Vehicle Trajectory Prediction Using Generative Adversarial Network With Temporal Logic Syntax Tree Features

Xiao Li, Guy Rosman, Igor Gilitschenski, Cristian-Ioan Vasile, Jonathan A. DeCastro, Sertac Karaman, and Daniela Rus

***Abstract*—In this work, we propose a novel approach for integrating rules into traffic agent trajectory prediction. Consideration of rules is important for understanding how people behave—yet, it cannot be assumed that rules are always followed. To address this challenge, we evaluate different approaches of integrating rules as inductive biases into deep learning-based prediction models. We propose a framework based on generative adversarial networks that uses tools from formal methods, namely signal temporal logic and syntax trees. This allows us to leverage information on rule obedience as features in neural networks and improves prediction accuracy without biasing towards lawful behavior. We evaluate our method on a real-world driving dataset and show improvement in performance over off-the-shelf predictors.**

***Index Terms*—Autonomous-driving, prediction, temporal logic.**

I. Introduction

PRIORS and structure have received increasing attention as elements of recent successful prediction models [1]. By designing structure/inductive biases into the model, sample efficiency and explainability of the model can be considerably improved. Useful priors include interaction with road agents [2]–[4], vehicle dynamics [5] and structure such as multi-modality [6]–[9]. While rule-based priors such as traffic rules and driving best practices are commonly incorporated into ego-vehicle planning, they are much less thoroughly explored in the prediction literature.

Vigorous definition of complex rules can be difficult. While some rules are simple (e.g., "don't hit the car in front of you," "don't cut in recklessly" [10]), traffic rules in general are more involved. For example, the 2020 Illinois rules of the road [11] describes a flagger sign as a warning for drivers that a flagger is ahead. It mandates that a *driver should use caution when approaching a flagger as the individual will be working close to traffic*. It also requires that the *driver should slow down and be prepared to obey the signals of the flagger* including being prepared to *stop if signaled to do so.* It is here where *compositional* formulations of rules, i.e. the ability to combine simple individual statements into rich behavioral expressions, can greatly simplify their definition. Without such a structure, maintaining a large set of rules and have them interact coherently at scale poses a challenge. We will exploit the structure and *discriminative* nature (define what is considered bad driving, instead of which trajectory should the driver take) of rules in trajectory prediction.

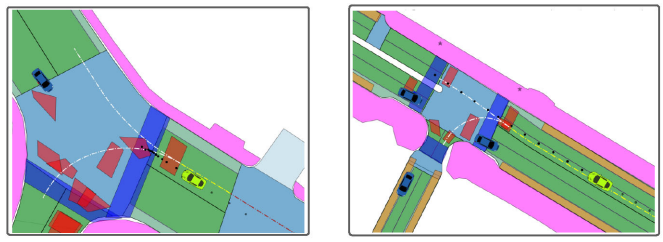

Fig. 1. **Typical Rule Violation.** While the rule "slow down when approaching pedestrian crossings" is usually satisfied (**a**), its violation (**b**) is still common. In this example, from the NuScenes dataset [13], the vehicle to be predicted is highlighted in green. The black dotted trajectory shows their ground-truth trajectory. The red patches represent stop areas (stop sign, pedestrian crosswalk, etc).

While traffic rules provide a strong prior on behaviour, they cannot be used as a hard constraint in prediction approaches because in certain situations it is common to disobey them. This is in contrast to planning where it is much more meaningful to incorporate rules as objectives or constraints. Fig. 1 illustrates an example where both vehicles to be predicted (in green) are approaching a pedestrian crossing. Even though it is desirable to "slow down near pedestrian crossings," their ground-truth futures (black dotted trajectory) show that one indeed slows down (Fig. 1(a)) but the other did not (Fig. 1(b)). This is different than the situation in planning for the ego-vehicle, where we can constrain ourselves to trajectories that enforce the satisfaction of the rules [12].

In this work, we address this problem by using signal temporal logic (STL) [14] as an expressive formalism to encode

Manuscript received October 15, 2020; accepted February 11, 2021. Date of publication March 1, 2021; date of current version March 23, 2021. This letter was recommended for publication by Associate Editor P. Tokekar and Editor D. Popa upon evaluation of the reviewers' comments. This work has been supported by the Toyota Research Institute (TRI). *(Corresponding author: Xiao Li.)*

Xiao Li, Igor Gilitschenski, and Daniela Rus are with the Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, United States of America (e-mail: xiaoli@mit.edu; igilitschenski@mit.edu; rus@csail.mit.edu).

Guy Rosman and Jonathan A. DeCastro are with the Computer Science and Artificial Intelligence Lab, Toyota Research Institute, Cambridge 02139, United States of America (e-mail: rosman@csail.mit.edu; jad455@cornell.edu).

Cristian-Ioan Vasile is with the Department of Mechanical Engineering and Mechanics, Lehigh University, Bethlehem, PA 18015, United States of America (e-mail: cvasile@lehigh.edu).

Sertac Karaman is with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139, United States of America (e-mail: sertac@mit.edu).

Digital Object Identifier 10.1109/LRA.2021.3062807

rules. For a given set of STL formulas, a syntax tree of their conjunction is constructed. This is used to extract features that represent different stages of rule satisfaction (further discussed in Section III-A). We observe that judging the usefulness of rules is similar to a discriminator in a GAN framework as it relates to how people use rules to reason about the plausibility of events. Thus, we use a discriminator to incorporate the extracted rule features into off-the-shelf trajectory predictors to make them rule-aware. Specifically, our contributions include (1) introducing features constructed from the sub-formulas of an STL syntax tree (which we refer to as syntax tree features) as a means to integrate temporal logic rules into existing trajectory predictors; (2) evaluation of different approaches to incorporate rules in a GAN-like prediction model. Specifically, we look at STL as a set of discriminator features as well as a generator auxiliary loss; (3) evaluating our architecture on a real world driving dataset.

II. Related Work

Our work relates to several active topics of research. Recent advances in vehicle trajectory prediction have explored the representation of multi-agent interactions [2]–[4], [15]–[17], uncertainty representation [2], [5], [9], [15], and different modality representations [16], [18]

Utilizing rules to constrain motion of dynamic systems is more commonly seen in trajectory planning of the ego vehicle. Recent work that use temporal logic to learn control policies in an RL setting include [19]–[21] whereas the authors of [22] use a differentiable linear temporal logic (LTL) loss in to learn policies from demonstrations. In [23], the authors transformed the syntax tree of an STL formula into a computation graph that can be integrated into existing deep architectures which we have adopted in this work. The authors of [24] introduce the RuleBook as a pre-ordered set of rules designed to help maintain traffic rules and driving heuristics in a explainable and prioritized manner. In [25], the authors use LTL to integrate rules of the road into route and trajectory planning. The authors of [26] use STL to specify safety contracts for evaluation of self-driving systems. In the more general context of planning, inference of active constraints can serve as a basis for more efficient learning from demonstration [27]. In prediction, rules are more commonly incorporated as feature maps [28] and cost-maps [29]. However, it is difficult to control the level of rule enforcement in the feature map approaches as it is determined by the features the network has extracted. The costmap approach utilizes a MPC (model predictive control)-like optimization rollout to enforce the costs onto the predictions. Multiple costmaps are combined using a weighted sum. The weights can have a significant impact on the meaning of the rules and hence the predictor's performance. Weight tuning can be difficult when the number of costmaps scales. In comparison, the STL rules we use follow a set of rigorous syntax and semantics. Our method uses the syntax tree of STL formulas to compute feature vectors that encode explicitly the satisfaction level at different stages of the rules at different time-steps. We introduce tree-level and node-level dropouts to control the influence of the rules on the predictions. We are able to make the prediction model rule-aware without explicitly enforcing predictions to follow the rules (which in certain cases will neglect rule-violating behaviors of road agents). The structure our method improves the predictor's explainability (by monitoring the satisfaction of the sub-formulas of the syntax tree with respect to the predicted trajectories).

III. Background

A. Signal Temporal Logic (STL)

STL offers a formalism for expressing and reasoning about rules for cars to follow. Properties expressed in STL capture rich car behaviors with timing constraints. The syntax of STL formulas is $\phi ::= p(s) > 0 \mid \neg\phi \mid \phi \vee \phi \mid \mathcal{F}_I\phi$, where $I \subset \mathbb{R}_{\geq 0}$ is a bounded time interval, $p(s) > 0$ ($p : \mathbb{R}^n \to \mathbb{R}$) is a predicate over state $s \in S \subseteq \mathbb{R}^n$, $\neg$ (not) and $\wedge$ (and) are Boolean operators, and $\mathcal{F}$ is the *eventually* operator. It can also involve other Boolean operators (e.g., $\vee$, $\Rightarrow$) and the *always* operator $\mathcal{G}_I$ which are defined in the usual way [14].

Let $s_{t_0:t_1} = (s_{t_0}, \ldots, s_{t_1})$ denote the discrete-time trajectory from t_0 to t_1. The finite-horizon trajectory $s_{t_0:t_1}$ satisfies ϕ, denoted by $s_{t_0:t_1} \models \phi$, if

$$
\begin{aligned}
s_{t_0:t_1} &\models (p(s) > 0) && \Leftrightarrow p\left(s_{t_0}\right) > 0 \\
s_{t_0:t_1} &\models \neg\phi && \Leftrightarrow \neg(s_{t_0:t_1} \models \phi) \\
s_{t_0:t_1} &\models \phi_1 \wedge \phi_2 && \Leftrightarrow (s_{t_0:t_1} \models \phi_1) \wedge (s_{t_0:t_1} \models \phi_2) \\
s_{t_0:t_1} &\models \mathcal{F}_I\phi && \Leftrightarrow \exists t' \in t_0 + I, t' < t_1, s_{t':t_1} \models \phi
\end{aligned}
$$

For example, STL formula $\phi_{ex} = \mathcal{G}_{[t_1,t_2]}(u > \epsilon_1) \wedge \mathcal{F}_{[t_3,t_4]}(v < \epsilon_2)$ means that "$u_t > \epsilon_1$ is true for all of $t \in [t_1, t_2]$ and $v_t < \epsilon_2$ holds for at least one of $t \in [t_3, t_4]$".

STL admits quantitative semantics called robustness that assigns degrees (real numbers) of satisfaction or violation to trajectories with respect to formulas. It is defined by

$$
\begin{aligned}
r(s_{t_0:t_1}, p(s) > 0) &= p\left(s_{t_0}\right) \\
r(s_{t_0:t_1}, \neg\phi) &= -r(s_{t_0:t_1}, \phi) \\
r\left(s_{t_0:t_1}, \phi_1 \wedge \phi_2\right) &= \min(r(s_{t_0:t_1}, \phi_1), r(s_{t_0:t_1}, \phi_2)) \\
r\left(s_{t_0:t_1}, \mathcal{F}_I\phi\right) &= \max_{t' \in (t_0+I) \cap [t_0,t_1)} r\left(s_{t':t_1}, \phi\right) \qquad (1)
\end{aligned}
$$

where $\max$ over an empty set is $-\infty$.

A robustness greater than zero signifies that the trajectory satisfies the given formula, i.e. $r(s_{t_0,t_1}, \phi) > 0 \Rightarrow s_{t_0,t_1} \models \phi$. Negative robustness implies violation of the formula. We define the robustness trace as $\tilde{r}(s_{t_0:t_1}, \phi) = [r(s_{t_0:t_1}, \phi), r(s_{t_0+1:t_1}, \phi), \ldots, r(s_{t_1-1:t_1}, \phi)]$.

For any STL formula ϕ, there is an *abstract syntax tree* $\mathcal{T}_\phi = \{\mathcal{N}_\phi, \mathcal{E}_\phi\}$, where intermediate nodes N_ϕ correspond to Boolean and temporal operators, leaf nodes correspond to predicates, and edges $\mathcal{E}_\phi$ connect operators to their operands. We identify a node $n \in \mathcal{N}_\phi$ with the associated sub-formula ϕ_n as shown in Fig. 2 (solid arrows) for formula ϕ_{ex}.

The robustness (1) can be computed by traversing the syntax tree. As noted in [30], the recursive computation can be thought of as a computation graph from predicate states to the root node. Fig. 2 shows the edges of the computation graph as dash arrows, and input nodes in blue. We take advantage of this computation graph to construct features for our prediction models.

B. Generative Adversarial Network (GAN)

A Generative Adversarial Network consists of two neural networks trained adversarially to each other [31]. A generative

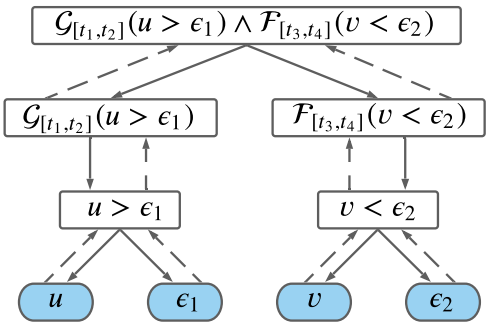

Fig. 2. **Syntax tree** for STL formula $\mathcal{G}_{[t_1,t_2]}(u > \epsilon_1) \wedge \mathcal{F}_{[t_3,t_4]}(v < \epsilon_2)$ which denotes "$u_t > \epsilon_1$ is true for all of $t \in [t_1, t_2]$ and $v_t < \epsilon_2$ holds for at least one of $t \in [t_3, t_4]$". Blue indicates input nodes.

model G that takes in a source distribution and outputs samples of the target distribution, and a discriminative model D that takes in a sample and estimates the probability that it comes from the target distribution (as oppose to being constructed by the generator). The generator takes a latent variable z as input (often from a well-known distribution such as uniform or Gaussian distribution), and outputs sample $s = G(z)$. The discriminator D takes in a sample s and outputs $D(s)$ which represents the probability that it is from the target distribution. The training process mimics a two-player min-max game with the objective function:

$$\begin{aligned} &\min_G \max_D \mathcal{L}(G, D) \\ &= \mathbb{E}_{s \sim p_{\text{data}}(s)}[\log D(s)] + \mathbb{E}_{z \sim p_{(z)}}[\log(1 - D(G(z)))]. \end{aligned} \tag{2}$$

GANs can used for conditional models by providing both the generator and discriminator with additional input $\mathcal{X}$, yielding $G(\mathcal{X}, z)$ and $D(\mathcal{X}, s)$ [2].

IV. STL-GAN

Let $\mathcal{X}$ be the input features (semantic maps, state vectors, etc) to a given prediction model, $s_{0:T}$ be the ground truth future trajectory of the agent to be predicted (s here is the 2D coordinate and steering, T is the prediction horizon). Denote $\tilde{s}_{0:T}$ and $\tilde{a}_{0:T}$ to be the predicted trajectory and controls generated by the model. We state the prediction problem as: *given $\mathcal{X}$ generate samples from the distribution $P(s_{0:T}|\mathcal{X})$* - the distribution over trajectory $s_{0:T}$ given input features $\mathcal{X}$.

Let us look at an STL formula ϕ and $s_{0:T} \models \phi$ describing whether the trajectory obeys the driving rules. We also consider the sub-formulas of ϕ given by its syntax tree. While computing ϕ does not generate new information, ϕ is an informative and compact representation for many of the samples in the dataset. This poses the question: "what is a good way to leverage ϕ as a representation, so that we may capture $P(s_{0:T}|\mathcal{X})$ more accurately?"

In this section, we introduce two approaches of integrating STL rules into trajectory prediction models. Fig. 3 illustrates our proposed architecture which consists of 3 components — a generator, a discriminator and a set of syntax tree features. Each of the components will be discussed in detail in the following sections. It is worth noting that even though our method is GAN-based, one can replace the base predictor with any trajectory generator provided that it can be made stochastic and able to cover the space of trajectories. In our case, we added a random variable to the input of the base predictor (the multimodal trajectory predictor [32]) such that samples of the random variable map to samples of the prediction. Given that the discriminator serves to incorporate information from the rules, the generator needs to be trained in conjunction with the discriminator (can not be pretrained with frozen weights).

Generator: We now describe our generator. We chose a generator based on dynamics integration rather than direct trajectory emission similar to [5] as the resulting control signals make it easier to consistently define rules. Given observation $\mathcal{X}$, noise feature z drawn from a uniform or normal distribution and a base predictor $\texttt{BP}$, a sequence of predicted controls are generated by $\tilde{a}_{0:T} = \texttt{BP}(\mathcal{X}, z)$.

Here we define controls at time t as $\tilde{a}_t = (v_t, \omega_t)$ where v_t, ω_t are the speed and steering rate respectively. The state trajectory is calculated using the unicycle model

$$[\dot{x},\, \dot{y},\, \dot{\psi}]^\top = \mathcal{V}(s, a) = [v\cos(\psi),\, v\sin(\psi),\, \omega]^\top. \tag{3}$$

Let $s = (x, y, \psi)$ be the 2D coordinates and heading respectively. Given the agent's current state s_0 (assumed to be part of input features $\mathcal{X}$), controls $a_{0:T}$ and (3), future trajectory can be calculated by $s_{0:T} = \texttt{Dyn}(s_0, a_{0:T})$ where $s_{t+1} = s_t + \Delta t \cdot \mathcal{V}(s_t, a_t)$, and Δt is the time interval. This way of predicting first the controls then the state trajectory using a dynamics model not only allows us to generate dynamically feasible trajectories, but also gives us access to the controls which will be useful in defining the STL rules. The generator takes the form $\tilde{s}_{0:T} = G(\mathcal{X}, z) = \texttt{Dyn}(s_0, \texttt{BP}(\mathcal{X}, z))$.

The generator loss is defined as

$$\begin{aligned} \mathcal{L}_G = \mathbb{E}_{\substack{z \sim \mathcal{N}(0,1) \\ \mathcal{X} \sim p_{data}(\mathcal{X})}} &[\log(1 - D(\mathcal{X}, G(\mathcal{X}, z)))] \\ &+ w_{MoN}\mathcal{L}_{MoN} + w_{BP}\mathcal{L}_{BP} \end{aligned} \tag{4}$$

where w_{MoN}, w_{BP} are scalar weights, $\mathcal{L}_{BP}$ is the loss of the base predictor, and $\mathcal{L}_{MoN}$ is defined as

$$\mathcal{L}_{\text{MoN}} = \min_k \left\| s_{0:T} - \tilde{s}^k_{0:T} \right\|_2. \tag{5}$$

Here $\tilde{s}^k_{0:T}$ denotes the k^{th} trajectory sampled from the generator (by sampling multiple noise terms for a given $\mathcal{X}$). Similar to [2], adding such a loss encourages the generator to produce diverse samples to increase the chance of covering the ground-truth.

Discriminator. The discriminator takes input $\mathcal{X}$ and constructs a feature vector $f_{\mathcal{X}}$ using a feature extractor (for example a CNN if $\mathcal{X}$ contains images). $f_{\mathcal{X}}$ along with the best predicted trajectory from the generator $\tilde{s}_{0:T}$ are passed to an LSTM encoder ($\tilde{s}_{0:T}$ as input, $f_{\mathcal{X}}$ used to initialize the hidden state) to generate feature f_{enc}. f_{enc} can be passed directly to an MLP to obtain a classification score.

The discriminator loss is defined as

$$\begin{aligned} \mathcal{L}_D = -\Bigg[&\mathbb{E}_{\substack{\mathcal{X} \sim p_{data}(\mathcal{X}) \\ s \sim p_{data}(s)}}[\log D(\mathcal{X}, s)] \\ &\mathbb{E}_{\substack{z \sim \mathcal{N}(0,1) \\ \mathcal{X} \sim p_{data}\mathcal{X}}}[\log(1 - D(\mathcal{X}, G(\mathcal{X}, z)))] \Bigg] \end{aligned} \tag{6}$$

STL Robustness As The Generator's Auxiliary Loss. We introduce a simple approach to integrate STL rules with the generator. We add the weighted absolute value of the robustness as an additional term to the generator loss in Equation (4). The

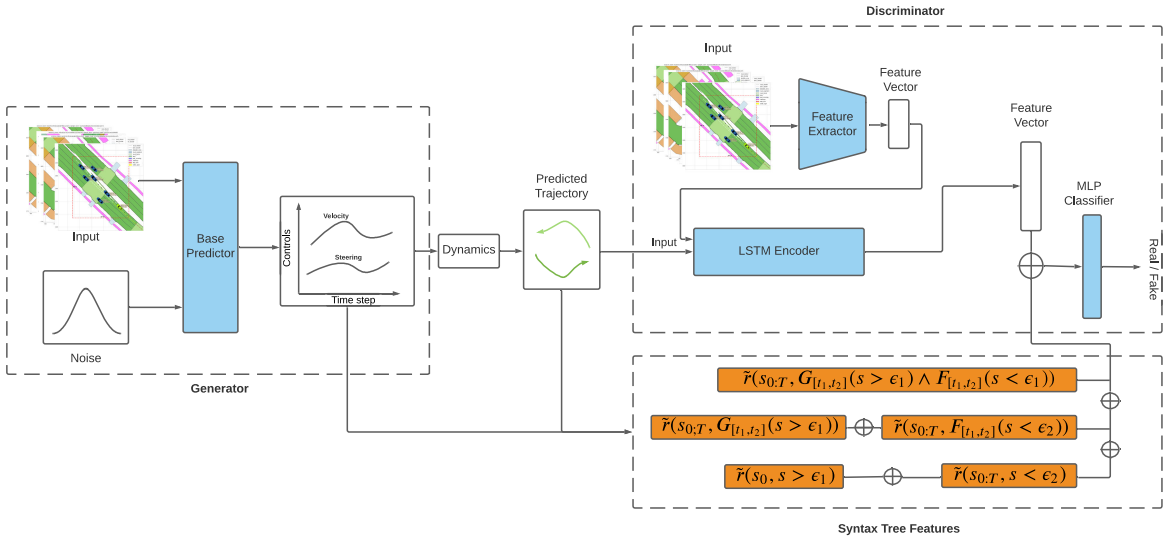

Fig. 3. **Architecture for STL-GAN**. The generator can be constructed with off-the-shelf trajectory predictors by injecting noise to its inputs. Syntax tree features are used to enhance the capacity of the discriminator and/or in training costs for the generator.

new loss is defined as

$$\bar{\mathcal{L}}_G = \mathcal{L}_G - w_r \min\{0, r((\tilde{s}_{0:T}, \tilde{a}_{0:T}), \phi)\}. \tag{7}$$

Note that here the robustness takes also the controls as input arguments, this allows us to design rules such "if pedestrian crossing then slow down". Minimizing this loss encourages the generator to output trajectories that comply with ϕ (robustness loss is non-zero if the generated trajectory is rule-violating and zero otherwise). One caveat to this approach is that depending on w_r, the generator can become overly biased towards the rules and neglect realistic but rule-violating behaviors.

STL Syntax Tree as Discriminator Features. In order to avoid biasing predicted trajectories into lawful behaviors, and have the predictor use the rules as cues, we propose an alternative approach. We use the robustness term of all nodes in the syntax tree as discriminator features. Given a syntax tree $\mathcal{T}_\phi = \{\mathcal{N}_\phi, \mathcal{E}_\phi\}$, for each node sub-formula $\phi_n, n \in \mathcal{N}_\phi$, we define a node feature as the robustness trace of that node sub-formula

$$f_{\phi_n} = \tilde{r}((\tilde{s}_{0:T}, \tilde{a}_{0:T}), \phi_n). \tag{8}$$

In order to encourage the network to leverage each sub-expression rather than counting on only the most prominent ones, at training time we apply dropout to the different node features, as is often done in multimodal network fusion [7]:

$$f_\phi = d_\mathcal{T} \cdot \underset{n \in \mathcal{N}_\phi}{\texttt{Concat}}(d_n \cdot f_{\phi_n}), \tag{9}$$

where $d_n, d_\mathcal{T} \sim Bernoulli(p)$ are Bernoulli random variables with probably of 1 being $p_n, p_\mathcal{T}$ respectively. p_n and $p_\mathcal{T}$ represent the dropout probabilities for the syntax tree node features and the entire tree respectively (which determine the probabilities that $d_n, d_\mathcal{T}$ take values 1 or 0). $d_n, d_\mathcal{T}$ are added during training and set to 1 during evaluation. f_ϕ is concatenated with f_{enc} and passed as input to the MLP classifier. Algorithm 1 provides an overview of our method.

In Algorithm 1, the gradient descent steps (lines 7, 9, 16, 18) can be realized using any stochastic gradient optimizers.

Algorithm 1: STL-GAN.

1: **Inputs**: STL formula ϕ; generator $G(\mathcal{X}, z|\theta_G)$ parameterized by θ_G; discriminator $D(\mathcal{X}, s_{0:T}|\theta_D)$ parameterized by θ_D; number of iterations N; learning rate α; number of sample trajectories to draw from generator K; syntax tree feature dropout probabilities $p_n, p_\mathcal{T}$.
2: $\mathcal{N}_\phi, \mathcal{E}_\phi \leftarrow \texttt{ConstructSyntaxTree}(\phi)$
3: **for** i=0 ...N-1
4: Sample minibatch of m noise $\boldsymbol{z} = \{z^0, .., z^{m-1}\}$ from normal distribution $\mathcal{N}(0, 1)$.
5: Sample minibatch of m data samples $\boldsymbol{X}, \boldsymbol{s}_{0:T} = (\mathcal{X}^0, \ldots, \mathcal{X}^{m-1}), (s^0_{0:T}, \ldots, s^{m-1}_{0:T})$
6: **if** STL generator loss **then**
7: $\theta_G \leftarrow \theta_G - \alpha \nabla_{\theta_G} \frac{1}{m} \sum_j^m \bar{\mathcal{L}}_G(\mathcal{X}^j, z^j, \phi)$
8: **else**
9: $\theta_G \leftarrow \theta_G - \alpha \nabla_{\theta_G} \frac{1}{m} \sum_j^m \mathcal{L}_G(\mathcal{X}^j, z^j)$
10: **end if**
11: sample $K \times m$ noise terms $\bar{\boldsymbol{z}}$ from $\mathcal{N}(0, 1)$
12: $\tilde{\boldsymbol{a}}_{0:T}, \tilde{\boldsymbol{s}}_{0:T} = \texttt{GetMoNTrajectory}(\theta_G, \boldsymbol{X}, \boldsymbol{s}_{0:T}, \bar{\boldsymbol{z}})$
13: if STL syntax tree features **then**
14: $f_\phi \leftarrow \texttt{GetFeature}(\tilde{\boldsymbol{a}}_{0:T}, \tilde{\boldsymbol{s}}_{0:T}, \mathcal{N}_\phi, p_n, p_\mathcal{T})$
15: $\theta_D \leftarrow \theta_D - \alpha \nabla_{\theta_D} \frac{1}{m} \sum_j^m \mathcal{L}_D(\tilde{\mathcal{X}}^j, \boldsymbol{s}^j_{0:T}|f_\phi)$
16: **else**
17: $\theta_D \leftarrow \theta_D - \alpha \nabla_{\theta_D} \frac{1}{m} \sum_j^m \mathcal{L}_D(\tilde{\mathcal{X}}^j, \boldsymbol{s}^j_{0:T})$
18: **end if**
19: **end for**

The function `GetMoNTrajectory` on line 12 samples K trajectories from the generator (for each element in the batch) and returns the prediction closest to the ground truth. In many GAN related predictors, randomly generated trajectories are fed to the discriminator. In practice we found that feeding the MoN trajectory to the discriminator produces better results when the syntax tree features are used.

V. EXPERIMENTS

Dataset. We use the NuScenes dataset [13] for training and evaluation. The dataset contains 1000 scenes of 20 s each collected in Boston and Singapore. It also includes rich semantic information including 23 object classes (pedestrian, vehicle, etc) and HD maps with 11 annotated layers (lanes, walkways, etc). Our goal is to show that using these semantic information we can define rules that will improve the performance of existing predictors.

Rules Used. We define the following rule

$$\begin{aligned}\phi = \ & \left(\mathcal{G}_{[0,T]} \text{ Drive near the center lane}\right) \\ & \wedge \left(\mathcal{G}_{[0,T]} \text{ Stop areas within 10 meters ahead}\right. \\ & \Rightarrow \qquad \left.\text{Drive slowly}\right). \end{aligned} \tag{10}$$

Here T is the prediction horizon. ϕ takes the form of the conjunction of a set of sub-rules. Each sub-rule is enforced at all times within the prediction horizon (one can also define rules that are enforced at specific time intervals). Details of the predicates in the rule definitions are as follows

- "*Drive near the center lane*": denote $dist(\tilde{s}_t, lane_t)$ as the distance between the predicted trajectory and its closest point on the center lane at time t. This predicate is defined as $dist(\tilde{s}_t, lane_t) < \epsilon_{lane}$ where ϵ_{lane} is a rule parameter that can be manually set or tuned during training. In our experiments, we set $\epsilon_{lane} = 2$ meters.
- "*Stop areas within 10 meters ahead*": a stop area is defined as a stop sign, pedestrian crossing, turning stop or traffic light (the red regions shown in Fig. 1). Let $dist(s_0, s_{\text{stop area}})$ denote the distance between the agent (at the current observed time) and the closest stop area in front of it. This predicate is then defined as $dist(s_0, s_{\text{stop area}}) < 10$ meters.
- "*Drive slowly*": Let $\tilde{a}_t = (\tilde{v}_t, \tilde{\omega}_t)$ be the predicted control at time t. This predicate is defined as $|\tilde{v}_t| < \epsilon_v$ where ϵ_v is another rule parameter that defines the speed range the agent should drive within. We set $\epsilon_v = 1$ m/s.

It is worth mentioning that within our training set, 80.4% of the samples are compliant to the rules above (with robustness greater than zero). 86% of the samples in the validation set are rule-compliant. Therefore, the proposed predictor needs to learn and be rule-aware but not blindly follow the rules in order to be effective.

Implementation Details. We use the multi-modal trajectory predictor (MTP) [33] with the MobileNet-V2 backbone [34] (implemented in [13]) as the base predictor. We modify this predictor to take in a noise term (drawn from a normal distribution) of dimension 8 as an additional input feature. Doing so transforms MTP into a generator that we can incorporate in our GAN structure. For each input feature, we sample 3 trajectories from the generator and the trajectory with minimum average displacement error is fed to the discriminator.

We train and test with an observation history of 2 seconds and prediction horizon of 6 seconds with a frequency of 2 hz. We use a train batch size of 32 and accumulate gradient for 3 batches. we train for a total of 20 epochs on a cluster using 4 NVidia Tesla V100 GPUs.

Method of Evaluation. As performance metrics, we use the average displacement error (ADE) - average L2-norm between prediction and ground truth trajectories; final displacement error (FDE) - L2-norm between the final prediction and ground truth poses; max distance (MaxDist) - max L2-norm between prediction and ground truth as performance metrics. For each set of input features, we use the generator to sample 3 trajectories and the minimum of the above metrics are reported. All metrics are in meters. We use a training set of around 20000 trajectories and a validation set of around 4000 trajectories.

Results And Discussion. We use four training configurations for comparison. $Base$ indicates training with only the generator and discriminator, no rules are included and the base predictor outputs directly state trajectory (no dynamics); $BaseDyn$ is $Base$ with dynamics (which is the generator architecture shown in Fig. 3); $GLoss$ indicates using the rules as a generator auxiliary loss (7); $DFeature$ indicates using the syntax tree as discriminator features. Note that except for $Base$, all other configurations use dynamics (the rules require controls as part of their definition). In addition, we have also implemented the policy anticipation network (*PAN*) in [29]. In *PAN*, we designed 2 costmaps corresponding to "drive near the center lane" and "slow down near stop areas." We have also included an integrator kinematics in the cost function.

Fig. 4 shows the metric histograms for the trained predictors evaluated on the validation set. We also provide their statistics in Table I. From this set of results, we can see that between rules (*GLoss* and *DFeature*) and no-rules (*Base* and *BaseDyn*), adding rules can considerably reduce the long-tail error (especially in max-FDE and max-MaxDist). However, *GLoss* suffers from over-correction in the low-error regions as shown by its high min-ADE and min-FDE (recall that not all the samples in the data are rule-compliant). This over-correction phenomenon also slightly affects *DFeature* compared to the methods without rules. Using rules as discriminator features provide the predictor with a set of helpful priors but does not blindly encourage it to follow the rules in generating predictions. This can be supported by the low mean-ADE, max-ADE, mean-FDE, max-FDE of *DFeature* compared to *Base* and *BaseDyn*. It is also noticeable that the min-ADE/FDE/MaxDist of *DFeature* are slightly higher than other methods. This is also due to the fact that the rules encourage the generator to produce rule-compliant predictions whereas some of the ground-truths are rule-violating. Such an effect exists with both *GLoss* and *DFeature* but is stronger with *GLoss*. Therefore, *GLoss* is more suitable for rules that are rarely broken (such as "always drive on a drive-way") where *DFeature* should be used with rules that are more frequently broken (such as "always drive below the speed limit"). *PAN* performed worst of all the comparison cases. The main reason is that *PAN*'s encoder module relies only on the pose and velocity history to predict a behavior without considering other agents or the map which limits it's ability to make accurate prediction. *PAN* is also sensitive to how the costmaps are scaled when combined into a single cost function.

As a crude estimate of the spread of the trajectory distributions, for each scene we generate 20 trajectories. Then we take the predictions at the last time-step (where spread is largest) and calculate the covariance and its Frobenius norm. For each comparison case (except for *PAN* which is unimodal), we report the average Frobenius norm over scenes in the validation set as 1.26, 1.11, 0.67, 0.83 for *Base*, *BaseDyn*, *GLoss*, *DFeature* respectively. Intuitively this norm can be seen as an average distance to the mean of the trajectory distribution. Training with rule results in distributions concentrated towards the center lane compared to training without rules. *GLoss* gives the most

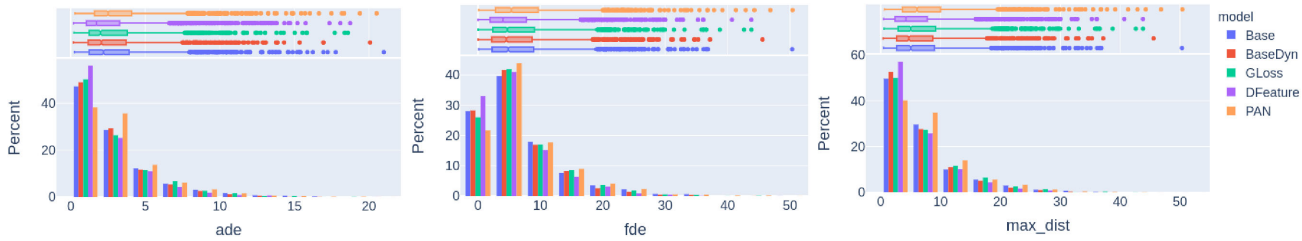

Fig. 4. **Performance metric histograms**. The top of each subfigure demonstrate a box plot of each statistic over the validation set examples. The bottom part of each subfigure demonstrates the binned distribution of each error statistic for each approach. The rule-based predictors reduce both the average and the spread of each of the error compared to the baseline. The DFeature approach results in the smallest overall spread of error.

TABLE I
PERFORMANCE METRIC STATISTICS. DFEATURE RESULTS IN IMPROVED ADE, FDE AND MAXIMUM DISTANCES COMPARED TO BOTH PREDICTION WITHOUT RULES AND THE GLOSS APPROACH, IN TERMS OF BOTH MEAN, MAX AND 90TH QUANTILE STATISTICS

	ADE (meters)				FDE (meters)				MaxDist (meters)			
	mean	**max**	**min**	**90^{th} quantile**	**mean**	**max**	**min**	**90^{th} quantile**	**mean**	**max**	**min**	**90^{th} quantile**
PAN	3.29	20.51	0.29	6.56	7.25	50.40	0.14	15.48	7.88	50.40	0.62	15.75
Base	3.08	20.99	0.24	6.57	6.74	50.33	0.12	15.22	6.98	50.34	0.49	15.21
BaseDyn	2.83	20.07	**0.18**	5.99	6.33	45.57	0.05	14.37	6.58	48.58	**0.4**	14.38
GLoss	2.91	**18.58**	0.22	6.43	6.77	43.82	0.03	15.38	7.08	43.82	0.56	15.52
DFeature	**2.53**	18.73	0.19	**5.44**	**5.74**	**43.81**	**0.02**	**12.98**	**5.99**	**43.81**	0.62	**12.98**

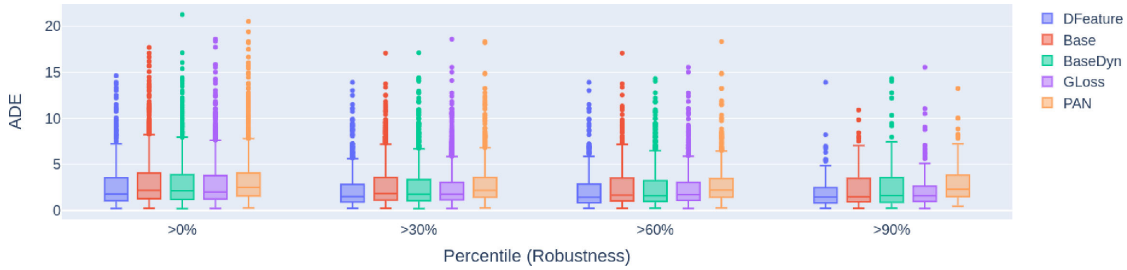

Fig. 5. **Distribution of ADE versus robustness percentile.** Each box plot is produced using predictions with robustness greater than the indicated percentile.

concentrated distribution given that it's more strict enforcement of the rules.

Fig. 5 shows distributions of ADE versus the robustness percentile. To produce this figure, the samples in the validation set are sorted by their robustness values and each box plot corresponds to the ADE statistics of samples greater than the indicated percentile We can see that for methods with rules, as the robustness of the data increases there is a notable decrease in ADE. This phenomenon is less apparent in methods without rules. This result indicates that for *DFeature* and *GLoss*, the predictor has learned to use the rules to improve prediction accuracy. Meanwhile, *Base* and *BaseDyn* are not provided with the rules during training and therefore results in weaker correlation between ADE and robustness.

Fig. 7 shows example scenarios trained with and without rules. For each of the sub-figures (a) and (b), the left scene depicts the prediction distribution from a *BaseDyn* model whereas the right depicts that from a *DFeature* model. Within each scene, the black dotted trajectory illustrates the agent's ground truth future; the dark gray dotted trajectory illustrates the agent's path history; the white trajectory distribution depicts 100 predicted trajectories sampled from the generator; the dot-dash lines represent lane centers (brown is incoming, yellow is current, white is outgoing).

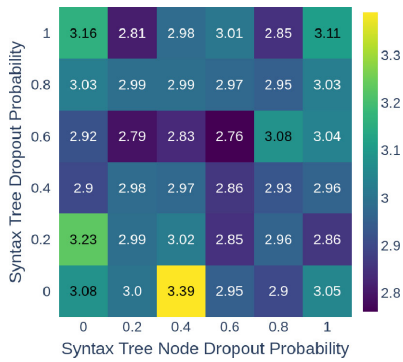

Fig. 6. **Syntax tree dropout analysis**. A grid search over the syntax tree dropout probabilities p_n and $p_{\mathcal{T}}$ in (9) with the ADE of each combination reported in the matrix. A probability of 0 indicates no dropout and probability of 1 indicates full dropout (turning the syntax/node features off).

Fig. 6 shows a grid search over the syntax tree dropout probabilities p_n and $p_{\mathcal{T}}$ in (9). The color scale and numbers shown in the matrix represent the resulting ADEs. Here $p_n, p_{\mathcal{T}} = 1$ corresponds to the syntax tree features completely

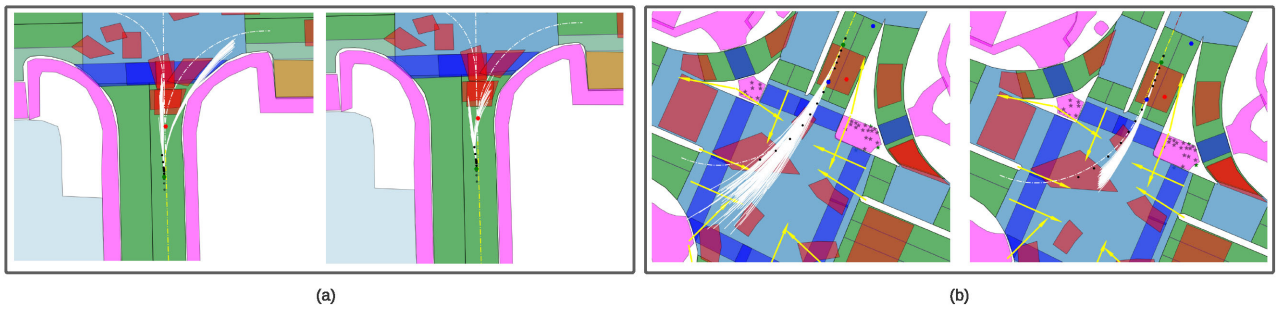

Fig. 7. **Example scenarios** of scenes are representative of rules (**a**) slow down approaching stop areas and (**b**) drive near the center lane. For each rule, the left scene depicts the prediction distribution from model *BaseDyn* and the right scene for that from model *DFeature*. Black dotted trajectories illustrate the agent's ground truth future. Dark gray dotted trajectories illustrate the agent's path history. White trajectory distributions depict 100 predicted trajectories sampled from the generator. Dot-dash lines represent lane centers (brown is incoming, yellow is current, white is outgoing). As can be seen, *DFeature* is able to generate lawful predictions but also maintains the distribution of unlawful behaviors.

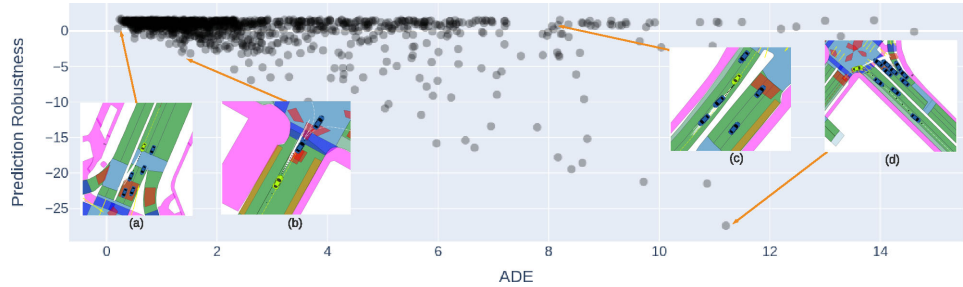

Fig. 8. **Robustness analysis** for model *DFeature*. Four example scenes are provided that illustrate the relationship between robustness and prediction accuracy. These include lawful and well-predicted examples (a,b), and lawful and unlawful prediction errors (c,d respectively).

turned off whereas a value of 0 corresponds to no dropout. We can observe from the figure that neither extreme values give satisfactory results and the best result occurs at $p_n, p_{\mathcal{T}} = 0.6$. The optimal values of the dropout values depend on the rules and the dataset. We expect that a lower value (keeping the tree features turned on) is more suitable when the dataset conforms to the given rules and vice versa.

Looking at the left scene of Fig. 7(a) we can observe that the baseline model failed to capture the fact that the agent has stopped in front of the stop area. However, training with rules results in more lawful predictions. Because the rule is provided as a set of discriminator features (as oppose to being strictly enforced), the model has learned a spectrum of predictions where both slowing down before and passing through the stop areas are possibilities. Fig. 7(b) shows example scenes where the "Drive near the center lane" takes effect. Compared to the model that generates the predictions in the left scene (trained without rules), the model in the right scene is able to generate a trajectory distribution that conforms to the provided rule and is closer to the ground truth on average.

Fig. 8 shows a scattered plot of prediction robustness versus ADE generated from evaluating *DFeature* on the validation data. We can observe a general trend that high robustness corresponds to low ADE (most agents drive according to the rules). In the figure we also show four examples scenes at different locations of the scatter plot. Scene (a) shows an example for high-robustness-low-ADE where the agent is driving according to the rules and the prediction is accurate. Scene (b) shows a case for low-robustness-low-ADE where the agent is violating the rules (does not slow down near stop areas and does not follow the provided lanes) and the model is also able to capture such behaviors. Scene (c) shows a case for high-robustness-high-ADE where the prediction abides by the rules but is not accurate with respective to the ground-truth. In this case, the prediction closely follows the center lane but fails to predict the correct driving speed. Scene (d) shows low-robustness-low-ADE which is the case where the predictor fails to predict rule-abiding agent behavior.

VI. Conclusion

In this letter, we propose two approaches of incorporating STL rules into a GAN style trajectory predictor. The first is to use the STL robustness as an auxiliary loss to the generator. The second is to use the sub-formulas of the STL syntax tree to calculate features for the discriminator. We show that compared to enforcing the rules as constraints as is commonly adopted in planning, our method is able to integrate the rules as soft priors to the predictor such that rule-violating agent behaviors can also be captured. Future direction includes exploring the use of the proposed architecture for ego-vehicle planning as well as considering rule priorities and trajectory uncertainty.

Acknowledgment

The content solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

REFERENCES

[1] S. Mozaffari, O. Y. Al-Jarrah, M. Dianati, P. Jennings, and A. Mouzakitis, "Deep learning-based vehicle behaviour prediction for autonomous driving applications: A review," 2019, *arXiv:1912.11676*.

[2] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social GAN: Socially acceptable trajectories with generative adversarial networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2255–2264.

[3] J. Li, H. Ma, Z. Zhang, and M. Tomizuka, "Social-WaGDAT: Interaction-aware trajectory prediction via wasserstein graph double-attention network," 2020, *arXiv:2002.06241*.

[4] X. Li, X. Ying, and M. Chuah, "GRIP : Enhanced graph-based interaction-aware trajectory prediction for autonomous driving," in *Proc. IEEE Intell. Transp. Syst. Conf.*, 2019, pp. 3960–3966.

[5] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, "Trajectron : Multi-agent generative trajectory forecasting with heterogeneous data for control," 2020, *arXiv:2001.03093*.

[6] N. Deo and M. M. Trivedi, "Convolutional social pooling for vehicle trajectory prediction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2018, pp. 1468–1476.

[7] X. Huang, S. G. McGill, B. C. Williams, L. Fletcher, and G. Rosman, "Uncertainty-aware driver trajectory prediction at urban intersections," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 9718–9724.

[8] H. Cui *et al.*, "Multimodal trajectory predictions for autonomous driving using deep convolutional networks," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 2090–2096.

[9] O. Makansi, E. Ilg, O. Cicek, and T. Brox, "Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 7144–7153.

[10] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," 2017, *arXiv:1708.06374*.

[11] J. White, "Illinois rules of the road," 2020. [Online]. Avilable: https://cyberdriveillinois.com/publications/pdf_publications/dsd_a112.pdf

[12] V. Raman, A. Donzé, M. Maasoumy, R. Murray, A. Sangiovanni-Vincentelli, and S. Seshia, "Model predictive control with signal temporal logic specifications," in *Proc. 53rd IEEE Conf. Decis. Control.*, 2014, pp. 81–87.

[13] H. Caesar *et al.*, "nuScenes: A multimodal dataset for autonomous driving," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11621–11631.

[14] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Proc. Int. Conf. Formal Model. Anal. Timed Syst.*, 2010, pp. 92–106.

[15] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. Torr, and M. Chandraker, "DESIRE: Distant future prediction in dynamic scenes with interacting agents," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 336–345.

[16] A. Sadeghian, V. Kosaraju, A. Sadeghian, N. Hirose, and S. Savarese, "Sophie: An attentive gan for predicting paths compliant to social and physical constraints," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 1349–1358.

[17] R. Chandra *et al.*, "Forecasting trajectory and behavior of road-agents using spectral clustering in graph-LSTMs," *IEEE Robot. Automat. Lett.*, vol. 5, no. 3, pp. 4882–4890, Jun. 2020.

[18] E. Wang, H. Cui, S. Yalamanchi, M. Moorthy, F.-C. Chou, and N. Djuric, "Improving movement predictions of traffic actors in bird's-eye view models using gans and differentiable trajectory rasterization," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 2340–2348.

[19] Z. Xu and U. Topcu, "Transfer of temporal logic formulas in reinforcement learning," in *Proc. Int. Joint Conf. Artif. Intell.*, vol. 28, 2019, p. 4010.

[20] P. Kapoor, A. Balakrishnan, and J. Deshmukh, "Model-based reinforcement learning from signal temporal logic specifications," 2020, *arXiv:2011.04950*.

[21] M. Hasanbeig, Y. Kantaros, A. Abate, D. Kroening, G. J. Pappas, and I. Lee, "Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees," in *Proc. IEEE 58th Conf. Decis. Control (CDC).*, 2019, pp. 5338–5343.

[22] C. Innes and S. Ramamoorthy, "Elaborating on learned demonstrations with temporal logic specifications," 2020, *arXiv:2002.00784*.

[23] K. Leung, N. Aréchiga, and M. Pavone, "Back-propagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods," 2020, *arXiv:2008.00097*.

[24] A. Censi *et al.*, "Liability, ethics, and culture-aware behavior specification using rulebooks," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 8536–8542.

[25] C. Vasile, J. Tumova, S. Karaman, C. Belta, and D. Rus, "Minimum-violation scLTl motion planning for mobility-on-demand," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 1481–1488.

[26] N. Aréchiga, "Specifying safety of autonomous vehicles in signal temporal logic," in *Proc. IEEE Intell. Veh. Symp. (IV)*, 2019, pp. 58–63.

[27] D. Park, M. Noseworthy, R. Paul, S. Roy, and N. Roy, "Inferring task goals and constraints using bayesian nonparametric inverse reinforcement learning," in *Proc. Conf. Robot. Learn.*, 2020, pp. 1005–1014.

[28] M. Bansal, A. Krizhevsky, and A. S. Ogale, "Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst," 2019, *arXiv:1812.03079*.

[29] W. Ding and S. Shen, "Online vehicle trajectory prediction using policy anticipation network and optimization-based context reasoning," *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 9610–9616.

[30] K. Leung, N. Aréchiga, and M. Pavone, "Backpropagation for parametric STL," in *Proc. IEEE Intell. Veh. Symp.*, 2019, pp. 185–192.

[31] I. J. Goodfellow *et al.*, "Generative adversarial nets," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014.

[32] H. Cui *et al.*, "Multimodal trajectory predictions for autonomous driving using deep convolutional networks," *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 2090–2096.

[33] N. Deo and M. M. Trivedi, "Multi-modal trajectory prediction of surrounding vehicles with maneuver based LSTMs," in *Proc. Intell. Veh. Symp. (IV)*, 2018, pp. 1179–1184.

[34] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.