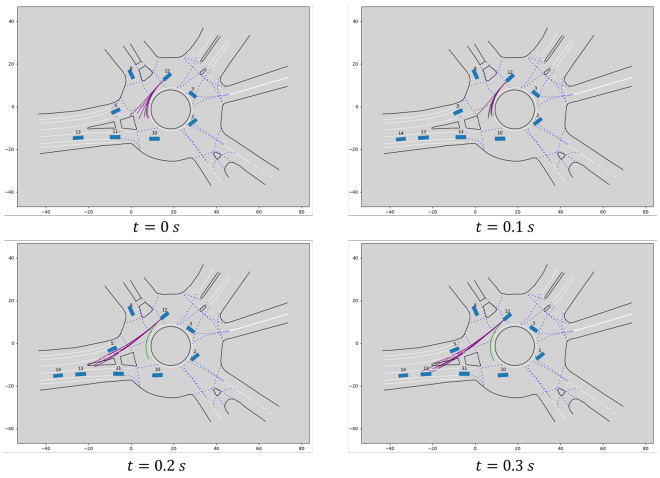


Fig. 6. Close-loop Simulation on Roundabout Scenario by Original Model Without Smoothing

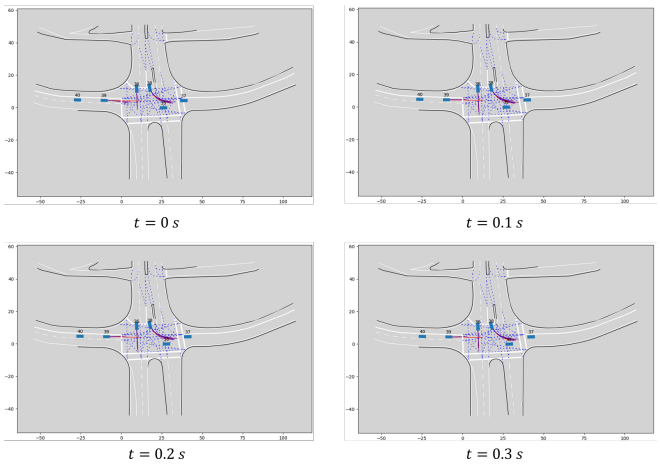


Fig. 7. Close-loop Simulation on Intersection Scenario by Original Model With Smoothing

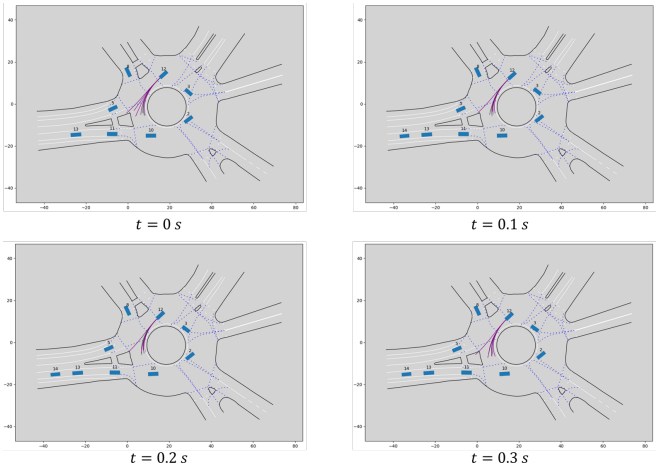


Fig. 8. Close-loop Simulation on Roundabout Scenario by Original Model With Smoothing

We conclude the reason for this phenomenon on the sense of close loop stability. Without the smoothing layer, the trajectory generated in different steps may have much difference as the reaction of other agents are also changing. Therefore, the historical states may have a great behavior switch, which causes the model to fail as this switch doesn't really happen in human behavior. After the smoothing layer is applied, the trajectory keeps some information of the last plan, which smooth the behavior change. Therefore, the input historical states will not shift from the dataset too much. As the smoothing layer solve the most of distribution shifting problem, the DAgger model can fix the left shifting problem easily. Therefore, while the performance of the model without smoothing wasn't improved much by applying DAgger, the smoothed model can benefit much from it.

VII. CONCLUSIONS

In this work, we implement an imitation learning framework to imitate the human behavior model for interactive autonomous driving simulation. We establish a simulator for closed-loop simulation and proposed an A* planning based pseudo-expert for DAgger. We evaluate our framework on the reactive simulation and conclude the reason for the failure of the imitation model. The results of the simulations demonstrate the effectiveness of the proposed method. The future work includes exploring a better method for ensuring the closed-loop stability of the imitation model and the study of better metrics for evaluating the effectiveness of the intended sub-optimal reactive model such as the human behavior model and critical agents model.

REFERENCES

[1] Z. Li, W. Zhan, L. Sun, C.-Y. Chan, and M. Tomizuka, "Adaptive sampling-based motion planning with a non-conservatively defensive strategy for autonomous driving," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15 632–15 638, 2020.

[2] C. Hubmann, M. Aeberhard, and C. Stiller, "A generic driving strategy for urban environments," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2016, pp. 1010–1016.

[3] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[4] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, Tech. Rep., 1992.

[5] W. Zhan, L. Sun, D. Wang, H. Shi, A. Clausse, M. Naumann, J. Kummerle, H. Konigshof, C. Stiller, A. de La Fortelle *et al.*, "Interaction dataset: An international, adversarial and cooperative motion dataset in interactive driving scenarios with semantic maps," *arXiv preprint arXiv:1910.03088*, 2019.

[6] J. Lee, "A survey of robot learning from demonstrations for human-robot collaboration," *arXiv preprint arXiv:1710.08789*, 2017.

[7] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 961–971.

[8] Y. Chai, B. Sapp, M. Bansal, and D. Anguelov, "Multipath: Multiple probabilistic anchor trajectory hypotheses for behavior prediction," *arXiv preprint arXiv:1910.05449*, 2019.

[9] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE AND PSYCHOLOGY ..., Tech. Rep., 1989.

[10] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 2010, pp. 661–668.

[11] S. Ross, G. Gordon, and D. 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*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[12] 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," *arXiv preprint arXiv:1709.07174*, 2017.

[13] 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 *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 763–11 773.

[14] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun, "Learning lane graph representations for motion forecasting," in *ECCV*, 2020.

[15] X. Huang, S. G. McGill, J. A. DeCastro, L. Fletcher, J. J. Leonard, B. C. Williams, and G. Rosman, "Diversitygan: Diversity-aware vehicle motion prediction via latent semantic sampling," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5089–5096, 2020.

[16] H. Cui, V. Radosavljevic, F.-C. Chou, T.-H. Lin, T. Nguyen, T.-K. Huang, J. Schneider, and N. Djuric, "Multimodal trajectory predictions for autonomous driving using deep convolutional networks," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 2090–2096.

[17] N. Djuric, V. Radosavljevic, H. Cui, T. Nguyen, F.-C. Chou, T.-H. Lin, and J. Schneider, "Short-term motion prediction of traffic actors for autonomous driving using deep convolutional networks," 2018.

[18] J. Mercat, T. Gilles, N. El Zoghby, G. Sandou, D. Beauvois, and G. P. Gil, "Multi-head attention for multi-modal joint vehicle motion forecasting," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 9638–9644.

[19] M. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, "Argoverse: 3d tracking and forecasting with rich maps," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[20] L. Sun, X. Jia, and A. D. Dragan, "On complementing end-to-end human motion predictors with planning," *arXiv preprint arXiv:2103.05661*, 2021.