Differentiable Logic Layer for Rule Guided Trajectory Prediction

Xiao Li
MIT
xiaoli@mit.edu

Guy Rosman
Toyota Research Institute
guy.rosman@tri.global

Igor Gilitschenski
MIT
igilitschenski@mit.edu

Jonathan A. DeCastro
Toyota Research Institute
jonathan.decastro@tri.global

Cristian-Ioan Vasile
Lehigh University
cvasile@lehigh.edu

Sertac Karaman
MIT
sertac@mit.edu

Daniela Rus
MIT
rus@mit.edu

Abstract: In this work, we propose a method for integration of temporal logic formulas into a neural network. Our main contribution is a new logic optimization layer that uses differentiable optimization on the formulas' robustness function. This allows incorporating traffic rules into deep learning based trajectory prediction approaches. In the forward pass, an initial prediction from a base predictor is used to initialize and guide the robustness optimization process. Backpropagation through the logic layer allows for simultaneously adjusting the parameters of the rules and the initial prediction network. The integration of a logic layer affords both improved predictions, as well as quantification rule satisfaction and violation during predictor execution. As such, it can serve as a parametric safety-envelope for black box behavior models. We demonstrate how integrating traffic rules improves the predictor performance using real traffic data from the NuScenes dataset.

Keywords: Trajectory prediction, temporal logic, autonomous driving

1 Introduction

Prediction is a key component in the autonomous driving stack that connects perception and planning. It is made difficult because a human driver's behavior is influenced by a number of factors including the semantics of the road, interactions with other road agents and personal preferences, and because the space of all possible future trajectories is prohibitively large even for modest prediction horizons. Recent methods in prediction mostly adopt a deep learning approach where much effort has been made in the design of network architectures and input features. While such approaches have seen significant progress, challenges such as encoding complex constraints and learning interpretable structures remain. *The aim of this paper is to develop a predictor that incorporates rules as a set of logic formulas and is compatible with existing deep architectures*.

The effective dimensionality of human behavior is significantly reduced by the set of rules that people tend to adhere to as they act and interact on the road. This makes the prediction problem more constrained, limiting the number of examples needed to train the prediction, as well as affording efficient reasoning about the system behavior both on- and off- line. These properties are important for efficiency, explainability, and safety. It is difficult to incorporate rules and regulations as priors to the predictor, and existing approaches tend to either use latent factors [1] or explicit modes to capture maneuver types [2], but fail to cater to the discrete constraint nature of rules in behavior.

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

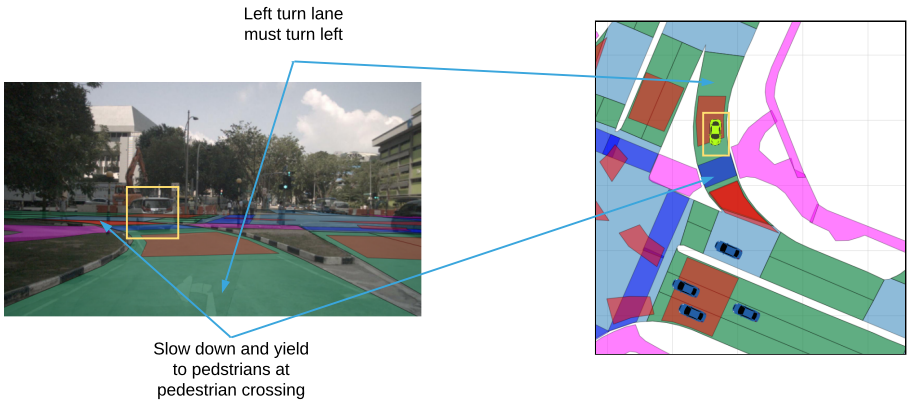

Figure 1 : Descriptive example of using rules for prediction. The agent to be predicted (vehicle in green box) is traveling on the left-turning lane approaching a pedestrian crossing. Applicable rules include "left turn lane must turn left" and "slow down at pedestrian crossing".

In the context of self-driving, symbolic information in the form of rules widely exists in traffic laws, regional regulations and driving heuristics. Consider the scenario shown in Figure 1 . Here the behavior of the car is governed by the fact that it is traveling in the left turning lane and is approaching a pedestrian walkway, therefore with high probability it will turn left while yielding to the pedestrian. Being able to explicitly integrate this information in prediction can considerably improve its performance. In addition, since rules and regulations vary across different regions [3], having a systematic means of maintaining the rules and explicitly integrating them with the predictor can be helpful in nationwide and global deployment of autonomous fleets.

In this paper, we propose to encode priors as a set of temporal logic formulas and integrate them into a differentiable logic control layer that can be used standalone or in conjunction with other predictors (Algorithm 1). Specifically, our contributions include:

- Utilizing techniques in formal control synthesis to incorporate rules and priors as a set of temporal logic formulas into trajectory predictors and maximize their degree of satisfaction;
- Using ideas in soft logic evaluation [4, 5] and differentiable optimal control [6, 7] to enable automatic tuning of logic parameters from data;
- Demonstrating on a real world dataset that incorporating our logic layer into existing deep prediction models improves sample efficiency and accuracy at long prediction horizons.

2 Related Work

Recent methods for trajectory prediction in autonomous driving have largely adopted a data-driven approach utilizing advances in deep learning. Advanced features include prediction of multi-modal trajectories [8], modeling of road agent interactions using graphs and graph neural networks [9, 10, 11], and incorporating both agent interactions and dynamic models [12, 13]. Generative adversarial networks have also seen increased usage as components of the predictor architectures [14, 15, 16], [17]. Mozaffari, et al [18] provide up-to-date surveys in this domain.

In addition to agent interactions, the semantics and rules of the road are also critical factors that influence driving behaviors. In recent literature, the semantics of the road are incorporated into trajectory predictors commonly as state features [19, 20]. Hong, et al [21] encode the current scene, road network and dynamic contexts as separate image inputs to the predictor network. Similarity, Bansal, et al[1] encode semantic information such as the roadmap, traffic lights and speed limits as image inputs to their proposed ChauffeurNet. A different approach taken by Ding, et al [2] encodes the road semantics as multi-layer costmaps that are used for trajectory optimization. More generally, many recent works in prediction highlight the mixed discrete-continuous nature of human behavior

on the road implicitly, either as GMM models [22, 23], or by conditioning on discrete modes and variables [24, 25, 12].

In comparison, we encode the semantics and rules of the road as a set of parameterized temporal logic formulas. Similar to the idea of rulebooks used by Censi, et al [26] for trajectory planning, having a repository of rules to govern the predictor's behaviour is helpful in promoting its interpretability and performance guarantees. In addition, we allow the parameters of the rules and their level of enforcement to be tuned using data. This alleviates the burden of having to hand design the details of the rules and only require the user to provide rule templates. The rules are ultimately incorporated into a differentiable trajectory generation layer that can be used to enhance the performance of other models.

Finally, our approach interacts with a large research effort aimed at taking programmatic concepts and algorithms and translating them into learned approaches. This has been shown to improve performance in optimization algorithms [27, 6], language constructs [28], data association [29], and object-based reasoning [30], as well other other topics.

3 Background - Signal Temporal Logic (STL)

STL offers a formalism for expressing and reasoning among a rich set of rules defined over predicates (inequalities of real-valued functions, e.g., the state-trajectories of cars) [31]. In this paper, the rules are encoded as a set of STL formulas with the following syntax

$$\phi := p(s) < \epsilon \,|\, \neg\phi \,|\, \phi \wedge \phi \,|\, \phi \vee \phi \,|\, F_{[a,b]}\phi \mid G_{[a,b]}\phi, \tag{1}$$

where $a, b \in \mathbb{R}_{\geq 0}$, $a < b$, represent finite time bounds; ϕ is an STL formula; $p(s) < \epsilon$ is a predicate; $s \in S \subseteq \mathbb{R}^n$ is a state; $p(\cdot) : \mathbb{R}^n \to \mathbb{R}$ is the predicate function, $\epsilon \in \mathbb{R}$ is a constant. $\neg$ (not), $\wedge$ (and) and $\vee$ (or) are Boolean operators. F (eventually) and G (always) are temporal operators.

Let $s_{t_0:t_1} = \{s_{t_0}, ..., s_{t_1}\}$ denote the state trajectory from time t_0 to t_1. $s_{t_0:t_1} \models \phi$ denotes that trajectory $s_{t_0:t_1}$ satisfies ϕ. The Boolean semantics of STL is provided in the supplementary material.

An example STL formula is $F_{[\tau_0,\tau_1]}(|s| < 2) \wedge G_{[\tau_1,\tau_2]}(s > 3)$ which means that "$|s_t| < 2$ is true for at least one $t \in [\tau_0, \tau_1)$ and $s_t > 3$ is true for all $t \in [\tau_1, \tau_2)$".

STL is equipped with a robustness degree (or robustness for short) that quantifies the level of satisfaction of a trajectory with respective to a formula. A robustness function takes in a trajectory and a STL formula and outputs a real-valued number. 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.The formal definition is given in the supplementary material.

Taking inspiration from parameterized STL (pSTL) [32], we allow the predicate function to be parameterized (i.e. $p^{\xi_p}(s)$ denotes a predicate function parameterized by ξ_p). Let ϕ^{ξ} denote an STL formula parameterized by $\xi = [\xi_p^1, \epsilon_1, ..., \xi_p^N, \epsilon_N]$ where N is the number of predicates in ϕ. For example, the pSTL formula $\phi = G_{[\tau_0,\tau_1]}\left(\alpha_1 x + \beta_1 y < \epsilon_1\right) \wedge F_{[\tau_2,\tau_3]}\left(\alpha_2 x^2 > \epsilon_2\right)$ have parameters $\xi = \{(\alpha_1, \beta_1), \epsilon_1, \alpha_2, \epsilon_2\}$. We will demonstrate how ξ can be tuned using data in later sections.

4 Differentiable STL Robustness Layer

In this section, we introduce a differentiable logic layer that is able to integrate a set of parametric temporal logic rules into a trajectory predictor. The idea of the differentiable STL robustness layer is that in the forward pass we find trajectories that maximize the robustness degree. In the backward pass, we differentiate the robustness with respect to STL parameters to find the parameters that allow the formula to best describe the agent behaviors in the dataset. In general, the $\max$ and $\min$ functions in the robustness definition cannot be differentiated with respect to their arguments. Hence, soft approximation such as the log-sum-exponential [33] or the soft-max [4] can be used (we use the latter in our experiments). A schematic is provided in Figure 2 .

The STL robustness layer can serve different purposes. It can be trained in conjunction with the base model which effectively "distills" the information contained in the rules into the base model's weights. It can also be used as a tuneable modification layer to a pre-trained base prediction model

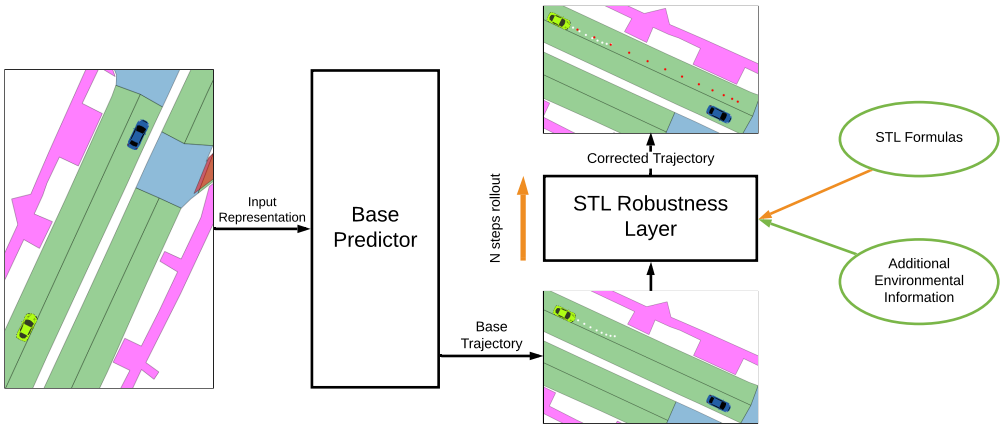

Figure 2 : Architectural schematic of differentiable STL robustness layer. The white trajectory represents the base predictor's output prediction. The red trajectory represents the output trajectory of the STL robustness layers.

whereby modifying the behavior of the base model using rules, dynamics and other priors without having to re-collect data or re-train the base model.

Recall that $s_{t_0:t_1} = \{s_{t_0}, ..., s_{t_1}\}$ is the state trajectory from time t_0 to t_1. Let $a_{t_0:t_1}$ be the set of actions from time t_0 to t_1. Given a base predictor that provides trajectory prediction, let $s^b_{t_0:t_1}$ be the trajectory output of the base predictor. $s^c_{t_0:t_1}$ is the corrected state trajectory output of the logic layer. $s^{gt}_{t_0:t_1}$ is the ground truth trajectory (the target). We distinguish the trajectory output from the state feature inputs to the base predictor. In our case, the former is a time sequence of (x, y) coordinates, and the latter can be any state representation (images, sequences, graphs, etc). Let ϕ^ξ be the pSTL formula that encodes the rules to be satisfied.

4.1 Without Dynamics

In the forward pass, we wish to solve the optimization problem in Equation (2)-*left*.

$$s^c_{t_0:t_1} = \arg\max_{s_{t_0:t_1}} r(s_{t_0:t_1}, \phi^\xi), \qquad s^i_{t_0:t_1} = s^{i-1}_{t_0:t_1} + \gamma \frac{\partial r(s^{i-1}_{t_0:t_1}, \phi^\xi)}{\partial s^{i-1}_{t_0:t_1}}. \tag{2}$$

Given the base prediction $s^b_{t_0:t_1}$ and the STL formula, Equation (2)-*left* tries to find a trajectory that maximizes the robustness (best satisfies the formula). To do so, we roll out N gradient ascent steps. Let $i \in [1, N]$, Equation (2)-*right* shows the gradient ascent step taken at each rollout step. $s^i_{t_0:t_1}$ is the output of the $(i-1)^{th}$ rollout step, $s^0_{t_0:t_1} = s^b_{t_0:t_1}$, $s^N_{t_0:t_1} = s^c_{t_0:t_1}$.

In the backward pass, we tune ξ to minimize the loss $L = ||s^c_{t_0:t_1} - s^{gt}_{t_0:t_1}||_2$ where $||\cdot||_2$ denotes the L2-norm (other losses, such as minimum of N loss [34] can be used in place of L, e.g. to address multi-modal predictions). Similar to backpropagation through time commonly used to train recurrent neural networks, the gradient for the parameters of the STL formula can be calculated by

$$\frac{\partial L}{\partial \xi} = \frac{\partial L}{\partial s^c_{t_0:t_1}} \sum_{i=0}^{N-1} \frac{\partial s^c_{t_0:t_1}}{\partial s^i_{t_0:t_1}} \frac{\partial s^i_{t_0:t_1}}{\partial \xi}, \qquad i \in \{0, N-1\}. \tag{3}$$

The above gradient can be used to update the STL parameters using data. Equation (3) can be combined with the gradients of the base predictor to train it with the STL layer simultaneously.

4.2 Dynamics-Aware Prediction

To incorporate dynamics in the above formulation, we augment our forward and backward passes with a differentiable dynamics model $s_{t+1} = f(s_t, a_t)$. In our forward pass, we update the control

action rather than state; hence Equation (2)-*right* becomes

$$a^i_{t_0:t_1} = a^{i-1}_{t_0:t_1} + \gamma \frac{\partial r(s^{i-1}_{t_0:t_1}, \phi^\xi)}{\partial a^{i-1}_{t_0:t_1}}, \quad s^i_{t_0:t_1} = \hat{f}(s^{i-1}_{t_0:t_1}, a^i_{t_0:t_1}) \tag{4}$$

where $\hat{f}$ is the vector operation that applies $f(\cdot,\cdot)$ to $(s^{i-1}_{t_0:t_1}, a^i_{t_0:t_1})$ at every time-step. The backward pass is modified to be

$$\frac{\partial L}{\partial \xi} = \frac{\partial L}{\partial s^c_{t_0:t_1}} \sum_{i=0}^{N-1} \frac{\partial s^c_{t_0:t_1}}{\partial s^i_{t_0:t_1}} \frac{\partial s^i_{t_0:t_1}}{\partial a^i_{t_0:t_1}} \frac{\partial a^i_{t_0:t_1}}{\partial \xi} \tag{5}$$

Having even a simple model (such as a unicyle model) can considerably enhance the predicted trajectory in terms of dynamic feasibility. This is especially important when differentiating through the robustness degree as it is possible to find a trajectory with a high robustness value but not achievable by a vehicle. Algorithm 1 summarizes the differentiable STL robustness layer. There, the computation of the formulas' robustness function (`ComputeRobustness`) and their gradients (`ComputeRobustnessGradient`) is implemented in a differentiable way. Thus, we can backpropagate through the entire optimization process.

Algorithm 1 STL Robustness Layer

Inputs: pSTL formula ϕ; formula parameters ξ; initial temporal sequence $s^{\text{init}}_{0:T}$; step size γ; number of iterations N
$s^0_{0:T} \leftarrow s^{\text{init}}_{0:T}$
for i=1 ... N **do**
    $g_{i-1} \leftarrow$ `ComputeRobustnessGradient`$(s^{i-1}_{0:T}, \phi, \xi)$
    $s^i_{1:T} \leftarrow s^{i-1}_{1:T} + \gamma \cdot g_{i-1}$
end for
$v \leftarrow$ `ComputeRobustness`$(s^N_{0:T}, \phi, \xi)$
return v, $s^N_{0:T}$

5 Experiments

We train and evaluate our method on the NuScenes dataset. NuScenes [35] is a dataset for autonomous driving based in Boston and Singapore. It contains 1000 scenes each 20s long, containing 23 object classes and HD semantic maps with 11 annotated layers. We chose this particular dataset for the rich semantics it provides which is well suited for rule definitions. Our goal is to show that given an off-the-shelf base predictor, incorporating the STL robustness layer provides valuable priors and constraints that improve the performance of the overall predictor in terms of accuracy, sample efficiency and prediction horizon. Details in experiment setup, implementation and hyperparameters used are provided in the supplementary material.

5.1 System Model

Most trajectory predictors in the literature directly output trajectories using fully connected or recurrent layers (e.g. LSTM). While such approaches have yielded satisfactory results, incorporating kinematics models of the vehicle often improves the predicted trajectories [12], and is easily supported in our method in a differentiable way (Section 4.2). In this set of experiments, we use the following unicycle model $[\dot{x},\ \dot{y},\ \dot{\psi}]^\top = [v\cos(\psi),\ v\sin(\psi),\ \omega]^\top$ where state $s = (x, y, \psi)$ contains the planar coordinates and heading of the predicted vehicle. The control $a = (v, \omega)$ contains the forward velocity and steering rate.

5.2 Rules Used

$$\begin{aligned}\phi = &(G_{[0,T]} \text{ Stay close to the center lane}) \wedge \\ &(G_{[0,T]} \text{ Stay close to the base prediction}) \wedge \\ &(G_{[0,T]} \text{ No stop areas within 30 meters ahead } \Rightarrow \text{ Stay close to the observed speed}),\end{aligned} \tag{6}$$

where T is the prediction horizon. ϕ takes the form of a conjunction of a set of sub-rules. Each sub-rule is a predicate that is enforced at all timesteps within the prediction horizon (hence the *always* operator). Detailed descriptions are as follows:

- "Stay close to the center lane": denoted $dist(s_t^c, lane_t)$ as the distance between the prediction and its closest lane at time t. This predicate is then defined as $dist(s_t^c, lane_t) < \epsilon_{lane}$ where ϵ_{lane} is a tune-able parameter for this predicate.
- "Stay close to the base prediction": we do not wish to deviate too much from the base prediction. Because we incorporate the system model, instead of encouraging the $s^c_{t_0:t_1}$ to be close to $s^b_{t_0:t_1}$, we encourage their corresponding actions to be close. Given the base prediction $s^b_{t_0:t_1}$ and the sample frequency, the corresponding controls of dynamic system can be estimated using finite differences of adjacent states. Let a_t^b and a_t^c be the corresponding action sequences, this predicate is defined as $||a_t^c - a_t^b||_2 < \epsilon_{base}$ where ϵ_{base} is the tune-able parameter.
- "No stop areas within ϵ_{stop} meters ahead $\Rightarrow$ Stay close to the observed speed": this "if-else" statement is used to enforce the assumption that if there is no reason to slow down, the vehicle should travel close to its current observed speed (for the prediction horizon). In NuScenes, areas where a stop sign, traffic light or pedestrian crossing exist are marked as stop areas (shown as red patches in Figure 1). We detect stop areas up to ϵ_{stop} meters in front of the vehicle where ϵ_{stop} is a tune-able parameter. Let $v_{t_0}^{obs}$ and v_t^c be the observed and predicted vehicle speed at time t_0 and t respectively. Let also s_t^{stop} be the position of closest stop area at time t. This predicate is defined as $||s_t^c - s_t^{stop}||_2 < \epsilon_{stop} \Rightarrow (v_t^c - v_{t_0})^2 < \epsilon_v$ where ϵ_v is another predicate parameter.

5.3 Method of Evaluation

Using the notation from Section 4, we denote $s^c_{t_0,t_1}$ as the prediction output (for multi-modal output, this is the trajectory with highest probability) and $s^{gt}_{t_0,t_1}$ as the ground truth. 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. All metrics are in meters. We use a training set of around 25000 trajectories and a validation set of around 3000 trajectories. We use the multi-modal trajectory predictor (MTP) [36] with the MobileNet-V2 backbone [37] (implemented in [35]) as the base predictor. The STL layer we use here contains 15 rollout steps and 4 rule parameters.

In our experiments, we first train the base predictor on the dataset and fix its weights. Subsequently, the logic layer is trained using the output of the base predictor as its input. We evaluate the effect of adding the logic layer on the prediction performance in terms of accuracy, prediction horizon and sample efficiency.

5.4 Results And Discussion

We study the performance of the STL robustness layer by connecting it to the output layer of MTP. We refer to results produced with the STL layer as MTP-STL. Figure 3 (top) shows the training curves with and without the STL layer. It can be observed that incorporating the STL layer reduces both the mean and variance of the performance metrics. Figure 3 (bottom) shows a histogram of the metric distributions for the trained predictor. As highlighted by the dashed red circle, the STL layer is effective in reducing the long tail of the distribution (eliminating edge cases). We are able to achieve 5.3%, 6% and 6.2% decrease in average ADE, FDE, and MaxDist respectively. And 11 %, 27% and 26.8% decrease in max ADE, FDE and MaxDist respectively. However, this can come at slight cost of over-correction in situations where the base prediction is accurate but doesn't fully comply with the rules. This can be observed in the histogram where the mass of the distribution for MTP-STL shifts slightly to the right at low metric values. This can be alleviated by either adding more rules (if there is systematic bias) or make the rules less constrained. Our method does not assume that rules are never broken and tries to strike a balance between rule satisfaction and accuracy with respect to the dataset.

Figure 4 shows two example scenarios in birds-eye view and their positions on the ADE histogram with and without using the STL layer. In the figures, black denotes ground truth trajectory, white denotes base prediction and red denotes corrected trajectory. The dot-dash line denotes the closest

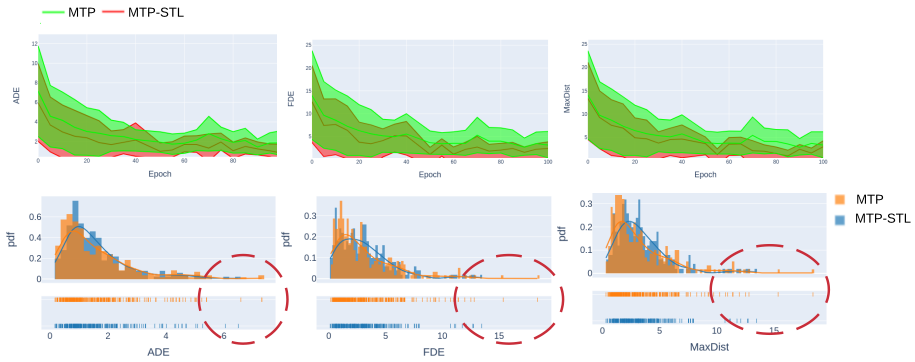

Figure 3 : Prediction performance comparison. (**top**): Training curves with and without the STL layer. Our approach results in consistently lower errors. (**bottom**): Histogram of the performance metric distribution over the validation set with 1 standard deviation (for the trained predictor). The dashed red circle highlights how the STL layer prevents outlier predictions that often appear in neural-network predictions, and reduced the long tail of prediction errors.

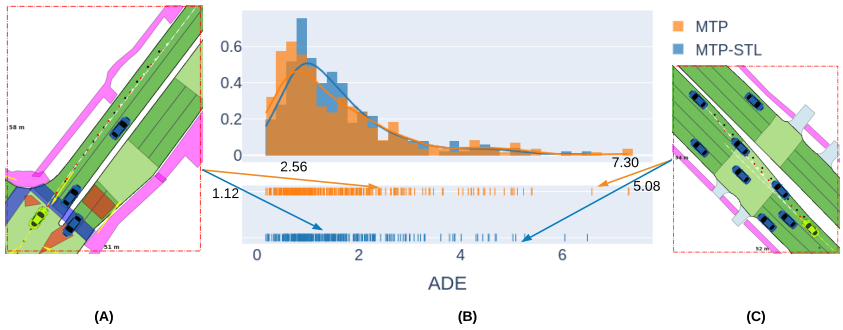

Figure 4 : Example birds-eye view of base and corrected predictions in sample scenarios. Black denotes ground truth trajectory, white denotes base prediction and red denotes corrected prediction. The dot-dash lines illustrate the closest center lane. (**A**): A scenario that shows preservation of accurate base prediction. (**B**): Corresponding ADE histogram (curve peaks represent the modes). (**C**): A scenario that show a larger modification to an inaccurate base prediction.

lane center. We can observe from the figures that when the base prediction is accurate (Figure 4 (A)), the STL layer can preserve this accuracy without making much corrections. When the base prediction is inaccurate (Figure 4 (B)), the STL layer is able to significantly correct the erroneous base prediction.

Figure 5 illustrates how a rule parameter is trained to adapt the rule to ground truth data. In Figure 5 (A), the agent is driving on the right-most lane about to make a lane change to the right. The base predictor predicted this lane change but its prediction is delayed and overshot to the right. Applying the forward pass of the STL layer to the base prediction results in over-correction towards the current center lane. In this small experiment, we took 20 backward passes using the ground truth to train the rule parameters. As shown in Figure 5 (B), the parameter ϵ_{lane} that controls how close the prediction should be with respective to the center lane increases from 0.1 to 1.32 meter and thus imposes less constraint on the correct trajectory. As a result, applying the STL layer with the updated ϵ_{lane} corrects for the overshooting but is not too constrained to the current lane center.

To analyze the performance gains in using the STL layer, we train comparison models at 3 different prediction horizons and 3 different training dataset sizes. All the results are reported on the validation set of the same size. Figure 6 shows that in general, given our setting and the rules we have chosen, the STL layer the effective in reducing the prediction errors especially in the edge cases.

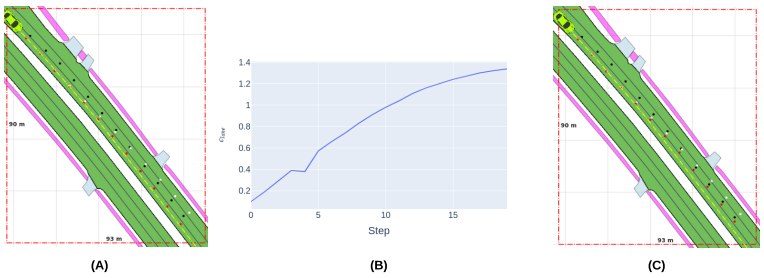

Figure 5 : Rule parameter training to adjust to the ground truth data. (**A**) : Trajectory correction before parameter learning. (**B**): Parameter learning curve. (**C**): Trajectory correction after parameter learning.

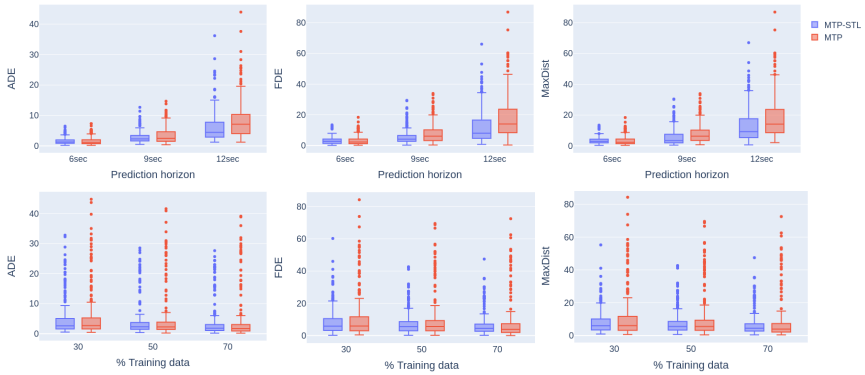

Figure 6 : Sample efficiency and prediction horizon study. Metric statistic comparison for models trained and evaluated at (**top**) 6, 9, 12 seconds prediction horizons and (**bottom**): 30%, 50% and 70% training data. The validation set is kept fixed. The STL layer improves the prediction results, with emphasis on long horizons and limited training data cases.

This can be observed by the lowered outlier values. The performance improvement is more evident as the base predictor's performance deteriorates (at longer prediction horizon and less training data). Therefore, it is reasonable to expect a well-designed set of rules can be used to compensate for data and base model deficiency. In addition, given that the robustness measures how much the base prediction violates the rules, a large violation can indicate potentially hazardous behavior and serve as an alert to the downstream planner.

6 Conclusion

In this paper, we propose to use a differentiable logic layer as a learnable correction layer for vehicle trajectory prediction. We show that the logic layer is capable of incorporating symbolic priors as parametric temporal logic formulas into the predictor. We show in experiments that using the logic layer we are able to considerably improve the performance of the base predictor without having to collect more data or re-train the base predictor. Our formulation easily integrates a system model, allowing us to predict the controls of the vehicles of interest. We can thus effectively turn trajectory predictors into trajectory generators that can be use in planning for the ego-vehicle. This is a future direction that we are highly interested in. We will also explore the scalability of the proposed approach in terms of the rule complexity it is able to handle, and the tradeoff with data prevalence.

Acknowledgments

This work is supported by the Toyota Research Institute (TRI). This article solely reflects the opinions and conclusions of its authors and not TRI, Toyaota, or any other Toyota entity. Their support is gratefully acknowledged. We also thank Nvidia for providing computation resources.

References

[1] M. Bansal, A. Krizhevsky, and A. S. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. ArXiv, abs/1812.03079, 2019.

[2] W. Ding and S. Shen. Online vehicle trajectory prediction using policy anticipation network and optimization-based context reasoning. 2019 International Conference on Robotics and Automation (ICRA), pages 9610–9616, 2019.

[3] H. Allen. State road rules: A troubling patchwork of regulations. https://medium.com/aurora-blog/state-road-rules-a-troubling-patchwork-of-regulations-f2b77629d523, August 2019.

[4] K. Leung, N. Aréchiga, and M. Pavone. Backpropagation for parametric stl. 2019 IEEE Intelligent Vehicles Symposium (IV), pages 185–192, 2019.

[5] I. Haghighi, N. Mehdipour, E. Bartocci, and C. Belta. Control from signal temporal logic specifications with smooth cumulative quantitative semantics. 2019 IEEE 58th Conference on Decision and Control (CDC), pages 4361–4366, 2019.

[6] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. In ICML, 2017.

[7] A. Agrawal, B. Amos, S. T. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. ArXiv, abs/1910.12430, 2019.

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

[9] D. Lee, Y. Gu, J. Hoang, and M. Marchetti-Bowick. Joint interaction and trajectory prediction for autonomous driving using graph neural networks. ArXiv, abs/1912.07882, 2019.

[10] R. Chandra, T. Guan, S. Panuganti, T. Mittal, U. Bhattacharya, A. Bera, and D. Manocha. Forecasting trajectory and behavior of road-agents using spectral clustering in graph-lstms. IEEE Robotics and Automation Letters, 5:4882–4890, 2020.

[11] X. Li, X. Ying, and M. C. Chuah. Grip: Graph-based interaction-aware trajectory prediction. 2019 IEEE Intelligent Transportation Systems Conference (ITSC), pages 3960–3966, 2019.

[12] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone. Trajectron++: Multi-agent generative trajectory forecasting with heterogeneous data for control. ArXiv, abs/2001.03093, 2020.

[13] J. Li, H. Ma, Z. Zhang, and M. Tomizuka. Social-wagdat: Interaction-aware trajectory prediction via wasserstein graph double-attention network. ArXiv, abs/2002.06241, 2020.

[14] A. Gupta, J. E. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2255–2264, 2018.

[15] E. Wang, H. Cui, S. B. Yalamanchi, M. P. S. 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. ArXiv, abs/2004.06247, 2020.

[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. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 1349–1358, 2019.

[17] V. Kosaraju, A. Sadeghian, R. Martín-Martín, I. D. Reid, S. H. Rezatofighi, and S. Savarese. Social-bigat: Multimodal trajectory forecasting using bicycle-gan and graph attention networks. ArXiv, abs/1907.03395, 2019.

[18] S. Mozaffari, O. Y. Al-Jarrah, M. Dianati, P. Jennings, and A. Mouzakitis. Deep learning-based vehicle bfehaviour prediction for autonomous driving applications: A review. ArXiv, abs/1912.11676, 2019.

[19] R. Chandra, U. Bhattacharya, A. Bera, and D. Manocha. Traphic: Trajectory prediction in dense and heterogeneous traffic using weighted interactions. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 8475–8484, 2019.

[20] X. Huang, S. G. McGill, J. A. DeCastro, L. Fletcher, J. J. Leonard, B. C. Williams, and G. Rosman. Diversitygan: Diversity-aware vehicle motion prediction via latent semantic sampling. IEEE Robotics and Automation Letters, 5(4):5089–5096, 2020.

[21] J. Hong, B. Sapp, and J. Philbin. Rules of the road: Predicting driving behavior with a convolutional model of semantic interactions. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 8446–8454, 2019.

[22] X. Huang, S. G. McGill, B. C. Williams, L. Fletcher, and G. Rosman. Uncertainty-aware driver trajectory prediction at urban intersections. In 2019 International Conference on Robotics and Automation (ICRA), pages 9718–9724. IEEE, 2019.

[23] A. Amini, G. Rosman, S. Karaman, and D. Rus. Variational end-to-end navigation and localization. In 2019 International Conference on Robotics and Automation (ICRA), pages 8958–8964. IEEE, 2019.

[24] N. Deo and M. M. Trivedi. Multi-modal trajectory prediction of surrounding vehicles with maneuver based LSTMs. In 2018 IEEE Intelligent Vehicles Symposium (IV), pages 1179–1184. IEEE, 2018.

[25] Y. C. Tang and R. Salakhutdinov. Multiple futures prediction, 2019.

[26] A. Censi, K. Slutsky, T. Wongpiromsarn, D. S. Yershov, S. Pendleton, J. G. M. Fu, and E. Frazzoli. Liability, ethics, and culture-aware behavior specification using rulebooks. 2019 International Conference on Robotics and Automation (ICRA), pages 8536–8542, 2019.

[27] K. Gregor and Y. LeCun. Learning fast approximations of sparse coding. In Proceedings of the 27th international conference on international conference on machine learning, 2010.

[28] Y. Kuo, B. Katz, and A. Barbu. Deep compositional robotic planners that follow natural language commands. CoRR, abs/2002.05201, 2020.

[29] Y. Xu, A. Osep, Y. Ban, R. Horaud, L. Leal-Taixé, and X. Alameda-Pineda. How to train your deep multi-object tracker. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 6787–6796, 2020.

[30] T. Kipf, E. van der Pol, and M. Welling. Contrastive learning of structured world models. arXiv preprint arXiv:1911.12247, 2019.

[31] A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In FORMATS, 2010.

[32] E. Asarin, A. Donzé, O. Maler, and D. Nickovic. Parametric identification of temporal properties. In Runtime Verification, pages 147–160. Springer, 2012.

[33] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta. Q-learning for robust satisfaction of signal temporal logic specifications. 2016 IEEE 55th Conference on Decision and Control (CDC), pages 6565–6570, 2016.

[34] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. S. Torr, and M. K. Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2165–2174, 2017.

[35] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. ArXiv, abs/1903.11027, 2019.

[36] N. Deo and M. M. Trivedi. Multi-modal trajectory prediction of surrounding vehicles with maneuver based lstms. Intelligent Vehicles Symposium (IV), Jun 2018.

[37] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 4510–4520, 2018.

Differentiable Logic Layer for Rule Guided Trajectory Prediction Supplementary Material

1 Signal Temporal Logic – Boolean and Quantitative Semantics

Let $s_{t_0:t_1} = \{s_{t_0}, ..., s_{t_1}\}$ denote the state trajectory from time t_0 to t_1. $s_{t_0:t_1} \models \phi$ denotes that trajectory $s_{t_0:t_1}$ satisfies ϕ. The Boolean semantics of STL is defined recursively in Equation (1a). STL also admits quantitative semantics in the form of a robustness degree $r(s_{t_0,t_1}, \phi)$ given in Equation 1b that quantifies satisfaction and violation of specification ϕ by signal s_{t_0,t_1}.

$$
\begin{aligned}
s_{t_0:t_1} &\models (p(s) < \epsilon) && \Leftrightarrow p\left(s_{t_0}\right) < \epsilon \\
s_{t_0:t_1} &\models \neg(p(s) < \epsilon) && \Leftrightarrow \neg(s_{t_0:t_1} \models (p(s) < \epsilon)) \\
s_{t_0:t_1} &\models \phi_1 \wedge \phi_2 && \Leftrightarrow s_{t_0:t_1} \models \phi_1 \wedge s_{t_0:t_1} \models \phi_2 \\
s_{t_0:t_1} &\models \phi_1 \vee \phi_2 && \Leftrightarrow s_{t_0:t_1} \models \phi_1 \vee s_{t_0:t_1} \models \phi_2 \\
s_{t_0:t_1} &\models G_{[a,b]}\phi && \Leftrightarrow s_{t',t_1} \models \phi \ \forall t' \in [a,b) \\
s_{t_0:t_1} &\models F_{[a,b]}\phi && \Leftrightarrow \exists t' \in [a,b) \text{ s.t. } s_{t':t_1} \models \phi
\end{aligned}
\tag{1a}
$$

$$
\begin{aligned}
r(s_{t_0:t_1}, (p(s) < \epsilon)) &= \epsilon - p\left(s_{t_0}\right) \\
r(s_{t_0:t_1}, \neg\phi) &= -r(s_{t_0:t_1}, \phi) \\
r\left(s_{t_0:t_1}, \phi_1 \wedge \phi_2\right) &= \min(r(s_{t_0:t_1}, \phi_1), r(s_{t_0:t_1}, \phi_2)) \\
r\left(s_{t_0:t_1}, \phi_1 \vee \rho_2\right) &= \max(r(s_{t_0:t_1}, \phi_1), r(s_{t_0:t_1}, \phi_2)) \\
r\left(s_{t_0:t_1}, G_{[a,b]}\phi\right) &= \inf_{t' \in [a,b)} r\left(s_{t':t_1}, \phi\right) \\
r\left(s_{t_0:t_1}, F_{[a,b]}\phi\right) &= \sup_{t' \in [t_0,t_1)} r\left(s_{t_0:t_1}, \phi\right)
\end{aligned}
\tag{1b}
$$

The robustness degree is sound [1, 2], meaning that positive values of r imply satisfaction, while negative values imply violation.

An STL formula ϕ is said to be time-bounded if there exists a finite duration d, the formula's horizon, such that the all signals of duration at least d can be checked for satisfaction or violation of the formula. The horizon of the an STL formula can be computed recursively based on the time bounds of the temporal operators [3].

In the definition of semantics, we assume that the lengths of the time intervals $[t_0, t_1]$ associated with a signals are always greater than the horizon of the considered STL formulas. Thus, satisfaction and robustness degrees can be computed.

Example 1. *Assume a robot is navigating in the 2D plane with its coordinates as states i.e. $s = (x, y)$. Let A and B be two circular regions with c_A, r_A and c_B, r_B be the center and radius of A and B respectively. Define predicates $\psi_A = ||s - c_A||_2 < r_A$ and $\psi_B = ||s - c_B||_2 < r_B$ where $||\cdot||_2$ is the L2-norm. ψ_A and ψ_B tracks whether the robot has entered A and B. We can define a task using STL as*

$$
\phi = F_{[0,T]}\psi_A \wedge G_{[0,T]}\neg\psi_B, \tag{2}
$$

where T is the horizon of the task. ϕ describes a navigation task that requires the robot to "eventually visit region A and always avoid region B for all times between 0 and T". Given a trajectory

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.

of the robot $s_{0:T}$, we can evaluate the robustness of the trajectory with respective to ϕ by applying Equation (1b) recursive which gives us

$$r(s_{0:T}, \phi) = \min \big[\max_{t \in [0,T)} (r_A - ||s_t - c_A||_2), \min_{t \in [0,T)} (||s_t - c_B||_2 - r_B) \big]. \tag{3}$$

$r(s_{0:T}, \phi) > 0$ *implies that the robot has completed the task.*

2 Learning Approach – Algorithmic Details

We first describe the learning approach used in our paper. The approach utilizes Algorithm 1 from the paper as a building block within either optimization of the STL parameters ξ, or a joint optimization w.r.t ξ, θ. The prediction correction, and the learning step for this approach are described in Algorithm 1 and Algorithm 2 respectively.

Algorithm 1 Prediction with the STL Robustness Layer

Inputs: The base predictor $\mathcal{B}^\theta$ parameterized by θ; the pSTL formula ϕ^ξ parameterized by ξ; Input representations $\mathcal{I}$ and target trajectories S^{gt}; number of robustness rollout steps N;
$s^0_{t_0:t_1} = \mathcal{B}^\theta(\mathcal{I})$
for $i = 1$ to N **do**
    $g_{i-1} \leftarrow \texttt{ComputeRobustnessGradient}(s^{i-1}_{0:T}, \phi, \xi)$
    $s^i_{1:T} \leftarrow s^{i-1}_{1:T} + \gamma \cdot g_{i-1}$
end for
return $s^N_{t_0:t_1}$

Algorithm 2 Learning with the STL Robustness Layer

Inputs: The base predictor $\mathcal{B}^\theta$ parameterized by θ; the pSTL formula ϕ^ξ parameterized by ξ; mode (can be "learn" or "predict"); dataset with input representations $\mathcal{I}$ and target trajectories S^{gt}; number of robustness rollout steps N; loss function L; learning rate γ
Initialize $\theta \leftarrow \theta_0$ and $\xi \leftarrow \xi_0$
if mode is "learn" **then**
    for $j = 0$ to number of training steps **do**
        $s^0_{t_0:t_1} = \mathcal{B}^\theta(\mathcal{I})$ ▷ Start of forward pass, cf. Alg 1 in the paper.
        for $i = 1$ to N **do**
            $g_{i-1} \leftarrow \texttt{ComputeRobustnessGradient}(s^{i-1}_{0:T}, \phi, \xi)$
            $s^i_{1:T} \leftarrow s^{i-1}_{1:T} + \gamma \cdot g_{i-1}$
        end for
        $L = ||s^c_{t_0:t_1} - s^{gt}_{t_0:t_1}||_2$ ▷ Start of backward pass
        $\frac{\partial L}{\partial \theta}, \frac{\partial L}{\partial \xi} = \texttt{CalculateGradient}(L, s^{0:N}_{t_0:t_1})$ ▷ Using Equations (2) or (4) in the paper.
        $\theta_j = \texttt{UpdateParameter}(\frac{\partial L}{\partial \theta}, \gamma)$
        $\xi_j = \texttt{UpdateParameter}(\frac{\partial L}{\partial \xi}, \gamma)$
    end for
return θ, ξ
end if

3 Experimental Details

We use the NuScenes [4] dataset for experimentation and validation. Two off-the-shelf predictors - the Multi-modal trajectory predictor - MTP [5] (implementation available at [6]) and Trajectron++ [7] (implementation available at [8]) are used as base predictors. We present the results for MTP in the paper and Trajectron++ in Section 4.1 below. For MTP, we trained with 3 modes (3 output trajectories each with a probability). The trajectory with the highest probability is passed on to the STL layer. We use a batch size of 16 for training. Gradients are accumulated for 6 batches before

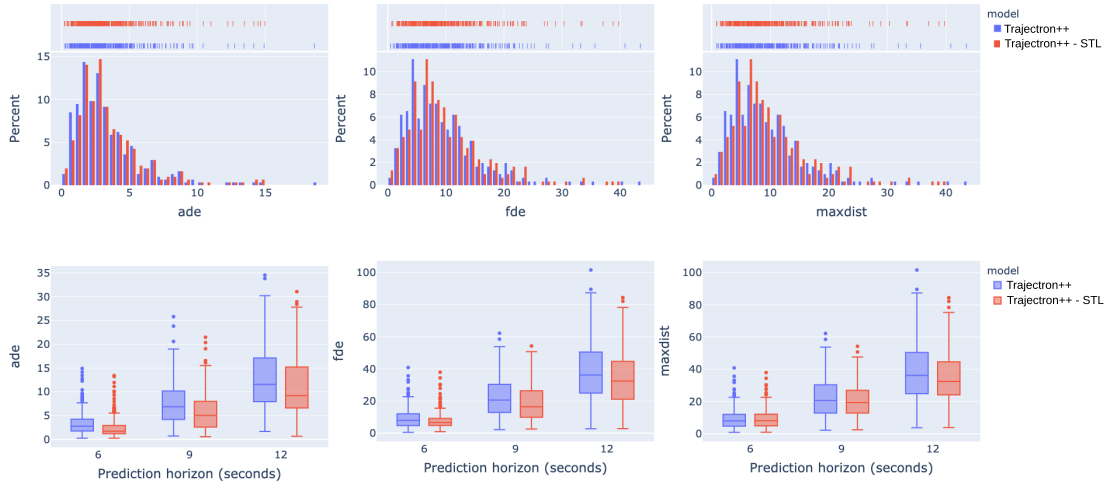

Figure 1 : Trajectron++ - STL results. (**top**): Histogram of the performance metric distribution over the validation set. (**bottom**): Comparison of validation results with and without the STL layer for different prediction horizons.

performing an update. A learning rate of 3×10^{-4} is used. We set the regression loss weight equal to 0.8 and trained for a total of 100 epoches in all our experiments. For Trajectron++, we use the default hyperparameters provided in the provided implementation. Training is performed on a cluster using 4 NVidia Tesla V100 GPUs.

In our implementation of the STL-layer, we use the STL Computation graph (STL-CG) [9] to construct and calculate the robustness. Gradient ascent is used in the forward rollout steps (Equations (2) and (4) in the paper) with a learning rate of 0.005 (number of rollout steps is set to 10). As is included in the STL-CG framework, a soft version of robustness is used (with the softmax approximation). In our experiments, we use a softmax scale of 0.3. Finite difference is used to calculate the gradient of the robustness with respective to the input trajectory (in the forward pass). Backpropagation is performed using Pytorch's autograd engine.

4 Additional Results

4.1 Results With Trajectron++ As The Base Predictor

In this section, we provide additional results with Trajectron++ [7] as the base predictor. We show the comparison histogram as well as the prediction horizon study in Figure 1 . In the top figure, we compare the performance metric histograms before and after applying the STL layer corrections. In regions with high base prediction errors, the STL layer is able to correct for the deviations using the defined rules. However, in regions with low base prediction errors, the STL layer suffers from over-correction which increases the prediction error. We will discuss failure cases in more detail in the next section. Figure 1 (bottom) shows the level of STL correction at 3 different prediction horizons (6, 9, 12 seconds). The STL layer is able to improve the base prediction results in all three cases.

4.2 Example Scenes

In this section, we show example scenes (both successes and failures) of predictions pre and post correction. In all the scenes, the dot-dash lines represent the closest lanes to the vehicle (green) with brown indicating the incoming lane, yellow indicating the current lane and white indicating outgoing lanes. The green vehicle is the vehicle to be predicted. The dots represent positions on the predicted trajectories with black indicating ground truth, white indicating the base prediction and red indicating the corrected prediction.

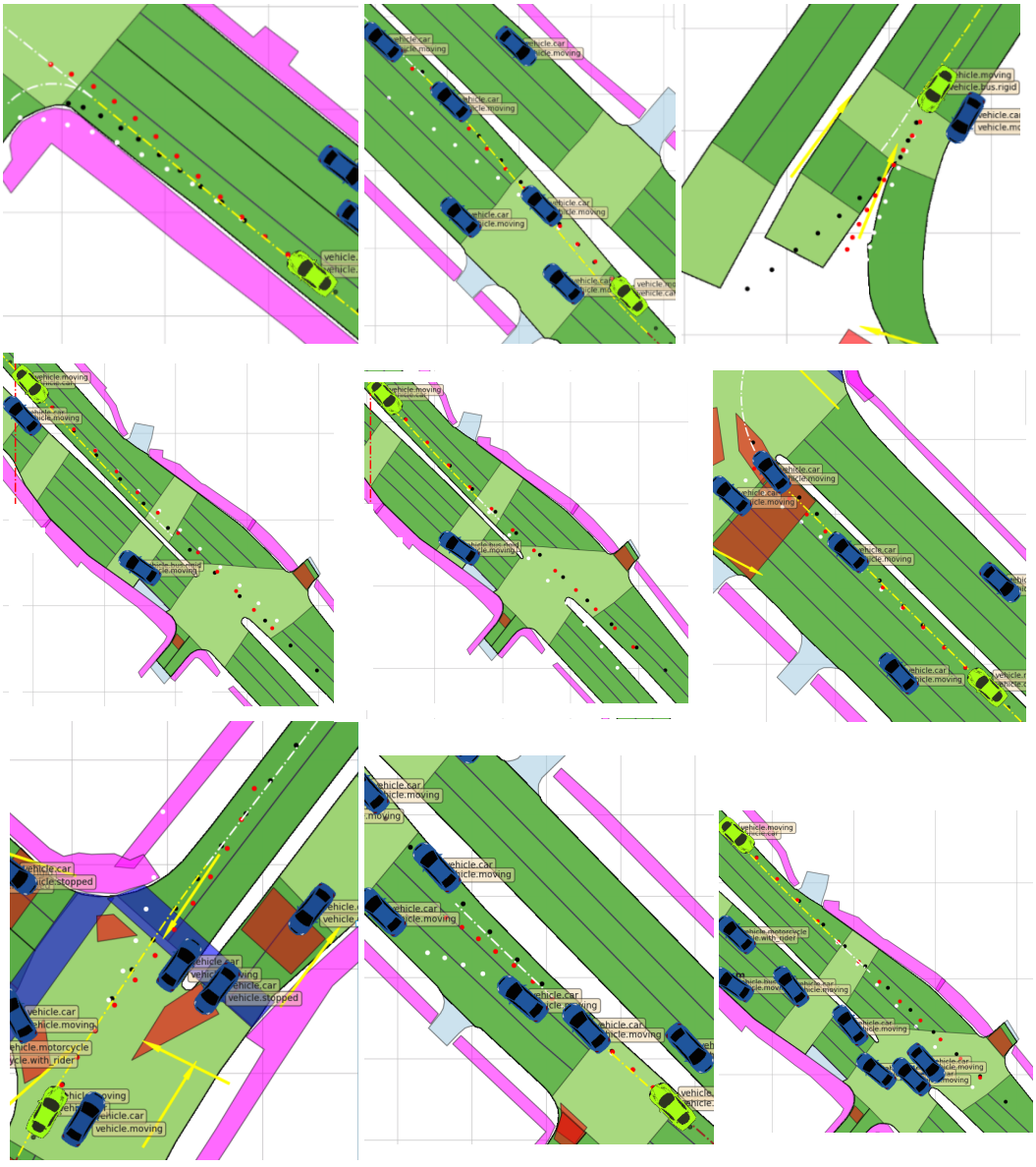

Figure 2 : Example scenes for successful correction. Dot-dash lines represent closest lanes to the vehicle to be predicted with brown indicating the incoming lane, yellow indicating the current lane and white indicating outgoing lanes. The green vehicle is the vehicle to be predicted. The dots represent positions on the predicted trajectories with black indicating ground truth, white indicating the base prediction and red indicating the corrected prediction.

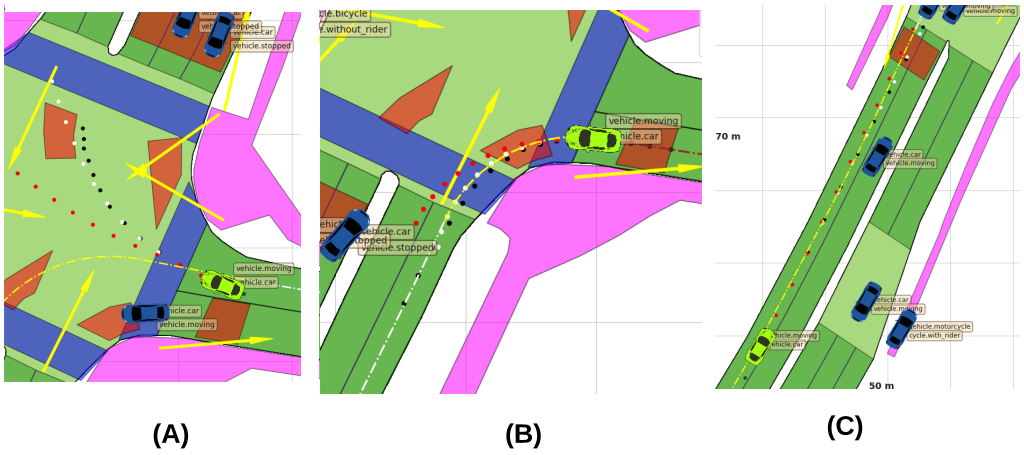

Figure 3 : Failure cases. (**A**): Failure due to in accurate map data. (**B**): Failure due to lane representation insufficiency. (**C**): Failure due to over-correction.

Recall that the rules we defined encourages the vehicle to (1) stay close to the base prediction, (2) stay close to the current center lane, (3) if there are no stop areas 30 meters ahead, stay close to the current observed speed. Figure 2 shows sample scenes of successful cases where the corrected prediction (red) has higher accuracy than the base prediction (white). Overall, rule (2) played the most important role in the success of the correction. Rule (3) helps in cases where the base predictor falls short in predicting the speed of the vehicle. In the bottom left scene, the base predictor provided a trajectory that is moving slower than what the vehicle is actually moving at (white trajectory travels a shorter distance than black). The corrected trajectory compensated for this shortage using rule (3). Rule (1) is necessary in cases where the base predictor captures information that the rules don't, it also gives the STL layer a reasonable starting point.

Figure 3 shows three representative failure modes. In all three cases, the prediction after correction performs worse than the base prediction. In Figure 3 (A), the failure is due to the map not providing the correct center lane. NuScenes provides a function that for each input position on the map outputs the closest center lane as a set of discretized poses. In this case, the agent is trying to make a right turn at the intersection, however, the lane resulted from the query is a left turning lane. This results in the corrected trajectory to lean towards the wrong lane. This failure mode can be partially addressed by comparing the queried center lane with the base prediction, if a large discrepancy is found then relax rule (2) at that time-step.

Figure 3 (B) shows a failure mode due to our lane representation. We calculate the distance between the vehicle and the lane by first fitting a line through the discrete representation of lane and then compute the point to line distance (this method is chosen for simplicity). A first order line fitting does not work well when the lane has a high curvature. This failure mode can be addressed by fitting the lane with higher order polynomials and calculating the approximate point to polynomial distance.

Figure 3 (C) shows a failure mode due to over correction. In this case, the vehicle is simply not driving on the center lane and the base predictor captured this behavior. However, applying the STL correction results in a trajectory that's closer to the center lane but with increased error (over-correction accounts for the behavior in Figure 1(top) at low base prediction error regions). This is a common problem when multiple constraints are active and the correct balance is to be achieved. A potential solution is to make the rule parameters conditioned on environmental information such that each rule is enforced with different strength in different scenarios (which are learned from data).

References

[1] G. E. Fainekos and G. J. Pappas. Robustness of temporal logic specifications for continuous-time signals. Theoretical Computer Science, 410(42):4262–4291, 2009.

[2] A. Donzé, T. Ferrere, and O. Maler. Efficient robust monitoring for STL. In Computer Aided Verification, pages 264–279. Springer, 2013.

[3] A. Dokhanchi, B. Hoxha, and G. Fainekos. 5th International Conference on Runtime Verification, Toronto, ON, Canada., chapter On-Line Monitoring for Temporal Logic Robustness, pages 231–246. Springer, 2014. ISBN 978-3-319-11164-3. doi:10.1007/978-3-319-11164-3_19.

[4] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. ArXiv, abs/1903.11027, 2019.

[5] N. Deo and M. M. Trivedi. Multi-modal trajectory prediction of surrounding vehicles with maneuver based lstms. Intelligent Vehicles Symposium (IV), Jun 2018.

[6] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. Nuscenes devkit. https://github.com/nutonomy/nuscenes-devkit, 2020.

[7] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone. Trajectron++: Multi-agent generative trajectory forecasting with heterogeneous data for control. ArXiv, abs/2001.03093, 2020.

[8] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone. Trajectron-plus-plus. https://github.com/StanfordASL/Trajectron-plus-plus, 2020.

[9] K. Leung, N. Aréchiga, and M. Pavone. Backpropagation for parametric stl. 2019 IEEE Intelligent Vehicles Symposium (IV), pages 185–192, 2019.