CCIL: CONTEXT-CONDITIONED IMITATION LEARNING FOR URBAN DRIVING

Anonymous authors
Paper under double-blind review

ABSTRACT

Imitation learning is a promising solution to the challenging autonomous urban driving task as experienced human drivers can effortlessly tackle highly complex driving scenarios. Behavior cloning is the most widely applied imitation learning approach in autonomous driving due to its exemption from potentially risky online interactions, but it suffers from the covariate shift issue. To mitigate this problem, we propose a context-conditioned imitation learning approach that learns a policy to map the context state into the ego vehicle's state instead of the typical formulation from both ego and context state to the ego action. Besides, to make full use of the spatial and temporal relations in the context to infer the ego future states, we design a novel policy network based on the Transformer, whose attention mechanism has demonstrated excellent performance in capturing relations. Finally, during evaluation, a linear quadratic controller is employed to produce smooth planning based on the predicted states from the policy network. Experiments on the real-world large-scale Lyft and nuPlan datasets demonstrate that our method can surpass the state-of-the-art method significantly.

1 INTRODUCTION

Planning a safe, comfortable, and efficient trajectory in a complex urban environment for a self-driving vehicle (SDV) is an important and challenging task in autonomous driving (Yurtsever et al., 2020). Unlike highway driving (Henaff et al., 2019), urban driving requires handling more varied road geometry such as roundabouts and intersections while interacting with traffic lights, pedestrians, and other vehicles. Classic manually-designed rule-based approaches (Fan et al., 2018) have achieved some success in industry but demand tedious human engineering to struggle with diverse real-world cases. Meanwhile, the rapid development of deep learning techniques motivates researchers (Bojarski et al., 2016; Pan et al., 2020) to employ a deep neural network to model the complicated driving policy. To learn such a policy, imitation learning (IL) from human drivers' demonstrations is a promising solution as experienced drivers can tackle even extremely challenging situations, and their driving data can be collected at scale.

The simplest IL algorithm is the behavior cloning (BC) method, which has wide applications in autonomous driving (Pomerleau, 1988; Bojarski et al., 2016; Codevilla et al., 2018b) due to its exemption from potentially dangerous online interactions. It learns a policy in a supervised fashion by minimizing the difference between the learner's action and the expert's action in the expert state distribution. However, the BC method suffers from the *covariate shift* issue (Ross et al., 2011), *i.e.* the state induced by the learner's policy cumulatively deviates from the expert's distribution.

To overcome the *covariate shift* obstacle, existing methods such as DAgger (Ross et al., 2011) and DART (Laskey et al., 2017) query supervisor corrections at the learner's states or perturbed expert's states. Since human supervision is hard to collect, recent works like GAIL (Ho & Ermon, 2016) seek to provide feedback from a neural network-based discriminator to recover from out-of-distribution states generated by the learner's policy. However, these data augmentation methods need either expert supervision or rolling out their policy in the real world or a realistic simulator, which are impractical in autonomous driving. On the contrary, some researchers attempt to constrain the learned policy formulation to ensure its robustness to the policy error by incorporating control theoretic prior knowledge. For example, Palan et al. (2020); Havens & Hu (2021) pose Kalman or linear matrix inequality constraints on the learned linear policy to guarantee the closed-loop stability in a linear

time-invariant (LTI) system. Yin et al. (2021) relaxes the policy formulation into a simple feed-forward neural network and East (2022) extends the method in Havens & Hu (2021) to polynomial system and policy. However, the urban driving task is too complex to be handled by these naive policy formulations.

To learn a stable and general policy for urban driving by imitating only the offline human demonstrations, we propose a context-conditioned imitation learning (CCIL) method, where we try to learn a policy to predict SDV future states using only its past local context, different from classic driving policy (Codevilla et al., 2019; Bansal et al., 2019) taking both SDV state and context as input to generate its action. The motivation is that the SDV state can be easily influenced by the policy's error, thus leading to the catastrophic distribution shift while map objects remain unchanged and human drivers in the context will try to recover from SDV's perturbation. Based on the stability assumption of the traffic system, we prove theoretically that our policy formulation can achieve closed-loop stability, thus addressing the distribution shift issue. However, without the SDV state input, it becomes challenging to predict its actions accurately. Therefore, we learn to infer SDV future states based on Transformer (Vaswani et al., 2017) which is capable of exploiting spatial and temporal information in highly interactive and constrained urban driving scenarios. Furthermore, during evaluation, we employ a linear-quadratic regulator (LQR) (Åström & Murray, 2021) to yield the SDV's action based on its current state and predicted states.

The main contributions of this paper can be summarized as follows:

1. To overcome the issue of distribution shift, we propose a novel offline imitation learning method, where we learn a policy to output the ego state using only its context as input. A robustness assurance is provided for our policy formulation based on an assumption of the context's stability.

2. To apply our method to urban driving, we remove the explicit ego state information input and propose a new ego-perturbed goal-oriented coordinate system to reduce the implicit ego information in the coordinate system. Besides, we design a Transformer-based network to make full use of the spatial and temporal information in the context.

3. To verify the effectiveness of our approach, we benchmark the real-world large-scale urban driving Lyft (Houston et al., 2020) and nuPlan (Caesar et al., 2021) datasets with state-of-the-art performance. The video and code can be found at `https://sites.google.com/view/contextconditionedil`.

2 Related Work

2.1 Imitation learning for autonomous driving

The objective of applying IL in autonomous driving is to learn a driving behavior mimicking human drivers. The most straightforward IL method is BC which minimizes the difference between the learner's action and the expert's action on the expert state without demanding extra manually labeled data and online interaction. Early BC applications in autonomous driving such as ALVINN (Pomerleau, 1988) and PilotNet (Bojarski et al., 2016) learn an end-to-end policy that directly maps sensor inputs to vehicle control commands using a large amount of human driving experience. Recently, ChauffeurNet (Bansal et al., 2019) instead provides intermediate planning using information from a perception system to improve generalization and transparency. However, the BC approach typically leads to the *covariate shift* (Ross et al., 2011) between the training distribution and deploying distribution, as the minor errors in the policy can lead to deviating from the expert state and then larger errors.

Methods to overcome the distribution shift obstacle can be categorized into online and offline methods. Online methods try to directly match the expert state-action distribution instead of matching the expert state conditioned action distribution in BC. For instance, the method in Zhang & Cho (2017) based on DAgger (Ross et al., 2011) queries supervisor actions at the state the learner visits and then adds the new data into the dataset, thus adjusting the expert state distribution to match the learner's state distribution. To get rid of the requirement of interactive expert, several methods like Wang et al. (2021) based on adversarial imitation learning (Ho & Ermon, 2016) use a discriminator to measure the difference between learner and expert's state-action distribution, which is used to calculate the reward of reinforcement learning (RL). By raising the policy's return using RL, the policy's state

distribution will get close to the expert distribution. However, the reinforcement loop in the method needs online interaction with the environment to generate the policy's state distribution, which is hard to apply in the safe-critical task like autonomous driving. Instead, we aim to match the context-conditioned ego state distribution that can be learned by supervised learning without interaction with the environment or access to information from the expert policy during training.

Several offline methods have been applied to alleviate the *covariate shift* issue. The most widely used model-free methods are based on DART (Laskey et al., 2017) by providing examples of how the expert policy recovers from error, which synthesizes the deviated state and recovering action by perturbing the data following some rules. In Codevilla et al. (2018b), they inject temporally correlated noise into the trajectory to simulate gradual drift away from the desired trajectory. Alternatively, ChauffeurNet (Bansal et al., 2019) adds a uniform perturbation to the current SDV location and fits a new smooth trajectory that brings the SDV back to the original target location. However, these rule-based trajectory augmentation methods are hard to cover the real motion distribution induced by the learner's policy and the policy is very likely to learn a propensity for perturbed driving. Our method also applies perturbation to the SDV's current position but it is used to mask the ego vehicle's information instead of data augmentation. Therefore, our method does not have the trajectory smoothing process during training. Rather than providing examples of how the expert policy recovers from error, we seek to endow a learned policy with the robustness properties directly.

The model-based IL methods aim at matching the simulated trajectory distribution with the expert trajectory distribution. They train a policy network to minimize a differentiable cost over trajectories rolling out in a differentiable learned (Henaff et al., 2019) or data-driven simulator (Scheel et al., 2022). In Amos et al. (2018), a dynamics model based Variational Auto-encoders (Kingma & Welling, 2013) is first learned from observational data. Then, they train the policy network to output actions that lead to a similar trajectory as the expert trajectory. Since the dynamics model is differentiable, the actions can receive gradients from multiple time steps ahead which can penalize actions that will lead to large diverges in the future even if the instantaneous divergence is small. Instead of learning a differentiable model, UrbanDriver (Scheel et al., 2022) constructs a differentiable simulator using vectorized representations of recorded perception data and High Definition (HD) maps, where new driving observations are calculated by a coordinate transformation based on the pose of controlled SDV and collected data. Then, they employ policy gradients to train a closed-loop policy by rolling it out in the simulator. However, the performance of the model-based approach is limited by the accuracy of the simulator's dynamics.

2.2 Imitation learning with robustness

Imitation learning is different from supervised learning by deploying the policy under dynamics, whose robustness is considered to be the ability of the learned policy to recover from policy errors. Some researchers have attempted to learn a policy with a stability guarantee using control-theoretic methods to apply policy constraints. In Taylor Series IL (Pfrommer et al., 2022), it is shown that the trajectory induced by the learner and expert will be close if the derivatives of the learner and expert policy match at the expert state. However, the high-order derivatives of the expert and learner policy are hard to compute without sufficient data. Others learn a robust linear (Palan et al., 2020; Havens & Hu, 2021) or simple feed-forward neural network (Yin et al., 2021) control policy for a linear dynamical system by posing constraints on the policy. Even though the authors in East (2022) extend the robustness guarantee to the polynomial systems and policy, obtaining guarantees on close-loop stability for the nonlinear autonomous driving system remains a challenge. Recently, the CMILe method (Tu et al., 2022) is capable of training nonlinear policies with the same safety guarantees as the expert but requires online expert access as DAgger. However, our method only constrains our policy in the formulation of receiving context state and producing ego state with a norm-constrained loss. Under a stability assumption of the environment subsystem, the stability of our policy is guaranteed.

3 Theoretical Analysis

We first introduce the notations and definitions used in this paper. For any vector $\boldsymbol{x} \in \mathbb{R}^n$, $\|\boldsymbol{x}\|_p$ stands for its L^p norm and $\|\boldsymbol{x}\|$ for L^2 norm. For any square matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\|\boldsymbol{A}\|$ denotes

its induced L^2 norm and $\rho(\boldsymbol{A})$ is its spectral radius (the maximum of the absolute values of its eigenvalues). For every induced matrix norm, we have $\rho(\boldsymbol{A}) \leqslant \|\boldsymbol{A}\|$.

Definition 1 (Comparison Functions) *A function $\gamma : \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ is class $\mathcal{K}$ if it is continuous, strictly increasing and satisfies $\gamma(0) = 0$. A function $\beta(x, t) : \mathbb{R}_{\geqslant 0} \times \mathbb{R}_{\geqslant 0} \to \mathbb{R}_{\geqslant 0}$ is class $\mathcal{KL}$ if it is continuous, $\beta(\cdot, t)$ is class $\mathcal{K}$ for each t and $\beta(x, \cdot)$ is decreasing for each x.*

Definition 2 (input-to-state stability (ISS) (Sontag, 2008)) *A discrete-time system $\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t), \boldsymbol{x}_0 = \boldsymbol{\xi}$ is input-to-state stable if there exists a class $\mathcal{KL}$ function β and a class $\mathcal{K}$ function γ such that, for each bounded input $\boldsymbol{u}$ and initial condition $\boldsymbol{\xi}$ and $t \in \mathbb{Z}_+$, it holds that:*

$$\|\boldsymbol{x}_t(\boldsymbol{\xi}, \boldsymbol{u})\| \leqslant \beta(\|\boldsymbol{\xi}\|, t) + \gamma(\|\boldsymbol{u}\|_\infty), \tag{1}$$

where $\|\boldsymbol{u}\|_\infty = \sup_{t \in \mathbb{Z}_+}(\boldsymbol{u}_t)$ is the sup norm of the input. If a system satisfies the ISS property, the distance between any two trajectories must eventually be bounded by its input signal and independent of initial conditions.

We consider performing imitation learning in the following nonlinear discrete-time system:

$$\boldsymbol{x}_{t+1} = f(\boldsymbol{x}_t, \boldsymbol{u}_t), \tag{2}$$

where $\boldsymbol{x}_t \in \mathbb{R}^n$ is the system state at time $t \in \mathbb{N}$ and $\boldsymbol{u}_t \in \mathbb{R}^u$ is the control input. For an autonomous driving system, its state can be separated into two parts $\boldsymbol{x}_t = (\boldsymbol{s}_t, \boldsymbol{c}_t)$: an SDV state $\boldsymbol{s}_t \in \mathbb{R}^m$ controlled by the control input $\boldsymbol{u}$ and context state $\boldsymbol{c}_t \in \mathbb{R}^{n-m}$ which is influenced by the SDV state. By viewing the policy error as a disturbance to the system, we apply linearization to the nonlinear system at the expert's state $\boldsymbol{x}^*$ to obtain the following linear subsystem:

$$\boldsymbol{c}_{t+1} = \boldsymbol{A}\boldsymbol{c}_t + \boldsymbol{B}\boldsymbol{s}_t, \tag{3}$$

with solution:

$$\boldsymbol{c}_{t+1} = \boldsymbol{A}^{t+1}\boldsymbol{c}_0 + \sum_{j=0}^{t} \boldsymbol{A}^{t-j}\boldsymbol{B}\boldsymbol{s}_j. \tag{4}$$

Here by an abuse of notation, we use the same symbol $\boldsymbol{c}_t$, $\boldsymbol{s}_t$ for the context deviation $\boldsymbol{c}_t - \boldsymbol{c}_t^*$ and SDV state deviation $\boldsymbol{s}_t - \boldsymbol{s}_t^*$.

Because the expert traffic system $\boldsymbol{c}_{t+1} = \boldsymbol{A}\boldsymbol{c}_t$ without the SDV state deviation is stable, we have $\rho(\boldsymbol{A}) < 1$. For such a Schur matrix, there are constants $c > 0$ and $0 \leqslant \sigma < 1$ such that $\|\boldsymbol{A}^t\| \leqslant c\sigma^t$ (Jiang & Wang, 2001). Besides, the context in traffic systems generally includes static map elements that are not influenced by SDV vehicles and intelligent human drivers who can quickly recover from the other vehicle's perturbation. It is natural to assume the subsystem in equation 3 to be input-to-state stable (ISS) with $\beta(r, t) < \sigma^{t-1}r$ and $\gamma(r) \leqslant \epsilon r$, where $\epsilon \geqslant 0$. Combining equation 1 and equation 4, we need

$$\beta(r, t) = c\sigma^t r < \sigma^{t-1}r, \quad \gamma(r) = \sum_{j=0}^{\infty} c\sigma^t \|\boldsymbol{B}\| r = \frac{c\|\boldsymbol{B}\|}{1-\sigma} r \leqslant \epsilon r, \tag{5}$$

which implies that $c < \frac{1}{\sigma}$, and $\|\boldsymbol{B}\| \leqslant \frac{\epsilon(1-\sigma)}{c}$.

Next, we study how the formulation of our policy can be stable under this ISS system. We learn a context state feedback control $\boldsymbol{s}_t = \boldsymbol{u}_t = f_{\boldsymbol{\theta}}(\boldsymbol{c}_t)$ which directly maps the context state to the SDV state. For simplicity, we consider a linear policy $\boldsymbol{u}_t = \boldsymbol{K}\boldsymbol{c}_t$. Then, the linear subsystem in equation 3 can be simplified as $\boldsymbol{c}_{t+1} = (\boldsymbol{A} + \boldsymbol{B}\boldsymbol{K})\boldsymbol{c}_t$ whose stable condition is $\rho(\boldsymbol{A} + \boldsymbol{B}\boldsymbol{K}) < 1$. Thus, we have if $\|\boldsymbol{K}\| < \frac{c(1-c\sigma)}{\epsilon(1-\sigma)}$, then

$$\rho(\boldsymbol{A} + \boldsymbol{B}\boldsymbol{K}) \leqslant \|\boldsymbol{A} + \boldsymbol{B}\boldsymbol{K}\| \leqslant \|\boldsymbol{A}\| + \|\boldsymbol{B}\|\|\boldsymbol{K}\| \leqslant c\sigma + \frac{\epsilon(1-\sigma)}{c}\|\boldsymbol{K}\| < 1. \tag{6}$$

This result shows that the stability of closed-loop autonomous driving can be guaranteed as long as the context system is stable enough and the norm of our policy is sufficiently small. In practice, we will add a L^2 norm regularization into the training loss of our neural network policy to internalize this stability prior.

Finally, we show how the naive BC policy formulation which takes both the SDV state and context state and generates the displacement from the current state can fail with even a fixed context $\boldsymbol{c}_t = 0$. We consider a linear policy: $\boldsymbol{u}_t = \boldsymbol{s}_t + \boldsymbol{K}_{bc}\boldsymbol{s}_t$ and update the state $\boldsymbol{s}_{t+1} = \boldsymbol{s}_t + \boldsymbol{u}_t$. Then, the dynamics is $\boldsymbol{s}_{t+1} = (\boldsymbol{I} + \boldsymbol{K}_{bc})\boldsymbol{s}_t$. Because $\|\boldsymbol{I}\| = 1$, we cannot obtain the stable condition $\rho(\boldsymbol{I} + \boldsymbol{K}_{bc}) < 1$ by limiting the $\|\boldsymbol{K}_{bc}\|$ as our policy formulation. Thus, the system's stability cannot be guaranteed even with a fixed context.

This simple analysis supports our choice to overlook the SDV history and current state in the learning to avoid causal confounding and distribution shift. Its advantage will also be validated in our experiments showing that this can significantly decrease the off-road rate.

4 METHOD

As shown in Figure 1, our architecture is composed of two modules: a neural planner which takes as input the environment state around the SDV to predict its future trajectories, and a LQR controller using the coarse planning to generate a smooth trajectory during testing.

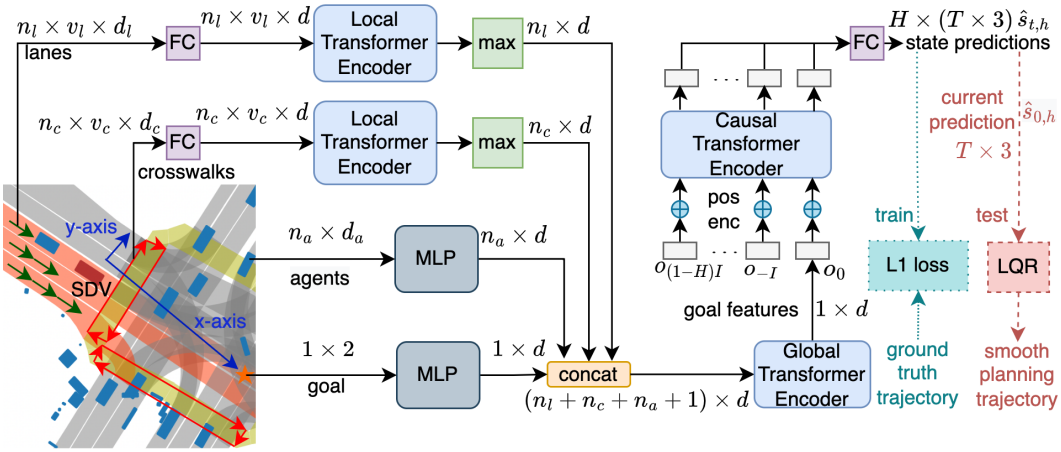

Figure 1: Overview of our approach

4.1 INPUT REPRESENTATION

As discussed above, we will overlook the SDV state to avoid distribution shift, keeping only the context as input on which our policy has limited influence. As shown in Figure 1, we consider four types of inputs to our neural planner: 1) a goal position: the mission goal that the SDV intends to reach; 2) n_a agents: each agent's d_a features composed of its types (*e.g.* car, bicycle, and cyclist), sizes, centroids and orientations of its bounding boxes at a few past time steps; 3) n_c crosswalks: each crosswalk polygon approximated by a sequence of v_c vectors as Gao et al. (2020), where each vector's d_c features consist of its starting and ending point coordinate, sequence orders; 4) n_l lanes: each lane polyline also approximated by v_l vectors whose d_l features additionally include its traffic light state, lane width, travel distances to left, right lane and goal from its starting point. The intermediate representations such as oriented bounding boxes from a perception system and road networks from HD maps rather than raw sensor data are harnessed to improve generalization and interpretability as Bansal et al. (2019).

Even though there is already no explicit SDV state input, the observations are typically translated to the SDV coordinate system using its rear axle's midpoint as origin and orientation as x-axis direction in previous works (Bojarski et al., 2016; Scheel et al., 2022). This implicit incorporation of the SDV state can still cause a distribution shift of inputs. To reduce the implicit influence and ensure local observability, we adopt an ego-perturbed goal-oriented coordinate system whose origin is the SDV position adding a 2D Gaussian perturbation, and x-axis direction is toward its goal position. Note that the perturbation is only required during training and the goal position can be replaced with any fixed point.

4.2 NEURAL PLANNER

Since it is more challenging to plan the SDV trajectory without its state, we need to make full use of its interactions with and constraints from the context to infer its future states. In practice, we construct our neural planner based on the Transformer (Vaswani et al., 2017) whose multi-head attention mechanism has demonstrated excellent performance in capturing relations.

As depicted in Figure 1, there are four Transformer encoders in our network. The two local Transformer encoders are employed to embed the map vectors and capture the relations between vectors belonging to the same polyline. Subsequently, we aggregate the features of vectors belonging to the same polyline using max-pooling to obtain polyline-level features. Besides, two multi-layer perceptions (MLP) are used to embed agents and the goal respectively. Then, to model the high-order interactions between the goal, agents, and map elements, we apply a global Transformer encoder whose goal embedding is output as observation features $\boldsymbol{o}_t$ at the observation time step t. To take the temporal relation into account, a Transformer encoder with a causal self-attention mask is employed to embed all context features at past H time steps with interval I. Finally, the causal Transformer's hidden state at each past time step h is embedded by a full-connected linear layer to output a prediction $\hat{\boldsymbol{s}}_{h,t}$ of the SDV's state $\boldsymbol{s}_{t-h}$ at each future time step $t = 1, \ldots, T$. For simplicity, we represent the SDV state with its rear axle's midpoint coordinate and orientation $\boldsymbol{s}_t = (x_t, y_t, \theta_t)$.

4.3 TRAINING PROCESS

We leverage supervised learning to train the neural planner to predict the expert future trajectories by minimizing the L^1 loss, as it is more correlated to driving performance compared with the mean square error (Codevilla et al., 2018a). To help the network converge and generalize, we introduce an auxiliary task of minimizing the L^1 error at all historical time steps of the Casual Transformer. In addition, we apply a squared L^2 norm regularization to the network parameters $\boldsymbol{\theta}$ according to equation 6. The final loss is:

$$L = \sum_{t=1}^{T} \left(\|\boldsymbol{s}_t - \hat{\boldsymbol{s}}_{0,t}\|_1 + \mu \sum_{h=1}^{H-1} \|\boldsymbol{s}_{t-h} - \hat{\boldsymbol{s}}_{h,t}\|_1 \right) + \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2, \tag{7}$$

where s_t is the ground-truth SDV state at time step t, μ is a small hyperparameter and λ is a regularization factor.

4.4 EVALUATION PROCESS

Even though the prediction without SDV state input is more stable, it has difficulty ensuring the smoothness of the planning from the current state. Prior works (Vitelli et al., 2022; Amos et al., 2018) usually add a differentiable kinematic layer into the policy network to generate physically feasible planning. However, the differentiable kinematic layer will bring in the current SDV state which is undesirable for learning a policy. Therefore, we choose to obtain a smooth trajectory during evaluation by a linear-quadratic regulator (LQR) (Åström & Murray, 2021) which can efficiently minimize the total commutative quadratic cost of a linear dynamic system. We consider a finite-horizon, discrete-time linear system described by:

$$\begin{bmatrix} \boldsymbol{p}_{t+1} \\ \dot{\boldsymbol{p}}_{t+1} \\ \ddot{\boldsymbol{p}}_{t+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{D} & \boldsymbol{D}^2 \\ 0 & \boldsymbol{I} & \boldsymbol{D} \\ 0 & 0 & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{p}_t \\ \dot{\boldsymbol{p}}_t \\ \ddot{\boldsymbol{p}}_t \end{bmatrix} + \begin{bmatrix} \boldsymbol{D}^2 \\ \boldsymbol{D} \\ \boldsymbol{I} \end{bmatrix} \boldsymbol{u}_t, \tag{8}$$

where $\boldsymbol{D}$ is a diagonal matrix with the time interval of each step as diagonal entries, $\dot{\boldsymbol{p}} = (\boldsymbol{v}_t, w_t), \ddot{\boldsymbol{p}}_t = (\boldsymbol{a}_t, \alpha_t), \boldsymbol{u}_t = (\boldsymbol{u}_t^p, u_t^a)$ denote the positional and angular velocity, acceleration, and control input respectively, subject to a quadratic cost function:

$$\mathcal{J} = \sum_{t=1}^{T} \left(\|\boldsymbol{p}_t - \hat{\boldsymbol{s}}_{0,t}\|_2^2 + w_w w_t^2 + w_a \|\boldsymbol{a}_t\|_2^2 + w_\alpha \alpha_t^2 + w_u ||\boldsymbol{u}_t^p||_2^2 \right), \tag{9}$$

where $\boldsymbol{p}_t$ is regarded as the planning trajectory and the predicted states $\hat{\boldsymbol{s}}_{0,t}$ at the current step from the neural planner are regarded as target poses; w_w, w_a, w_α and w_u are the weights to balance the safety and the smoothness of the planned trajectory.

5 EXPERIMENTS

5.1 DATASET

To benchmark our method's performance, we conduct experiments on two real-world large-scale datasets: **Lyft Level 5 Prediction Dataset** (Houston et al., 2020): contains about 1000h urban driving demonstrations in Palo Alto, which have been separated into independent scenes of nearly 25s at 10Hz. We train our network on the provided 100h (16265 scenes) subset as Scheel et al. (2022) and test with all 16220 validation scenes. **nuPlan Dataset** (Caesar et al., 2021) provides an urban driving dataset with 1312h of human driving data from 4 cities (Boston, Pittsburgh, Las Vegas, and Singapore). Due to the huge difference in traffic rules and patterns in different cities, we extract driving data in Las Vegas as Phan-Minh et al. (2022). Then, we separate the data into independent scenes of 25s at 10Hz like Lyft, and filter out the scene without a mission goal. After filtering, we obtain 63181 training, 4774 validation, and 6386 testing scenes. More details about the data prepossessing and corresponding model details are presented in the appendix.

5.2 CLOSED-LOOP EVALUATION

To evaluate the closed-loop performance of our method, we use a log-replay simulator as prior work (Scheel et al., 2022; Huang et al., 2022). At each step in the log-replay simulator, the SDV vehicle updates its state according to the planned trajectory, while the other agents are assumed to follow their recorded trajectories in the dataset. For both datasets, we evaluate our method for 25s at 10Hz with the following metrics:

Collision Rate: If the SDV collides with the other agents at any time step in a scene, the scene is deemed as a collision scene. The collision rate is the ratio of the collision scenes in all scenes.

Off-road Rate: In nuPlan, we use the official drivable area compliance metric, *i.e.* the distance of a corner of SDV's bounding box from the drivable area is more than 0.3m. But in Lyft, without access to the drivable area, if the SDV deviates laterally from the human driver's ground truth more than 2m like in a scene, the scene will be deemed off-road as Scheel et al. (2022).

Discomfort: we count the rate of the absolute value of the acceleration more than 3 $\mathrm{m/s^2}$ over time steps to quantify the comfort and feasibility of the planned trajectory.

L2: we use the average L2 position errors between the roll-out trajectory and the human driver's ground truth to quantify the human driving similarity.

5.3 PERFORMANCE EVALUATION

On the Lyft dataset, we compare our methods against three state-of-the-art methods to demonstrate the advantage of the proposed framework:

Raster-perturb: an official BC planning baseline, which based on ResNet50 receives a Bird-Eye-View (BEV) representation of the scene surrounding the SDV and produces a trajectory of position and yaw displacements. To augment data, a perturbation is applied to the current position, and then a new kinematically feasible trajectory to reach the original endpoint is generated as Chauffeur-Net (Bansal et al., 2019).

BC-perturb: a baseline BC model provided by Scheel et al. (2022) with the same trajectory perturbation and output as **Raster-perturb** but its inputs are presented in vector formulation, which are processed by a VectorNet (Gao et al., 2020) and Transformer.

UrbanDriver: an offline policy gradient method proposed by Scheel et al. (2022) to imitate the expert's policy exploiting a differentiable data-driven simulator with the same data and model structure as **BC-perturb**.

On nuPlan dataset, we compare our methods against three official baselines learned by BC due to a lack of other prior work:

Raster: a raster-based model that uses the ResNet-50 backbone to encode SDV, agent, and map information as raster layers to plan the SDV's trajectory.

Table 1: Comparison with baselines of 25s closed-loop performance on Lyft and nuPlan dataset

Model	Collision(%)	Off-road(%)	Discomfort(%)	L2(m)
Raster-perturb (Houston et al., 2020)	15.48	5.06	4.00	5.90
BC-perturb (Scheel et al., 2022)	9.38	6.77	39.10	4.77
UrbanDriver (Scheel et al., 2022)	13.28	7.27	39.41	5.74
CCIL (ours)	**3.30**	**0.59**	**3.53**	**0.97**
Raster (Caesar et al., 2021)	64.9	34.1	18.3	24.9
Vector-perturb (Caesar et al., 2021)	81.3	69.7	28.1	51.2
LaneGCN-perturb (Caesar et al., 2021)	63.3	34.9	18.1	23.2
CCIL (ours)	**6.89**	**3.01**	**1.38**	**3.67**

Vector-perturb: a simple vector-based model that consumes SDV, agent, and lane signals through a series of MLPs using perturbed trajectory input as ChauffeurNet (Bansal et al., 2019).

LaneGCN-perturb: a vector-based model that uses a series of MLPs to encode SDV and agent signals and one LaneGCN (Liang et al., 2020) to encode vector-map elements and a fusion network to capture lane and agent intra/inter-interactions through attention layers. To augment data, the SDV trajectories are perturbed and other agents are randomly dropped out.

Our performance is shown in Table 1. It demonstrates that our method can outperform previous work significantly on both datasets. The collision rate on the nuPlan dataset is higher than the Lyft dataset because there are much more road agents in its scene.

5.4 Ablation Study

The following ablation experiments on the Lyft dataset are used to expose the significance of different components of our model:

BC: Firstly, we consider training a model by BC using the SDV state without perturbation. In detail, we concatenate the SDV state including SDV's current position, velocity, and acceleration with the goal position and represent the observation in the classical SDV vehicle coordinate system. The result in Table 2 shows our CCIL outperforms the naive BC method by a large margin.

SDV input: To show the influence of *explicit SDV state input*, we concatenate the SDV state with the goal position, keeping other parts of our method unchanged. Then, we consider the influence of *implicit SDV state input* in the coordinate system, where our coordinate system is replaced with the classical coordinate system. Both ego information input ways will deteriorate the performance significantly (see Table 2).

Network structure: To demonstrate the effectiveness of the causal Transformer in capturing the temporal interactions, we replace it with a MLP. As is shown in Table 2, the temporal relations embedded by the causal Transformer can improve the performance.

Loss: To demonstrate the effectiveness of the additional term, we remove the auxiliary and regularization loss term separately. The result in Table 2 shows that they are both beneficial to improving performance.

Perturbation: To investigate how the implicit ego information affects the closed-loop performance, we adjust the Gaussian perturbation's standard deviation (std). We can see from Table 2 that with the increasing of the std and then less ego information, the closed-loop L2 decreases but the discomfort increases, which demonstrates that ego information can improve the instantaneous prediction accuracy but deteriorate the closed-loop performance.

5.5 Toy Experiment

We design a toy example to vividly show our method's ability to avoid the compounding error. The training data are trajectories on a circle of a random radius ranging from 10 to 100 with speed 1.

Table 2: Ablation experiments on 25s closed-loop performance on Lyft dataset

Model	Perturb	Ego	Collision(%)	Off-road(%)	Discomfort(%)	L2(m)
BC		✓	33.03	16.70	2.60	11.67
explicit ego input	✓	✓	28.16	23.4	**0.80**	6.23
implicit ego input	✓	✓	4.83	0.89	3.33	1.52
w/o causal Trans	✓		4.49	1.80	4.76	1.74
w/o regularization	✓		4.29	1.24	3.16	1.31
w/o auxiliary	✓		5.35	1.85	3.52	2.19
std=0 (no perturb)			4.39	1.88	1.01	2.97
std=1	✓		3.40	0.78	2.91	1.72
std=2 (ours)	✓		**3.30**	0.59	3.53	0.97
std=3	✓		3.65	**0.35**	5.96	**0.89**

The inputs of BC are composed of ego states (positions and orientations) and context (nearest circle point's coordinate) represented in the ego coordinate system in the past 10 steps. However, our CCIL method only receives the context in the perturbed-ego origin-oriented coordinate system. We train both models with 100 random initial weights and evaluate their closed-loop performance with a random starting point on a circle of radius 50. The closed-loop trajectories depicted in Figure 2 demonstrate that the BC method deviates from the expert state distribution but our method does not.

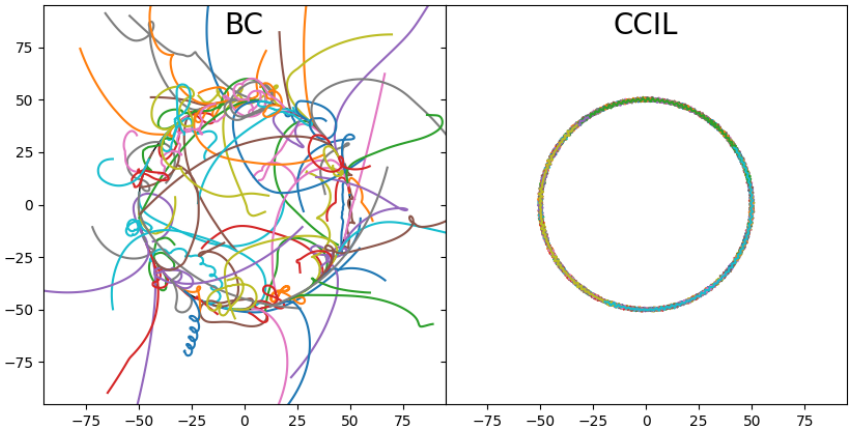

Figure 2: Closed-loop trajectories of BC and CCIL model trained on the toy dataset.

6 Conclusion

We have proposed a new offline imitation learning method to mitigate the distribution shift of behavior cloning, where we learn a neural network to predict the SDV's future positions and orientations without SDV information by making full use of its constraints from and interaction with the context. Firstly, we analyze theoretically the stability of our policy formulation. In addition to removing the explicit SDV information input to the network, we present a new SDV-perturbed goal-oriented coordinate system for representing represent the observation input to remove the implicit SDV information and ensure local observability. We design a Transformer-based network to make full use of the history information to handle the more challenging learning task. Finally, we demonstrate the effectiveness of our approach in two real-world large-scale datasets with state-of-the-art performance.

REFERENCES

Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable mpc for end-to-end planning and control. *NeurIPS*, 31, 2018.

Karl Johan Åström and Richard M Murray. *Feedback systems: an introduction for scientists and engineers*. Princeton University Press, 2021.

Mayank Bansal, Alex Krizhevsky, and Abhijit Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. In *RSS*, 2019.

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.

Holger Caesar, Juraj Kabzan, Kok Seang Tan, Whye Kit Fong, Eric Wolff, Alex Lang, Luke Fletcher, Oscar Beijbom, and Sammy Omari. Nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles. In *CVPR ADP3 workshop*, 2021.

Felipe Codevilla, Antonio M Lopez, Vladlen Koltun, and Alexey Dosovitskiy. On offline evaluation of vision-based driving models. In *ECCV*, pp. 236–251, 2018a.

Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *ICRA*, pp. 4693–4700, 2018b.

Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *CVPR*, pp. 9329–9338, 2019.

Sebastian East. Imitation learning of stabilizing policies for nonlinear systems. *European Journal of Control*, pp. 100678, 2022.

Haoyang Fan, Fan Zhu, Changchun Liu, Liangliang Zhang, Li Zhuang, Dong Li, Weicheng Zhu, Jiangtao Hu, Hongye Li, and Qi Kong. Baidu apollo em motion planner. *arXiv preprint arXiv:1807.08048*, 2018.

Jiyang Gao, Chen Sun, Hang Zhao, Yi Shen, Dragomir Anguelov, Congcong Li, and Cordelia Schmid. Vectornet: Encoding hd maps and agent dynamics from vectorized representation. In *CVPR*, pp. 11525–11533, 2020.

Aaron Havens and Bin Hu. On imitation learning of linear control policies: Enforcing stability and robustness constraints via lmi conditions. In *ACC*, pp. 882–887, 2021.

Mikael Henaff, Alfredo Canziani, and Yann LeCun. Model-predictive policy learning with uncertainty regularization for driving in dense traffic. In *ICLR*, 2019.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *NeurIPS*, 29, 2016.

John Houston, Guido Zuidhof, Luca Bergamini, Yawei Ye, Long Chen, Ashesh Jain, Sammy Omari, Vladimir Iglovikov, and Peter Ondruska. One thousand and one hours: Self-driving motion prediction dataset. In *CoRL*, 2020.

Zhiyu Huang, Haochen Liu, Jingda Wu, and Chen Lv. Differentiable integrated motion prediction and planning with learnable cost function for autonomous driving. *arXiv preprint arXiv:2207.10422*, 2022.

Zhong-Ping Jiang and Yuan Wang. Input-to-state stability for discrete-time nonlinear systems. *Automatica*, 37(6):857–869, 2001.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. Dart: Noise injection for robust imitation learning. In *CoRL*, pp. 143–156. PMLR, 2017.

Ming Liang, Bin Yang, Rui Hu, Yun Chen, Renjie Liao, Song Feng, and Raquel Urtasun. Learning lane graph representations for motion forecasting. In *ECCV*, pp. 541–556, 2020.

Malayandi Palan, Shane Barratt, Alex McCauley, Dorsa Sadigh, Vikas Sindhwani, and Stephen Boyd. Fitting a linear control policy to demonstrations with a kalman constraint. In *Learning for Dynamics and Control*, pp. 374–383. PMLR, 2020.

Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos A Theodorou, and Byron Boots. Imitation learning for agile autonomous driving. *The International Journal of Robotics Research*, 39(2-3):286–302, 2020.

Daniel Pfrommer, Thomas TCK Zhang, Stephen Tu, and Nikolai Matni. Tasil: Taylor series imitation learning. *arXiv preprint arXiv:2205.14812*, 2022.

Tung Phan-Minh, Forbes Howington, Ting-Sheng Chu, Sang Uk Lee, Momchil S Tomov, Nanxiang Li, Caglayan Dicle, Samuel Findler, Francisco Suarez-Ruiz, Robert Beaudoin, et al. Driving in real life with inverse reinforcement learning. *arXiv preprint arXiv:2206.03004*, 2022.

Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *NeurIPS*, 1, 1988.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, pp. 627–635, 2011.

Oliver Scheel, Luca Bergamini, Maciej Wolczyk, Błażej Osiński, and Peter Ondruska. Urban driver: Learning to drive from real-world demonstrations using policy gradients. In *CoRL*, pp. 718–728, 2022.

Eduardo D Sontag. Input to state stability: Basic concepts and results. In *Nonlinear and optimal control theory*, pp. 163–220. Springer, 2008.

Stephen Tu, Alexander Robey, Tingnan Zhang, and Nikolai Matni. On the sample complexity of stability constrained imitation learning. In *Learning for Dynamics and Control Conference*, pp. 180–191. PMLR, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pp. 5998–6008, 2017.

Matt Vitelli, Yan Chang, Yawei Ye, Ana Ferreira, Maciej Wołczyk, Błażej Osiński, Moritz Niendorf, Hugo Grimmett, Qiangui Huang, Ashesh Jain, et al. Safetynet: Safe planning for real-world self-driving vehicles using machine-learned policies. In *ICRA*, pp. 897–904, 2022.

Pin Wang, Dapeng Liu, Jiayu Chen, Hanhan Li, and Ching-Yao Chan. Decision making for autonomous driving via augmented adversarial inverse reinforcement learning. In *ICRA*, pp. 1036–1042. IEEE, 2021.

He Yin, Peter Seiler, Ming Jin, and Murat Arcak. Imitation learning with stability and safety guarantees. *IEEE Control Systems Letters*, 6:409–414, 2021.

Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE access*, 8:58443–58469, 2020.

Jiakai Zhang and Kyunghyun Cho. Query-efficient imitation learning for end-to-end simulated driving. In *AAAI*, pp. 2891–2897, 2017.

A MAP PREPOSSESSING

In Lyft, there are only two types of map elements: lane and crosswalk. In nuPlan, the map elements consist of lane, lane connector, intersection, stop line, crosswalk, walkway, and car park. We categorize them into polyline elements including lane and lane connector, and polygon elements including stop line, crosswalk, intersection, walkway, and car park.

For each polyline element, it is approximated by a sequence of vectors with an interval of 3 meters. For both datasets, we prepossess the polyline map into a graph to take advantage of the topological information. We connect each vector with its nearest left, right, and next vector if exists. The nearest left or right vector means the nearest vector of its reachable left or right polyline. And the distance between vectors is represented by the Euclidean distance between their starting points. After building the vector graph, we can compute the travel distance between any two vectors using Dijkstra's algorithm. For both datasets, the polyline vector has shared features including coordinates of starting and ending point, distance to left and right vector, distance to the mission goal, traffic light states (red or green), and the sequence order. However, lane vectors in Lyft have a lane width feature because the vectors approximate computed middle lines of lanes, while the lane or lane connector vectors in nuPlan have a lane left and width feature because the vectors approximate an annotated baseline. In addition, the polyline vectors in nuPlan have additional speed limit and type features.

For each polygon element, we approximate it also with a sequence of vectors. In the Lyft dataset, the crosswalk vectors are directly constructed by connecting the original sequential annotation coordinates. But we approximate each polygon with fixed 20 vectors because the annotation point number is too large for elements like intersection. For both datasets, each polygon vector has features including coordinates of starting and ending points, and its order sequence, while vectors in nuPlan have additional type features.

The inputs to our neural network are composed of two types of map elements: polyline and polygon, and two types of agent elements: other agents and ego goal. The missing inputs are padded with zeros and masked out when calculating the attention. The origin is the perturbed SDV position.

Polyline: 30 topologically nearest polylines with vectors whose starting points are within 35m from the origin. The topological distance between the origin and a polyline is the minimum of topological distances between the origin and its vectors.

Polygon: 20 polygons whose boundaries are within 35m from the origin. If there are more than 20 polygons, they are selected according to the importance of their types: stop line, crosswalk, intersection, walkway, and car park.

Agents: the nearest 30 agents whose oriented boxes' centroids are within 50m from the origin. The agent features in Lyft include its centroid coordinates, yaws, shapes, types, and relative times in the past 2 and current steps. The nuPlan agents additionally have velocity features as they are provided.

Goal: the x, y coordinates of the SDV's mission goal of a scene. In Lyft, the mission goal is not provided, so we regard the ending point of the lane where the SDV locates at the last time step of the scene as the mission goal. In nuPlan, we directly take the provided mission goal at the last time step of the scene as the scene mission goal.

B MODEL

For both datasets. the same model architecture is used, whose hyper-parameters are listed in Table 3. Our model is trained using the Adam optimizer with a learning rate of 0.0005, 10000 steps linear warm-up, $\beta = (0.9, 0.999)$, and batch size 128.

C EVALUATION

For Lyft baselines, we directly evaluate the pretrained model provided by the Lyft dataset (Houston et al., 2020) and UrbanDriver (Scheel et al., 2022). We evaluate all Lyft models from the second time step to compute current velocity using position information as input to the LQR with zero assumed initial acceleration.

For nuPlan baselines, we train the official models by ourselves using provided hyper-parameters and evaluate from the first time step because the velocity and acceleration information is provided.

Hyper-parameter	value
Future steps T	15
All Transformers dropout	0.1
All Transformers head number	8
All Transformers hidden size	256
Local Transformer layer number	3
Global Transformer layer number	6
Causal Transformer layer number	3
Causal Transformer length H	15
Causal Transformer interval I	2
Auxiliary weight μ	0.3
Regularization weight λ	0.0001

Table 3: Hyper-parameters for both datasets