FLoRA: A Framework for Learning Scoring Rules in Autonomous Driving Planning Systems

Zikang Xiong, Joe Eappen, and Suresh Jagannathan

***Abstract*— In the pursuit of safer and more efficient autonomous driving systems, scoring of motion plans plays a crucial role. This work introduces a novel framework for learning interpretable scoring rules represented in temporal logic. Our method features a learnable logic structure that represents nuanced relationships in diverse driving scenarios. By employing a data-driven, fully differentiable framework, we optimize the rules and parameters directly from driving demonstrations. Our approach also overcomes the limitation of single-class training data, distinguishing desirable behaviors without explicit negative examples. Evaluations in closed-loop planning simulations demonstrate that our learned scoring rules outperform existing techniques, including expert-crafted rules and neural network scoring models, while maintaining interpretability. This work contributes a versatile, data-driven approach to enhance the scoring mechanism in autonomous driving systems, which is applicable to various motion planning methods. Our video and code are available on xiong.zikang.me/FLoRA/.**

I. Introduction

Autonomous vehicles face the complex task of navigating through dynamic environments safely and efficiently. At the heart of this challenge lies the process of motion planning - determining the best path for the vehicle to take. However, generating a single optimal path is often not enough. Instead, modern autonomous driving systems typically produce multiple potential plans [1]–[4], which then need to be evaluated and refined. This is where scoring and selection of motion plans come into play. These techniques act as a crucial filter, assessing the quality of each proposed path and choosing the most suitable one for execution. By applying additional evaluation steps after the initial plan generation, we can significantly enhance the safety, efficiency, and overall performance of autonomous vehicles. The importance of effective scoring becomes even more apparent in complex autonomous driving systems, particularly in end-to-end approaches. These systems often utilize large, intricate neural networks that incorporate elements of randomness, such as dropout or sampling from probability distributions. While powerful, these characteristics can make it challenging to predict the system's behavior consistently. By implementing interpretable scoring rules, we introduce a layer of predictability and reliability to these complex systems. These rules act as a safeguard, evaluating the generated motion plans against clear, understandable criteria. This additional step helps to mitigate the uncertainties inherent in complex planning systems, providing a more robust and trustworthy framework for autonomous vehicle decision-making. In essence, scoring techniques serve as a critical bridge, connecting the raw outputs of planning algorithms to the final, executable plans that guide autonomous vehicles through our highways and cities.

Authors are with the Computer Science Department, Purdue University. {xiong84,jeappen,suresh}@cs.purdue.edu

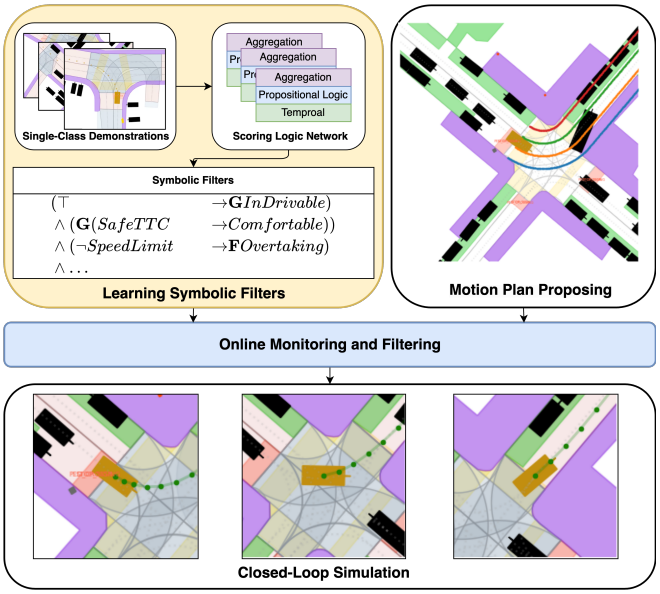

Fig. 1. Learning and applying scoring rules represented in temporal logic. The **Learning Scoring Rules** block, which is the key contribution of our work, shows a Scoring Logic Network (SLN) that learns from demonstration data on good behaviors. We can extract and simplify symbolic rules from this network. Here we show three of the rules learned by the SLN: (1) $\top \rightarrow \mathbf{G}\ \mathit{InDrivable}$ (always stay in the drivable area, e.g., lane, intersection); (2) $\mathbf{G}(\mathit{SafeTTC} \rightarrow \mathit{Comfortable})$ (if the vehicle has safe Time-To-Collision (TTC) with surrounding vehicles, it should always be subject to all comfort constraints); (3) $\neg \mathit{SpeedLimit} \rightarrow \mathbf{F}\ \mathit{Overtaking}$ (the vehicle should not exceed the speed limit unless it is overtaking another vehicle). The **Motion Plan Proposing** block shows 4 motion plans colored in different colors, generated by PDM's proposer [1]. The **Online Monitors and Scoring** block monitors and scores the proposed plans in 20 Hz, with the learned rules. The green plan receives the highest score by satisfying all rules.

We focus on learning scoring rules represented in temporal logic for evaluating autonomous driving plans, which assess and rank plans generated by motion plan proposers. These scoring rules capture the latent relationships between various driving rules and constraints; for example, if a vehicle has a safe time-to-collision with surrounding vehicles, it should always be subject to all comfort constraints. By applying these rules to the output of a motion planner, we can score and select desirable plans, ensuring that the planned paths adhere to safety standards and traffic regulations while maintaining optimal performance. Figure 1 illustrates the learning process and how we might apply scoring rules.

In practice, building these scoring rules presents several significant challenges. First, *the latent relationships and dependencies among various rules are often non-trivial*. For instance, while a vehicle is generally not permitted to exceed the speed limit, exceptions may exist in specific scenarios such as overtaking another vehicle. These nuanced dependencies make it challenging to create a comprehensive set of rules that account for all possible situations. Second, *determining the optimal parameters for rules is a complex task*. For example, establishing appropriate thresholds for safe time-to-collision, comfortable acceleration, or acceptable steering angle requires careful consideration of multiple factors. These parameters must balance safety concerns with the need for efficient and smooth vehicle operation. Third, *available demonstration data typically only showcases correct behavior and lacks sufficient examples of incorrect actions*. Having only single-class training data poses a significant challenge for machine learning algorithms, as they must learn to distinguish between acceptable and unacceptable behaviors without a sufficient number of explicit negative samples.

Our approach addresses these challenges through three interconnected key ideas. First, to capture latent relationships among driving rules, we introduce a learnable logic structure that seamlessly integrates temporal and propositional logic. Our structure can represent nuanced decisions such as when it is appropriate to exceed the speed limit for a safe overtaking maneuver. Second, we tackle the challenge of parameter optimization by letting the data speak for itself. Rather than relying on manual tuning, our system learns optimal rule parameters directly from driving demonstrations, leveraging the fully differentiable logic structure used to represent rules. Third, we overcome the limitation of learning from only positive examples through a novel regularization-constrained optimization framework that simultaneously rewards correct demonstration behaviors and restricts the space of acceptable ones.

We evaluate the efficacy of our learned scoring rules in NuPlan [5] closed-loop simulations. These results demonstrate that our learned rules can effectively score and select desirable plans, outperforming both expert-crafted rules and neural network-based approaches when considering both interactive and non-interactive scenarios. We further show that the learned rules perform consistently well across different proposers, including PDM [1], PDM-Hybrid [1], ML-Prop [5], and a rule-based acceleration-time sampler [4]. In summary, our contributions are as follows:

- We propose a novel learnable logic structure that discovers and captures latent relationships among candidate driving rules, represented in temporal logic.
- We introduce a data-driven approach to optimize rule parameters, enabling the system to learn effective scoring rules from driving demonstrations.
- We present an optimization framework that allows the system to learn from single-class data, enabling it to distinguish between acceptable and unacceptable behaviors in the scoring process.
- We demonstrate the effectiveness of our approach in scoring and selecting desirable plans via NuPlan closed-loop simulation, outperforming expert-crafted rules and neural network-based approaches across various scenarios and proposers.

II. Related Work

Scoring models are widely used in autonomous driving systems, which evaluate the safety, efficiency, and comfort of an autonomous driving system. Existing approaches can be broadly categorized into rule-based and learning-based models. Rule-based scoring models [1], [2], [5]–[13] leverage expert knowledge and domain-specific rules, offering interpretability but lacking data-driven adaptability. In contrast, learning-based scoring models, typically approximated with deep neural network, [3], [4], [14]–[19] are data-driven and can capture complex patterns, but often sacrifice interpretability, which can be intimidating for safety-critical applications. Several works [16]–[19] have employed inverse-reinforcement learning for scoring model development, learning from single-class demonstrations. However, these approaches either require expensive closed-loop simulation [19] or additional samplers to generate non-ideal demonstrations [16]–[18], while also producing uninterpretable scoring models. These challenges underscore the need for scoring models that balance adaptability and interpretability in autonomous driving systems.

In the broader context of temporal logic rule learning [20], some researchers have explored learning interpretable rules from demonstrations. For instance, [21] proposed a backpropagation-based approach to learn temporal logic specifications from demonstrations. However, this approach is designed only to learn the parameters of the logic formula, not the structure. [22] developed an approach capable of learning both the structure and parameters of logic formulas, but it relies on heuristic search with manually designed selection criteria, which may not generalize well across domains. Lastly, none of these works have been applied to mining autonomous driving rules or validated on large-scale real-world datasets such as NuPlan [5].

Current literature has not fully explored the potential of learning interpretable rules directly from driving data. Our work addresses this gap by proposing a novel approach that learns interpretable rules and their parameters from single-class demonstrations through a regularization-constrained optimization framework, which avoids additional samplers or expensive closed-loop simulations.

III. Preliminaries

We formulate the key technical components and our objective in this section.

a) Predicate P_θ: At a certain time point, given all environment information $\mathcal{E}$ (e.g., map, traffic light state, current and history states of all the agents in the scene) and motion plan τ, the differentiable predicate P_θ is defined as: $P_\theta : (\mathcal{E} \times \tau) \rightarrow [-1, 1]$, which evaluates driving conditions

and the ego[1] car's motion plan. It maps $\mathcal{E}$ and τ to a truth confidence value in $[-1, 1]$, where θ are the predicate parameters. When designing the predicate, we ensure that the gradient $\nabla_\theta P_\theta$ exists and can be computed. The sign of P_θ indicates truth. $P_\theta < 0$ implies False; $P_\theta > 0$ implies True. The absolute value $|P_\theta|$ indicates the degree of confidence. Our work is evaluated on NuPlan [5], a state-of-the-art autonomous driving planning benchmark. $\mathcal{E}$ and τ follow the definition in [5]. As a concrete example, predicate $SafeTTC(\mathcal{E}, \tau)$ in Fig. 1 checks if the ego car has a safe time-to-collision to surrounding cars. This can be formulated as: $SafeTTC(\mathcal{E}, \tau) = \tanh(TTC_{min}(\mathcal{E}, \tau) - \theta)$, where $TTC_{min}(\mathcal{E}, \tau)$ is the minimum time-to-collision between the ego car and any surrounding vehicle, and θ is the differentiable safety threshold. The tanh function ensures differentiability and bounds the output between -1 and 1. A positive value indicates a safe TTC (i.e., $TTC_{min} > \theta$), while a negative value indicates an unsafe TTC. Similar to most existing work [20], we explicitly design the predicates and focus this paper on learning logical connections and parameters assuming a given set of predicates. With the predicate defined, we can now move on to discussing how these predicates are combined into logical formulas.

b) Formula $\mathcal{L}$: Given a differentiable predicate set $\mathcal{P} = \{P^1_{\theta_1}, P^2_{\theta_2}, \ldots, P^n_{\theta_n}\}$, we introduce a LTL_f logic space [23] that includes compositions of predicates from $\mathcal{P}$ and logic operators. The logic formula $\mathcal{L}$ can be generated from the following grammar:

$$\mathcal{L} := P_\theta \mid \mathbf{G}\ \mathcal{L} \mid \mathbf{F}\ \mathcal{L} \mid \neg\mathcal{L} \mid \mathcal{L} \wedge \mathcal{L}' \mid \mathcal{L} \vee \mathcal{L}' \tag{1}$$

where $P_\theta \in \mathcal{P}$ is a differentiable predicate, **G** and **F** are temporal operators representing "globally" and "finally" respectively, $\neg$ is logical negation, $\wedge$ is logical and, and $\vee$ is logical or. Like most existing work [20], the strong "Until" ($\mathcal{L}\ \mathbf{U}\ \mathcal{L}'$) is not included because it can be represented using existing logic operators ($\mathbf{F}\ \mathcal{L}' \wedge \mathbf{G}(\mathcal{L} \vee \mathcal{L}')$). Having established the syntax for our logic formulas, we now need a way to evaluate them quantitatively.

c) Quantitative Evaluation of Formula: Given a finite input sequence $S = \{(\mathcal{E}_t, \tau_t)\}_{t=0}^{T}$ sampled at different time points, up to a bounded time T, we can evaluate the logic formula $\mathcal{L}$ quantitatively using a set of min and max operators[24], [25]. This evaluation maps the sequence S to a value in $[-1, 1]$, denoted as $\mathcal{L}(S; \boldsymbol{\theta}) \to [-1, 1]$. Here, $\boldsymbol{\theta}$ represents all the predicates' parameters. Specifically, we define the quantitative evaluation of an atomic predicate P_θ at time t as $P_\theta(\mathcal{E}_t, \tau_t) \in [-1, 1]$. In (2), the temporal logic operators **G** and **F** evaluate the formula $\mathcal{L}$ over the entire sequence from time t onwards. $\mathbf{G}\mathcal{L}$ (globally) returns the minimum value of $\mathcal{L}$ over all future time points, which ensures the property holds throughout the sequence if $\mathbf{G}\mathcal{L}$ evaluates to a positive value. $\mathbf{F}\mathcal{L}$ (finally) returns the maximum value, indicating the property is satisfied at least once in the future.

[1]The ego car refers to the vehicle being controlled in a driving scenario.

The evaluation function ρ is defined as:

$$\rho(\mathbf{G}\mathcal{L}, t) = \min_{t' \geq t} \rho(\mathcal{L}, t') \qquad \rho(\mathbf{F}\mathcal{L}, t) = \max_{t' \geq t} \rho(\mathcal{L}, t') \tag{2}$$

For single time point evaluation, the logical operators and ($\wedge$), or ($\vee$), and not ($\neg$) are defined using min, max, and negation operations:

$$\begin{aligned} \rho(\mathcal{L} \wedge \mathcal{L}', t) &= \min\{\rho(\mathcal{L}, t), \rho(\mathcal{L}', t)\} \\ \rho(\mathcal{L} \vee \mathcal{L}', t) &= \max\{\rho(\mathcal{L}, t), \rho(\mathcal{L}', t)\} \\ \rho(\neg\mathcal{L}, t) &= -\rho(\mathcal{L}, t) \end{aligned} \tag{3}$$

All the operations defined by ρ are differentiable, which allows us to backpropagate through. In practice, we use softmin and softmax to approximate min and max operators for a smooth gradient [21]. With the evaluation framework in place, we can now define our overall objective for learning optimal driving rules. For simplicity, we define $\rho(\cdot) := \rho(\cdot, 0)$, meaning evaluate from the initial of input sequence.

d) Objective: Our objective is twofold: (1) learn the optimal logic formula $\mathcal{L}^*$, and (2) optimize the parameters $\boldsymbol{\theta}$ of the predicates, which characterize the demonstration data accurately. Formally, we aim to solve the following problem:

$$\mathcal{L}^*, \boldsymbol{\theta}^* = \underset{\mathcal{L} \in \Omega_\mathcal{P}, \boldsymbol{\theta}}{\operatorname{argmax}} \mathbb{E}_{S \sim \mathcal{D}^+}[\mathcal{L}(S; \boldsymbol{\theta})] \tag{4}$$

where $\mathcal{D}^+$ represents driving demonstrations, which consist *solely* of correct demonstrations that represent ideal driving behaviors.

IV. Approach

This section presents our approach to learning interpretable driving rules from demonstrations. We begin by introducing the concept of condition-action pairs in Sec. IV-A, which forms the foundation of our rule representation. Sec. IV-B introduces the core of our method: the learnable logic structure. Here, we explain its components, analyze its capabilities, and describe how we extract and simplify rules from it. Finally, Sec. IV-C addresses the challenge of learning from single-class demonstrations, introducing our novel regularization techniques to overcome the limitations of positive-only examples.

A. Condition-Action Pair

We consider rules that consist of conditions and expected actions. For instance, one such rule might require that the ego car eventually stop when approaching a stop sign. This pattern of condition-action pairing extends to countless driving situations. Thus, we focus on effectively learning and reducing driving rules to condition-action pairs.

Predicates are the basic unit of our rules. We categorize our predicates into two types: condition predicates $\bar{\mathcal{P}} = \{\bar{P^1_{\theta_1}}, \bar{P^2_{\theta_2}}, \ldots, \bar{P^n_{\theta_n}}\}$ and action predicates $\dot{\mathcal{P}} = \{\dot{P^1_{\theta_1}}, \dot{P^2_{\theta_2}}, \ldots, \dot{P^m_{\theta_m}}\}$. Condition predicates evaluate traffic conditions (e.g., if approaching a stop sign), while action

predicates assess the motion plan (e.g. if the ego car is stopped). Given

$$\begin{aligned} condition &:= \bar{P}_\theta \mid \mathbf{G}\ condition \mid \mathbf{F}\ condition \mid \neg condition \\ action &:= \dot{P}_\theta \mid \mathbf{G}\ action \mid \mathbf{F}\ action \mid \neg action \end{aligned} \tag{5}$$

we extract and simplify the learned logic formula $\mathcal{L}$ to a set of propositional rules of the form:

$$\bigodot_{i=1}^{m} \left(\bigwedge_{j=1}^{n} condition_j \rightarrow action_i \right) \tag{6}$$

where $\bigodot$ denotes that this condition-action pairs are connected by $\wedge$ or $\vee$ operators. The conjunction of conditions ($\bigwedge_{j=1}^{n} condition_j$) allows for more precise and specific criteria to be defined for each action, thereby describing precisely when the action should be allowed. Building upon this foundation, we introduce a learnable logic structure to represent and learn these condition-action pairs.

B. Learnable Logic Structure

The learnable logic structure $\bar{\mathcal{L}}$ is a directed acyclic computation graph that represents a compositional logic formula. It consists of three types of layers: Temporal, Propositional, and Aggregation. These layers are interconnected through learnable gates that determine the flow and combination of logical operations. An example of this structure is shown in Fig. 2.

1) Layer Definition: In the lowest block of Fig. 2, the frame batch is processed through $\mathcal{P}$ and then passed through Temporal layers. Let $F = \{f_1, f_2, ..., f_T\}$ be a sequence of T frames, where each frame represents a time point. Define $\mathcal{P} = \{P^1_{\theta_1}, P^2_{\theta_2}, ..., P^N_{\theta_N}\}$ as a set of N predicates. For each predicate $P^i_{\theta_i} \in \mathcal{P}$ and each frame $f_t \in F$, we compute $X^t_i = P^i_{\theta_i}(f_t)$. Let $X^T_i = [x^1_i, x^2_i, ..., x^T_i]$ be the sequence of predicate values for predicate $P^i_{\theta_i}$ across all time steps. The output of the predicates is then defined as $\mathcal{X}^T = \{X^T_1, X^T_2, ..., X^T_N\}$, where each $X^T_i \in \mathbb{R}^T$ contains the predicate values computed over the entire time sequence for the i-th predicate. These frame sequences can be batched as shown in Fig. 2, to simplify the symbols, we only discuss the case with batch size one in the following parts.

The **Temporal** layer $\mathcal{T}$ operates on each element $X^T_i \in \mathcal{X}^T$, potentially applying a temporal operator. Formally, the output of $\mathcal{T}$ is $O^T = \mathcal{T}(\mathcal{X}^T) = \{o^T_1, o^T_2, \ldots, o^T_N\}$, where $o^T_i = \mathcal{T}(x^T_i) \in \{\mathbf{G}X^T_i, \mathbf{F}X^T_i, X^T_i\}$. Temporal layers can be stacked, allowing for the composition of temporal operators. For instance, with two stacked temporal layers, we could have $\mathcal{T}_2(\mathcal{T}_1(X^T_i)) = \mathbf{F}(\mathbf{G}(X^T_i))$. This composition allows for expressing more nuanced and complex temporal properties.

The **Propositional** logic layer $\mathcal{F}$ operates on its input set O^T, generating $\binom{N}{2}$ clusters, each containing a combination of two inputs connected by a logical operator. The behavior of $\mathcal{F}$ is formalized as $\mathcal{F}(O^T) = O^P = \{o^P_1, o^P_2, \ldots, o^P_{\binom{N}{2}}\}$, where $o^P_i = (\neg)o^T_j \circ (\neg)o^T_k$, $\circ \in \{\wedge, \vee\}$, and $O^T = \{o^T_1, o^T_2, \ldots, o^T_N\}$ is the output of a Temporal layer. Here, j and k represent the indices of the two different inputs being combined. Each input can be negated or unchanged when passing through a logic layer. We do not stack the Propositional layer as it would lead to exponential growth in the number of clusters, and aggregating on one Propositional layer can represent any formula in the form of (6) as proved in IV-B.3.

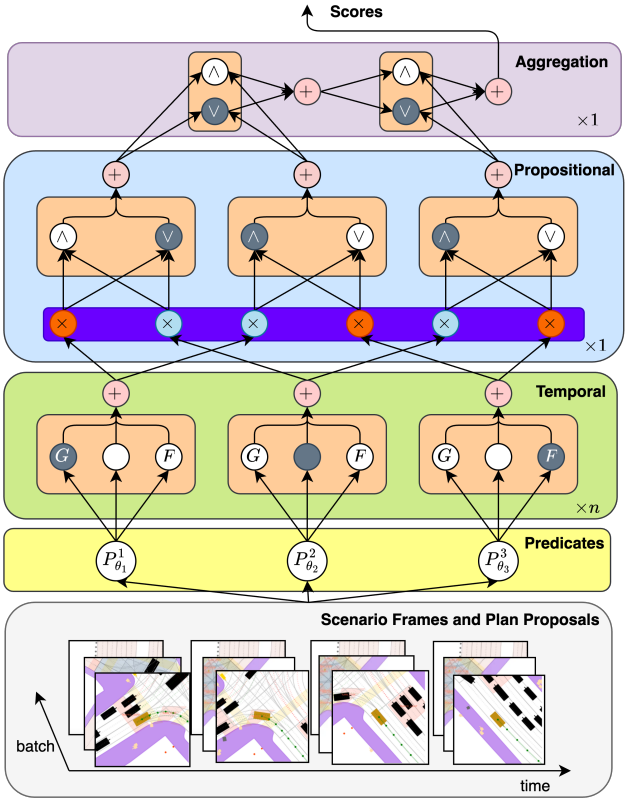

Fig. 2. The logic structure $\bar{\mathcal{L}}$ consists of three types of layers: Temporal, Propositional, and Aggregation. The Temporal layer processes the initial predicates, applying temporal operators. The Propositional layer generates all possible pairs of predicates connected by logical operators. The Aggregation layer aggregates the output of the Propositional layer into one cluster by deciding the logic operator to connect neighboring clusters. The Temporal layers can be stacked. Layer's formal definition is in Sec. IV-B.1. Two types of gates, the selection gate, and the negation gate, are used to control the logic operators and the sign of the cluster inputs, respectively. Each clear circle (◯) in these gates represents a single value *weight*. In the selection gate, the ● circle represents the operator with the largest weight, meaning the operator is selected. In the negation gate, the ● circle represents the negation of the input (i.e., multiply with a negative number), while the ● circle represents the original input (i.e., a positive number). The gate implementation is in Sec. IV-B.2. Supposing we only consider one layer of Temporal layer ($n = 1$), and given a set of predicates $\mathcal{P} = \{P^1_{\theta_1}, P^2_{\theta_2}, P^3_{\theta_3}\}$, $P^2_{\theta_2} \in \bar{\mathcal{P}}$ and $P^1_{\theta_1}, P^3_{\theta_3} \in \dot{\mathcal{P}}$, this learnable logic structure represents the logic formula $(\mathbf{G}P^1_{\theta_1} \vee \neg P^2_{\theta_2}) \vee (\neg\mathbf{G}P^1_{\theta_1} \wedge \mathbf{F}P^3_{\theta_3}) \vee (\neg P^2_{\theta_2} \wedge \mathbf{F}P^3_{\theta_3})$ This formula can be further reduced to $P^2_{\theta_2} \rightarrow (\mathbf{G}P^1_{\theta_1} \vee \mathbf{F}P^3_{\theta_3})$.

The **Aggregation** layer aggregates the output of the Propositional (O^P) layer into one cluster by deciding the logic operator from $\{\wedge, \vee\}$ to connect neighboring clusters. Formally, given the input O^P, the output of the Aggregation layer can be represented as $\mathcal{A}(O^P) = o^P_1 \circ_1 o^P_2 \circ_2 \ldots \circ_{\binom{N}{2}-1} o^P_{\binom{N}{2}}$, where $\circ$ represents $\wedge$ or $\vee$.

A logic structure can be formally defined as $\bar{\mathcal{L}} = \mathcal{A}(\mathcal{F}(\mathcal{T}^{\times n}(\mathcal{P})))$, where n is the number of stacked Temporal layers.

2) Gate Implementation: The layers in Fig. 2 are composed of selection gates and negation gates. Each ◯ in these

gates represents a weight $w \in \mathbb{R}$.

The selection gate acts as a soft attention mechanism to select between different operators across all layers (i.e., $\mathbf{G}, \mathbf{F}$ and identity operator in the Temporal layer, $\wedge, \vee$ in the Propositional and Aggregation layers), defined as:

$$g_s(O) = \sigma([w_s^1; \cdots; w_s^k]) \cdot [o_1; \cdots; o_k]^T \tag{7}$$

where $\sigma(\cdot)$ denotes the softmax function. For temporal operators in the Temporal layer, $O = [\rho(\mathbf{G}X_i^T), \rho(\mathbf{F}X_i^T), \rho(X_i^T)]^\top \in \mathbb{R}^3$ and $k = 3$. For logic operators in the Propositional and Aggregation layers, $O = [\rho(o_1 \wedge o_2), \rho(o_1 \vee o_2)]^\top \in \mathbb{R}^2$ and $k = 2$. The evaluation function ρ is defined in (2) and (3). The selection gate chooses between operators across layers by computing a weighted sum, where larger weights in the softmax output correspond to higher selection probabilities for their associated operators. This selection mechanism is soft, allowing for a continuous blend of different operators rather than a hard, discrete choice.

The negation gate, given by $g_n = \tanh(w_{neg}) \cdot x$, with learnable parameters w_{neg}, controls the sign of cluster x inputs to the Propositional layer.

$$g_n = \tanh(w_{neg}) \cdot x, \quad w_{neg} \in \mathbb{R}, x \in \mathbb{R} \tag{8}$$

The tanh function constrains the output to $[-1, 1]$. According to the quantitative semantics defined in (3), multiplying by a negative value negates the input. The gradient properties of tanh encourage w_{neg} to converge towards either -1 or 1 during training, effectively learning whether to negate the input.

3) Logic Space Analysis: Stacking Temporal layers monotonically increases the logic space. Creating a logic space that contains up to n nested temporal operators can easily achieved by stacking n Temporal layers. For the Propositional layer, we assert that:

Theorem 1: Aggregating the output from a single Propositional layer can represent any formula in the form of (6).

Proof: Consider a Propositional layer with inputs $condition_1, \ldots, condition_n, action_1, \ldots, action_m$. For each $action_i$, combine:

$$(\neg condition_1 \vee action_i) \wedge \ldots \wedge (\neg condition_n \vee action_i)$$

This is equivalent to $(condition_1 \wedge \ldots \wedge condition_n) \rightarrow action_i$. Aggregating for all $action_i$ yields:

$$\bigodot_{i=1}^{m} ((condition_1 \wedge \ldots \wedge condition_n) \rightarrow action_i)$$

where $\bigodot$ is either $\wedge$ or $\vee$, matching the theorem's form. ■

This theorem is significant for our learnable logic structure as it establishes that we can express all necessary condition-action pairs without the need for multiple Propositional layers. This result justifies our design choice of using a single Propositional layer, which helps to keep the structure computationally efficient while maintaining full expressiveness.

4) Ensembling: Given a set of logic structures $\{\bar{\mathcal{L}}_1, \bar{\mathcal{L}}_2, \ldots, \bar{\mathcal{L}}_k\}$, we can ensemble them by aggregating their outputs with an additional Aggregation layer. Formally,

$$\bar{\mathcal{L}}_{\text{ensemble}} = \mathcal{A}(\bar{\mathcal{L}}_1, \bar{\mathcal{L}}_2, \ldots, \bar{\mathcal{L}}_k) \tag{9}$$

In practice, we noticed that ensembling multiple logic structures can improve the robustness of the learned rules.

5) Rule Extraction and Simplification: The rule extraction and simplification process transforms the learned logic structure into interpretable condition-action pairs. To interpret the learned logic structure $\bar{\mathcal{L}}$, we extract its logic formula $\mathcal{L}$ by traversing the structure and selecting the most probable operators by concretizing the selection gates and negation gates. Given a selection gate g_s, let $C(\cdot)$ denote the concretizing function:

$$C(g_s) = \underset{op}{\text{argmax}}\, w_s^{op}, \quad op \in \begin{cases} \{\mathbf{G}, \mathbf{F}, \text{id}\}, \\ \{\wedge, \vee\}, \end{cases} \tag{10}$$

where w_s^{op} represents the weights associated with each operator in the selection gate. For the negation gate g_n, the concretizing is determined by the sign of the weight:

$$C(g_n) = \begin{cases} \text{negation}, & \text{if } w_{neg} < 0 \\ \text{original}, & \text{otherwise} \end{cases} \tag{11}$$

A concrete example is shown in Fig. 2 by iteratively applying (10) and (11) to the selection and negation gates. For the ensembling logic structure, we only need to apply the same rule further for the additional Aggregation layer.

The extracted formula is then simplified to a set of condition-action pairs in the form of (6). We apply the Quine-McCluskey algorithm [26] to simplify the extracted formula. Such simplification removes redundant cluster (e.g., $\wedge(P_{\theta_1}^1 \vee \neg P_{\theta_1}^1)$). The resulting formula is then converted to conjunctive normal form: $\bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{m} \bar{P}_j \vee \bigvee_{k=1}^{l} \dot{P}_k \right)$, where $\bar{P}_j$ and $\dot{P}_k$ represent condition and action predicates, respectively. By double negating the condition predicates and applying De Morgan's laws, this formula can be further simplified to condition-action pairs in the form of (6).

The extracted formula is not necessary to be identical to the original formula, because soften operators are used to approximate the logic operators. However, in practice, we find the extracted formula is close to the original one (0.051 mean absolute error over all scenarios). More importantly, the scoring rule we learned will be used to rank different motion plans in closed-loop evaluation. We noticed that such approximation almost does not affect the ranking.

C. Learning From Single-Class Demonstration

Most demonstration datasets consist solely of examples representing ideal driving behaviors. Our goal is to learn the logic structure $\bar{\mathcal{L}}^*$ [2] and predicate parameters $\boldsymbol{\theta}^*$ that grade demonstrations with the highest score and unseen behaviors with lower scores. Directly optimizing on (4) would simply

[2] $\bar{\mathcal{L}}$ means the learnable logic structure, while $\mathcal{L}$ means a concrete logic formula defined by (1).

make the score $\bar{\mathcal{L}}(S;\boldsymbol{\theta})$ to be 1 for all demonstrations. The optimizer could find "shortcuts" and result in learning trivial formulas like $P^1_{\theta_1} \vee \neg P^1_{\theta_1}$, which are always true and do not provide meaningful rules. To address this issue, we introduce two regularization terms for $\boldsymbol{\theta}$ and $\bar{\mathcal{L}}$. The full learning algorithm is presented in Algorithm 1.

Algorithm 1: Training with Regularization

Input: Dataset $\mathcal{D}^+$, initial logic structure $\bar{\mathcal{L}}$, initial parameters $\boldsymbol{\theta}$, learning rates α, β, max weight $w_{\max}$, batch size B, number of epochs E
Output: Optimized $\bar{\mathcal{L}}^*$ and $\boldsymbol{\theta}^*$

Function `Update`$(\bar{\mathcal{L}}, \boldsymbol{\theta}, \mathcal{J}, \gamma)$**:**
  Update $\boldsymbol{\theta}$ with $\nabla_{\boldsymbol{\theta}}\mathcal{J}$;
  Update $\bar{\mathcal{L}}$ (i.e., gate weights) with $\nabla_{\bar{\mathcal{L}}}\mathcal{J}$;
  return $\bar{\mathcal{L}}, \boldsymbol{\theta}$;
Function `Regularize`$(\bar{\mathcal{L}}, \boldsymbol{\theta}, \alpha, \beta, w_{\max})$**:**
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \text{sign}(\frac{\partial \bar{\mathcal{L}}}{\partial \boldsymbol{\theta}})$;
  foreach *Aggregation layer in* $\bar{\mathcal{L}}$ **do**
    $w_\wedge \leftarrow \min(w_\wedge + \beta, w_{\max})$;
  return $\bar{\mathcal{L}}, \boldsymbol{\theta}$;
for *epoch* $\leftarrow 1$ **to** E **do**
  for *each batch* $\{S_1, \ldots, S_B\} \sim \mathcal{D}^+$ **do**
    $\mathcal{J} \leftarrow \frac{1}{B}\sum_{i=1}^{B} \bar{\mathcal{L}}(S_i;\boldsymbol{\theta})$;
    $\bar{\mathcal{L}}, \boldsymbol{\theta} \leftarrow$ `Update`$(\bar{\mathcal{L}}, \boldsymbol{\theta}, \mathcal{J}, \gamma)$;
    $\bar{\mathcal{L}}, \boldsymbol{\theta} \leftarrow$ `Regularize`$(\bar{\mathcal{L}}, \boldsymbol{\theta}, \alpha, \beta, w_{\max})$;
return $\bar{\mathcal{L}}^* \leftarrow \bar{\mathcal{L}}$, $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$

Given the state space as $S = \{(\mathcal{E}_t, \tau_t)\}_{t=0}^{T}$, the acceptable region is defined as $\{S \mid \bar{\mathcal{L}}(S;\boldsymbol{\theta}) > 0\}$. The shortcut issue arises when the acceptable region encompasses states that should be excluded. Thus, our objective is to constrain the acceptable region through regularization.

1) Regularization of Predicate Parameters $\boldsymbol{\theta}$: We impose constraints on the predicate parameters $\boldsymbol{\theta}$ to induce movement in a direction that reduces the acceptable region. The following regularization is applied: $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \cdot \text{sign}\left(\frac{\partial \bar{\mathcal{L}}}{\partial \boldsymbol{\theta}}\right)$, where α is a small positive constant controlling the update magnitude. The sign function ensures that $\boldsymbol{\theta}$ moves in the direction that decreases the overall logic value, irrespective of gradient magnitude. This regularization serves to reduce the number of states included in the acceptable region.

2) Regularization of Logic Structure $\bar{\mathcal{L}}$: In the Aggregation layer, we promote the selection of the $\wedge$ operator while preventing excessive bias. For a weight vector $W_s = [w^\wedge_{op}, w^\vee_{op}]$, we apply: $w^\wedge_{op} = \min(w^\wedge_{op}+\beta, w_{\max})$, where β is a small positive constant and $w_{\max}$ is a predefined maximum weight value. This regularization contributes to reducing the number of states included in the acceptable region, as the $\wedge$ operator is more restrictive than $\vee$. The capping mechanism ensures that $w_\wedge$ does not grow infinitely.

The key argument here is that if a particular logic structure or parameter is truly significant for capturing the essential rules of the demonstration data, it will not be eliminated by the regularization process. This self-correcting mechanism ensures that the regularization primarily affects spurious or overly permissive rules while preserving the core driving principles embedded in the demonstrations.

V. Experiments

A. General Experiment Setup

1) Predicate Types: We consider 63 condition predicates and 20 action predicates in the predicate set $\mathcal{P}$. All predicates are categorized into four types: (1) safety-related (e.g., if the plan will collide with obstacles) (2) comfort-related (e.g., if the acceleration is in a reasonable range); (3) efficiency-related (e.g., if the car is in a slow lane); (4) environment-related (e.g., if the curvature of the current lane is high). Implementing the predicate set $\mathcal{P}$ involves LLM-aided design [27], but the final predicates [3] were manually checked to ensure correctness and sensibility.

2) Dataset and Scenarios: We used all the NuPlan demonstrations in Singapore, Pittsburgh, and Boston, and part of the demonstrations in Las Vegas, to train the logic structure. The scenarios are grouped into 9 types (the first column in Table II). The dataset is split into 90% training and 10% validation. The validation set is used for hyperparameter tuning and early stopping.

3) Hyperparameters and Training Details: We trained an ensemble $\mathcal{L}_{ensemble}$ with 10 different $\mathcal{L}$. Each $\mathcal{L}$ has 2 layers of the Temporal layer. The single class regularization parameters α and β are set to 10^{-5} and 10^{-3}, respectively. The learning rate is set to 10^{-4}, and optimized with the Adam optimizer. The batch size is set to 32. The training process is stopped when the validation loss does not decrease for 10 epochs.

B. Case Study

1) Logic Rules Discovered: Fig. 1 shows three of the rules learned by the grading logic network. Given the motion plan proposed by PDM [1], the grading logic network assigns a score to each plan based on the learned rules. Fig. 3 shows why the blue, red and orange plans receive lower scores. The blue plan receives a low score for going beyond the drivable area (the gray area), which violates the rule $\top \rightarrow$ **G***InDrivable*. The red plan is penalized for exceeding comfort constraints (the blue dashed line) on lateral acceleration, which violates the rule **G***SafeTTC* $\rightarrow$ **G***Comfortable*. The orange plan receives a lower score for exceeding the speed limit (the red dashed line) when not overtaking another vehicle, which violates the rule $\neg$*SpeedLimit* $\rightarrow$ **F***Overtaking*. These rules are almost always discovered in our experiment. One exception is that the rule $\neg$*SpeedLimit* $\rightarrow$ **F***Overtaking* could sometimes devolve to *SpeedLimit* or **G***SpeedLimit* in scenarios where overtaking happens rarely (e.g., following other cars).

We also discovered several interesting non-trivial rules. For instance, *CutInBehind* $\wedge$ *Cruise* $\rightarrow$ **F***Accelerate*. This rule indicates that when a vehicle cuts in behind the ego vehicle

[3]The predicate code will be published online.

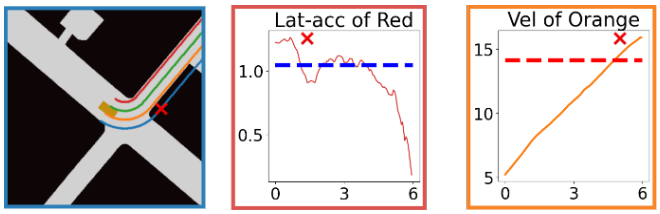

Fig. 3. Case study on discovered rules. From left to right, these sub-figures explained why the blue, red, and orange plans received lower scores.

(*CutInBehind*) and the ego car runs at a constant speed (*Cruise*), it will eventually lead to acceleration (**F***Accelerate*). This makes sense from a driving perspective because when another vehicle cuts from behind, it often makes the driver stressed and leads to acceleration.

2) Parameter Optimization: $Comfortable_{\boldsymbol{\theta}}$ is a predicate measure if a motion plan is comfortable, which has parameters $\boldsymbol{\theta} = [\theta_{a_{lat}}, \theta_{a_{lon}}, \theta_{j_{lat}}, \theta_{j_{lon}}]$, representing thresholds for acceleration in forward, backward, left, and right directions. It can be defined as:

$$Comfortable_{\boldsymbol{\theta}} := \tanh(\min_{i \in \{a_f, a_b, a_l, a_r\}} \{\theta_i - i(\tau)\}) \qquad (12)$$

where $i(\tau)$ represents the maximum acceleration in the plan τ for each respective direction. Only when all the accelerations are below their thresholds, is the predicate evaluated as positive. The tanh function is used to ensure the output is in $[-1, 1]$. All the threshold parameters in (12) are differentiable and can be learned from the data. We learned the parameters $\boldsymbol{\theta}$ from all the training demonstration data. The learned parameters are shown in Table I.

TABLE I

CASE STUDY ON PARAMETERS OF $Comfortable_{\boldsymbol{\theta}}$

	a_f	a_b	a_l	a_r
Standard	1.23	1.13	0.98	0.98
Learned	1.1 ± 0.21	1.045 ± 0.12	0.9 ± 0.11	0.95 ± 0.51

The learned parameters are shown in the format of mean ± standard deviation. It is computed from 5 independent runs.

The **Standard** of comfortable acceleration is provided in [28], we choose the thresholds when the participants feel middling. The parameter **Learned** is close to the **Standard**. In practice, it is noticeable that the acceleration thresholds are hard to estimate. However, the learned parameters are close to the standard by learning from data, which indicates the proposed method can learn the parameters characterizing the demonstration data well.

C. Evaluation in Closed-loop Planning

A practical way to evaluate the learned rules is to deploy them in a closed-loop planning system. A good logic structure should be able to filter out undesirable plans and only allow the well-performing plans to be executed. We evaluate the learned logic structure in the closed-loop planning system on the NuPlan, for both Closed-Loop Non-Reactive (CL-NR) and Closed-Loop Score Reactive (CL-R) settings. In the CL-NR setting, all the other agent except the ego vehicle is log-replayed on the original data, while in the CL-R setting, the different agents are controlled by an IDM policy instead. The results are shown in Table II, which reflects an overall performance in terms of the metrics defined by NuPlan. Notice that these metrics are not the same as the scores given by our learned rules, because they require knowing the full simulation log to compute, while our learned rules can only use history and current information. Here, we compared our learned Scoring Logic Network (SLN) with the Expert Rules (ER) in PDM [1] and the neural critic [4]. The last two rows show the overall performance on "All" the scenarios and the Val14 split originally used for evaluating the ER [1].

TABLE II

CLOSED LOOP PLANNING PERFORMANCE

	Rule Complexity			CLS-NR ↑			CLS-R ↑		
	$\lvert\bar{\mathcal{P}}\rvert$	$\lvert\dot{\mathcal{P}}\rvert$	#. Rules	ER	NC	SLN	ER	NC	SLN
Change Lane	20		24	0.89	0.79	**0.92**	0.88	0.77	**0.91**
Following	10		16	**0.94**	0.88	0.91	**0.96**	0.90	**0.96**
Near Static	10		19	0.87	0.77	**0.93**	0.87	0.85	**0.87**
Near VRU	10		17	0.87	0.81	**0.93**	**0.89**	0.75	0.87
Turn	20	20	31	0.89	0.82	**0.91**	0.89	0.71	**0.91**
Stopping	10		13	**0.90**	0.77	**0.90**	0.92	0.85	**0.93**
Starting	20		27	**0.91**	0.85	0.89	0.88	0.84	**0.90**
Stationary	20		21	**0.94**	0.73	**0.94**	0.96	0.81	**0.97**
Traversing	20		29	0.87	0.71	**0.90**	0.89	0.70	**0.90**
All	63	20	124	0.90	0.79	**0.92**	0.91	0.81	**0.93**
Val14				0.93	0.81	**0.94**	0.92	0.83	**0.93**

The videos and explanation of all the scenarios are available online.
The action predicate set $\bar{\mathcal{P}}$ is identical across all the scenarios.
The condition predicate set $\dot{\mathcal{P}}$ in different scenarios shares some common predicates such as *SafeTTC*, but are not identical (e.g., the near Venerable Road Unit (VRU) scenario has a unique predicate like *VRUInCrosswalk*).

The complexity of rules is measured by the number of action predicates $|\bar{\mathcal{P}}|$, the number of condition predicates $|\dot{\mathcal{P}}|$, and the total number of extracted action-condition pairs. The detailed results are shown in Table II. In the All and Val14 splits, we used all of our 63 condition predicates and 20 action predicates. In the CL-NR setting, SLN outperforms ER in 6 out of 9 scenario types, ties in 2, and underperforms in 1, while in CL-R, it surpasses ER in 5 scenarios, ties in 2, and falls short in 2. Notably, SLN achieves this performance without the complex relationship modeling or extensive parameter tuning required by ER. Compared to the NC, SLN exhibits superior performance across all 9 scenario types in both settings, offering substantial improvement coupled with interpretability. Overall (in the last two rows), SLN consistently outperforms both ER and NC for all the scenarios and the Val14 [1] split is used to evaluate ER, in both reactive and non-reactive settings.

D. Proposal Approach

A robust rule should be above all able to filter out undesirable plans for different proposal approaches. We evaluate the learned rules on different proposal approaches, including PDM [1], AT-Sampler [4], Hybrid-PDM [1], and ML-Prop [29]. The results are shown in Table III.

For both rule-based [1], [4] and learning-based proposal approaches [1], [29], our learned rules demonstrate superior performance in the closed-loop planning system, as shown by the data presented in Table III.

TABLE III

PROPOSAL APPROACH ABLATION ON "ALL"

	CLS-NR ↑			CLS-R ↑		
	ER	NC	SLN (ours)	ER	NC	SLN (ours)
PDM [1]	0.90	0.79	**0.92**	0.91	0.81	**0.93**
AT-Sampler [4]	0.83	0.81	**0.89**	0.84	0.80	**0.90**
Hybrid-PDM [1]	0.90	0.79	**0.92**	0.91	0.79	**0.92**
ML-Prop* [29]	0.81	0.75	**0.86**	0.82	0.76	**0.85**

* [29] introduced an deterministic policy. We generate one initial proposal with this policy and then add lateral deviations to generate multiple parallel proposals.

VI. CONCLUSION

This paper introduced FLoRA, a framework for learning interpretable scoring rules expressed in temporal logic for autonomous driving planning systems. FLoRA addresses key challenges by developing a learnable logic structure to capture nuanced relationships among driving predicates; proposing a data-driven method to optimize rule structure and parameters from demonstrations; and presenting an optimization framework for learning from single-class driving demonstration data. Experimental results on the NuPlan dataset show that FLoRA outperforms both expert-crafted rules and neural network approaches across various scenarios, with different proposer types. This work represents a significant step towards creating more adaptable, interpretable, and effective scoring mechanisms for autonomous driving systems, bridging the gap between data-driven approaches and rule-based systems. The most significant limitation of FLoRA is that it can only discover the relationships between the provided predicates with differntiable parameters, but cannot discover the predicates themselves. Future work will focus on extending FLoRA with LLM-aided predicate discovery.

REFERENCES

[1] D. Dauner, M. Hallgarten, A. Geiger, and K. Chitta, "Parting with misconceptions about learning-based vehicle motion planning," in *Conference on Robot Learning (CoRL)*, 2023.

[2] Y. Hu, J. Yang, L. Chen, K. Li, C. Sima, X. Zhu, S. Chai, S. Du, T. Lin, W. Wang *et al.*, "Planning-oriented autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 17 853–17 862.

[3] T. Phan-Minh, F. Howington, T.-S. Chu, S. U. Lee, M. S. Tomov, N. Li, C. Dicle, S. Findler, F. Suarez-Ruiz, R. Beaudoin *et al.*, "Driving in real life with inverse reinforcement learning," *arXiv preprint arXiv:2206.03004*, 2022.

[4] S. Jiang, Z. Xiong, W. Lin, Y. Cao, Z. Xia, J. Miao, and Q. Luo, "An efficient framework for reliable and personalized motion planner in autonomous driving," *Preprint*, 2022.

[5] N. Karnchanachari, D. Geromichalos, K. S. Tan, N. Li, C. Eriksen, S. Yaghoubi, N. Mehdipour, G. Bernasconi, W. K. Fong, Y. Guo, and H. Caesar, "Towards learning-based planning: The nuplan benchmark for real-world autonomous driving," *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 629–636, 2024.

[6] W. Xiao, N. Mehdipour, A. Collin, A. Y. Bin-Nun, E. Frazzoli, R. D. Tebbens, and C. Belta, "Rule-based optimal control for autonomous driving," in *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, 2021, pp. 143–154.

[7] J. Wishart, S. Como, M. Elli, B. Russo, J. Weast, N. Altekar, E. James, and Y. Chen, "Driving safety performance assessment metric for ads-equipped vehicles," *SAE International Journal of Connected and Automated Vehicles*, vol. 3, no. 2020-01-1206, pp. 23–35, 2020.

[8] P. Junietz, F. Bonakdar, B. Klamann, and H. Winner, "Criticality metric for the safety validation of automated driving using model predictive trajectory optimization," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 60–65.

[9] M. Hekmatnejad, S. Yaghoubi, A. Dokhanchi, H. B. Amor, A. Shrivastava, L. Karam, and G. Fainekos, "Encoding and monitoring responsibility sensitive safety rules for automated vehicles in signal temporal logic," in *International Conference on Formal Methods and Models for System Design*. ACM, 2019.

[10] Y. Deng, Y. Jiang, X. Xia, L. Zhu, and J. Gao, "Target: Automated scenario generation from traffic rules for testing autonomous vehicles," *arXiv preprint arXiv:2305.06018*, 2023.

[11] Y. Zhou, Y. Sun, Y. Tang, Y. Chen, J. Sun, C. M. Poskitt, Y. Liu, and Z. Yang, "Specification-based autonomous driving system testing," *IEEE Transactions on Software Engineering*, 2023.

[12] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, L. Chen, A. Jain, S. Omari, V. Iglovikov, and P. Ondruska, "One thousand and one hours: Self-driving motion prediction dataset," in *Conference on Robot Learning*, 2021.

[13] S. Riedmaier, T. Ponn, D. Ludwig, B. Schick, and F. Diermeyer, "A survey on scenario-based safety assessment of automated vehicles," *IEEE Access*, vol. 8, pp. 87 456–87 477, 2020.

[14] B. Weng, S. J. Rao, E. Deosthale, S. Schnelle, and F. Barickman, "Model predictive instantaneous safety metric for evaluation of automated driving systems," *IEEE Access*, 2020.

[15] A. B. Vasudevan, N. Peri, J. Schneider, and D. Ramanan, "Planning with adaptive world models for autonomous driving," 2024.

[16] Z. Huang, J. Wu, and C. Lv, "Driving behavior modeling using naturalistic human driving data with inverse reinforcement learning," *IEEE transactions on intelligent transportation systems*, vol. 23, no. 8, pp. 10 239–10 251, 2021.

[17] M. Kuderer, S. Gulati, and W. Burgard, "Learning driving styles for autonomous vehicles from demonstration," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[18] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner, "Large-scale cost function learning for path planning using deep inverse reinforcement learning," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1073–1087, 2017.

[19] D. Silver, J. A. Bagnell, and A. Stentz, "Learning autonomous driving styles and maneuvers from expert demonstration," in *Experimental Robotics*. Springer, 2013, pp. 371–386.

[20] E. Bartocci, C. Mateis, E. Nesterini, and D. Nickovic, "Survey on mining signal temporal logic specifications," *Information and Computation*, vol. 289, p. 104957, 2022.

[21] K. Leung, N. Aréchiga, and M. Pavone, "Backpropagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods," *The International Journal of Robotics Research*, vol. 42, pp. 356 – 370, 2020.

[22] S. Jha, A. Tiwari, S. A. Seshia, T. Sahai, and N. Shankar, "Telex: learning signal temporal logic from positive examples using tightness metric," *Formal Methods in System Design*, 2019.

[23] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, ser. IJCAI '13. AAAI Press, 2013, p. 854–860.

[24] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theoretical Computer Sci.*, 2009.

[25] J. V. Deshmukh, A. Donz'e, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, "Robust online monitoring of signal temporal logic," *Formal Methods in System Design*, vol. 51, no. 1, pp. 5–30, 2017.

[26] E. J. McCluskey, "Minimization of boolean functions," *Bell System Technical Journal*, vol. 35, pp. 1417–1444, 1956.

[27] B. Chen, F. Zhang, A. Nguyen, D. Zan, Z. Lin, J.-G. Lou, and W. Chen, "Codet: Code generation with generated tests," *ArXiv*, vol. abs/2207.10397, 2022.

[28] K. N. de Winkel, T. Irmak, R. Happee, and B. Shyrokau, "Standards for passenger comfort in automated vehicles: Acceleration and jerk," *Applied Ergonomics*, vol. 106, p. 103881, 2023.

[29] O. Scheel, L. Bergamini, M. Wolczyk, B. Osiński, and P. Ondruska, "Urban driver: Learning to drive from real-world demonstrations using policy gradients," in *Conference on Robot Learning*, 2022.