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

Zikang Xiong, Joe Eappen, and Suresh Jagannathan

Abstract—In autonomous driving systems, motion planning is commonly implemented as a two-stage process: first, a trajectory proposer generates multiple candidate trajectories, then a scoring mechanism selects the most suitable trajectory for execution. For this critical selection stage, rule-based scoring mechanisms are particularly appealing as they can explicitly encode driving preferences, safety constraints, and traffic regulations in a formalized, human-understandable format. However, manually crafting these scoring rules presents significant challenges: the rules often contain complex interdependencies, require careful parameter tuning, and may not fully capture the nuances present in real-world driving data. This work introduces FLoRA, a novel framework that bridges this gap by learning interpretable scoring rules represented in temporal logic. Our method features a learnable logic structure that captures nuanced relationships across diverse driving scenarios, optimizing both rules and parameters directly from real-world driving demonstrations collected in NuPlan. Our approach effectively learns to evaluate driving behavior even though the training data only contains positive examples (successful driving demonstrations). Evaluations in closed-loop planning simulations demonstrate that our learned scoring rules outperform existing techniques, including expert designed rules and neural network scoring models, while maintaining interpretability. This work introduces a data-driven approach to enhance the scoring mechanism in autonomous driving systems, designed as a plug-in module to seamlessly integrate with various trajectory proposers. Our video and code are available on xiong.zikang.me/FLoRA/.

I. INTRODUCTION

Modern autonomous driving systems typically produce multiple potential plans [2]–[6], as this parallel approach offers several key advantages: it allows the system to consider different driving modalities (such as aggressive or conservative behaviors), accounts for future uncertainties, and provides redundancy in case certain paths become infeasible. These generated plans then need to be evaluated through a scoring mechanism to select the most suitable one for execution.

The importance of effective scoring becomes even more apparent in complex autonomous driving systems, particularly in end-to-end approaches [3], [6]. These systems often utilize large, intricate neural networks that incorporate elements of randomness, such as dropout or sampling from probability distributions. While powerful, such characteristics can make it challenging to predict the system's behavior consistently [6]. By implementing interpretable scoring rules as a final evaluation layer, we can assess these generated plans against clear, understandable criteria, thereby introducing much-needed predictability and reliability to these complex systems. In essence, scoring techniques serve as a critical bridge, connecting the

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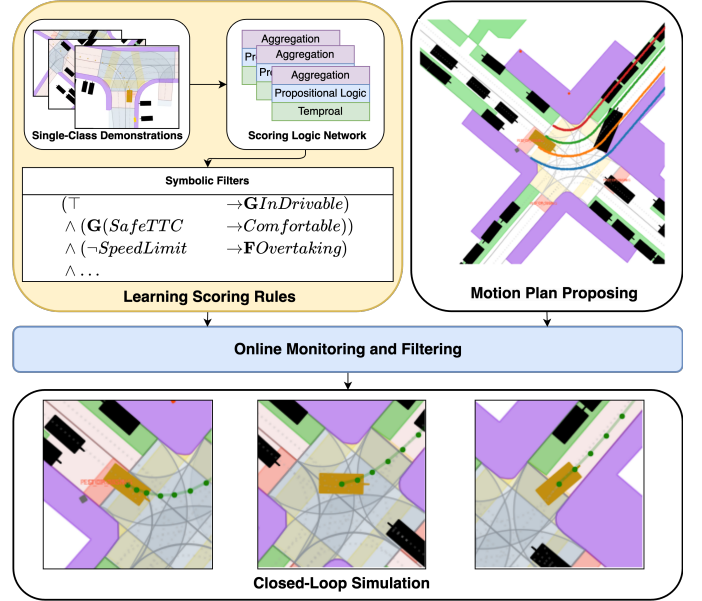

Fig. 1. This figure illustrates our framework for scoring and selecting trajectories in autonomous driving systems. Modern autonomous driving planners typically follow a propose-selection paradigm, where multiple candidate trajectories are first generated and then filtered through a scoring mechanism. As shown in the **Motion Plan Proposing** block, multiple trajectories (colored lines) are proposed as potential future paths for the autonomous vehicle. These candidates need to be evaluated and ranked to select the most suitable trajectory for execution. Our key contribution, the **Learning Scoring Rules** block, demonstrates how we learn interpretable scoring rules from human driving demonstration from NuPlan [1]. Instead of manually crafting rules or using black-box scoring models, we develop a Scoring Logic Network (SLN) that automatically learns temporal logic rules from data (specific rules illustrated in Sec. V-B1). These learned rules are then deployed in the **Online Monitoring and Filtering** block, which continuously evaluates the proposed trajectories at 20 Hz following NuPlan's log frequency [1] during operation. In the scene visualization, the ego vehicle is shown in gold, while other vehicles are represented in black, the curbs are marked in purple, and the driveable areas (lanes and intersections) are marked as pink and blue, respectively. Among the proposed trajectories, the green trajectory receives the highest score as it successfully maintains a safe distance from the curbs, while exhibiting appropriate curvature and comfort characteristics. All other colored trajectories are not selected for violating one or more learned rules, as detailed in Sec. V-B1. During the monitoring, following the setting in Planning Driver Model (PDM) [2], we assume that the future 4-second trajectories of other cars are known. The selected trajectory is then executed in the **Closed-Loop Simulation** block.

raw outputs of planning algorithms to the final, executable plans. This additional evaluation step helps mitigate the uncertainties inherent in complex planning systems, significantly enhancing the safety, efficiency, and overall performance of autonomous vehicles as they navigate through our highways and cities.

With real-world driving data available in datasets like Nu-

Plan [1], most current learning methods focus on directly learning motion plan proposers rather than learning interpretable scoring mechanisms to evaluate these plans. We instead 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* [1], [6], [7]. Having only single-class (i.e., correct-behavior only) training data poses a significant challenge for learning scoring models, 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 [1] 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 [2], PDM-Hybrid [2], ML-Prop [1], and a rule-based Acceleration-Time (AT) sampler [5]. In summary, our contributions are as follows:

- We propose a novel learnable logic structure that discovers and captures latent relationships 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 rules and parameters from human driving demonstrations, without requiring unsafe and rare accident examples.
- We demonstrate the effectiveness of our approach in scoring and selecting desirable plans via NuPlan closed-loop simulation, outperforming expert-crafted rules [2] and neural network-based approaches [5] across various scenarios and proposers.

II. RELATED WORK

Scoring models are widely used in autonomous driving systems to evaluate the safety, efficiency, and comfort of planned trajectories [1]–[5], [8]–[21]. This evaluation becomes particularly crucial in end-to-end autonomous driving systems, where behavioral uncertainties arise from multiple sources: neural network stochasticity (e.g., dropout, distributional sampling) [3], environmental condition variations [3], and complex multi-agent interaction modeling [6].

Existing scoring models can be broadly categorized into rule-based and learning-based models. Rule-based scoring models [1]–[3], [8]–[15] 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, [4], [5], [16]–[21] are data-driven and can capture complex patterns, but often sacrifice interpretability, which can be intimidating for safety-critical applications.

In the broader context of temporal logic rule learning [22], some researchers have explored learning interpretable rules from demonstrations. However, the current literature has not explored the potential of learning interpretable rules directly from driving data, in the context of autonomous driving systems.

III. PRELIMINARIES

In this section, we formulate the key technical components and our objectives.

a) Predicate P_θ: At a certain time point, given all environment information $\mathcal{E} = (M, I, A)$ and a motion plan τ, the differentiable predicate P_θ is defined as: $P_\theta : (\mathcal{E} \times \tau) \to [-1, 1]$, which evaluates driving conditions and the ego car's motion plan [1]. Here, M represents the HD map API that provides the functionality to query a drivable area, lane information, routing information and occupancy grid; $I = \{i_1, ..., i_K\}$ represents the set of traffic light states, where

---

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

each $i_k \in \{red, yellow, green\}$; $A = \{a_1, ..., a_N\}$ represents the set of other agents, where each $a_i = \{(x_t, y_t, v_t)\}_{t=0}^T$ contains the agent's trajectory for the next 4 seconds (i.e., $T = 80$, $20Hz \times 4s$). The motion plan $\tau = \{(x_t, y_t, v_t)\}_{t=0}^T$ consisting of a sequence of positions (x_t, y_t) and speeds v_t, where higher-order derivatives (e.g., acceleration, jerk) can be computed through numerical differentiation when needed. P_θ maps $\mathcal{E}$ and τ to a truth confidence value in $[-1, 1]$, where θ are the predicate parameters. Each predicate P_θ is parameterized by θ, which defines thresholds or constraints specific to that predicate (e.g., safe time-to-collision threshold, comfortable acceleration bounds). 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. Similar to most existing work [22], 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 $\mathcal{P} = \{P_{\theta_1}^1, P_{\theta_2}^2, \ldots, P_{\theta_n}^n\}$, we introduce a LTL_f logic space [23] (LTL over finite traces) that includes compositions of predicates from $\mathcal{P}$ and logic operators. The logic formula $\mathcal{L}$ can be generated from the following grammar:

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

where $P_\theta \in \mathcal{P}$ is a differentiable predicate, $\mathbf{G}$ and $\mathbf{F}$ 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 [22], 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 $\mathcal{L}(S; \theta) \rightarrow [-1, 1]$. Here, θ 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 $\mathbf{G}$ and $\mathbf{F}$ 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. The evaluation function ρ is defined as:

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

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

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

All the operations defined by ρ are differentiable, which enables the use of backpropagation in the learning process. In practice, we use softmin and softmax to approximate min and max operators for a smooth gradient [26]. 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 θ of the predicates, which characterize the demonstration data accurately. Formally, we aim to solve the following problem:

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

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 introduces our novel regularization techniques to overcome the limitations of learning from 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 $\bar{\mathcal{P}} = \{\bar{P}_{\theta_1}^1, \bar{P}_{\theta_2}^2, \ldots, \bar{P}_{\theta_n}^{\bar{n}}\}$ and action predicates $\dot{\mathcal{P}} = \{\dot{P}_{\theta_1}^1, \dot{P}_{\theta_2}^2, \ldots, \dot{P}_{\theta_m}^{\dot{m}}\}$. 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

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

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

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

where $\odot$ denotes that this condition-action pairs are connected by $\wedge$ or $\vee$ operators. The conjunction of conditions ($\wedge_{j=1}^n condition_j$) 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 bottom 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 $\mathcal{P} = \{P_{\theta_1}^1, P_{\theta_2}^2, ..., P_{\theta_N}^N\}$ as a set of N predicates. For each predicate $P_{\theta_i}^i \in \mathcal{P}$ and each frame $f_t \in F$, we compute $X_i^t = P_{\theta_i}^i(f_t)$. Let $X_i^T = [x_i^1, x_i^2, ..., x_i^T]$ be the sequence of predicate values for predicate $P_{\theta_i}^i$ across all time steps. The output of the predicates is then defined as $\mathcal{X}^T = \{X_1^T, X_2^T, ..., X_N^T\}$, where each $X_i^T \in \mathbb{R}^T$ 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 $X_i^T \in \mathcal{X}^T$, potentially applying a temporal operator. Formally, the output of $\mathcal{T}$ is $O^T = \mathcal{T}(\mathcal{X}^T) = \{o_1^T, o_2^T, ..., o_N^T\}$, where $o_i^T = \mathcal{T}(x_i^T) \in \{\mathbf{G}X_i^T, \mathbf{F}X_i^T, X_i^T\}$. Temporal layers can be stacked, allowing for the composition of temporal operators. For instance, with two stacked temporal layers, we could have $\mathcal{T}_2(\mathcal{T}_1(X_i^T)) = \mathbf{F}(\mathbf{G}(X_i^T))$. 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 $\mathcal{F}(O^T) = O^P = \{o_1^P, o_2^P, ..., o_{\binom{N}{2}}^P\}$, where $o_i^P = (\neg)o_j^T \circ (\neg)o_k^T$, $\circ \in \{\wedge, \vee\}$, and $O^T = \{o_1^T, o_2^T, ..., o_N^T\}$ 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; aggregating on one **Propositional** layer can represent any formula in the form of (6) as we show in IV-B3.

The **Aggregation** layer aggregates the output of the **Propositional** (O^P) layer into one cluster using logical connectives $\{\wedge, \vee\}$ to connect neighboring clusters. Formally, given the input O^P, the output of the **Aggregation** layer can be represented as $\mathcal{A}(O^P) = o_1^P \circ_1 o_2^P \circ_2 ... \circ_{\binom{N}{2}-1} o_{\binom{N}{2}}^P$, where $\circ$ represents $\wedge$ or $\vee$.
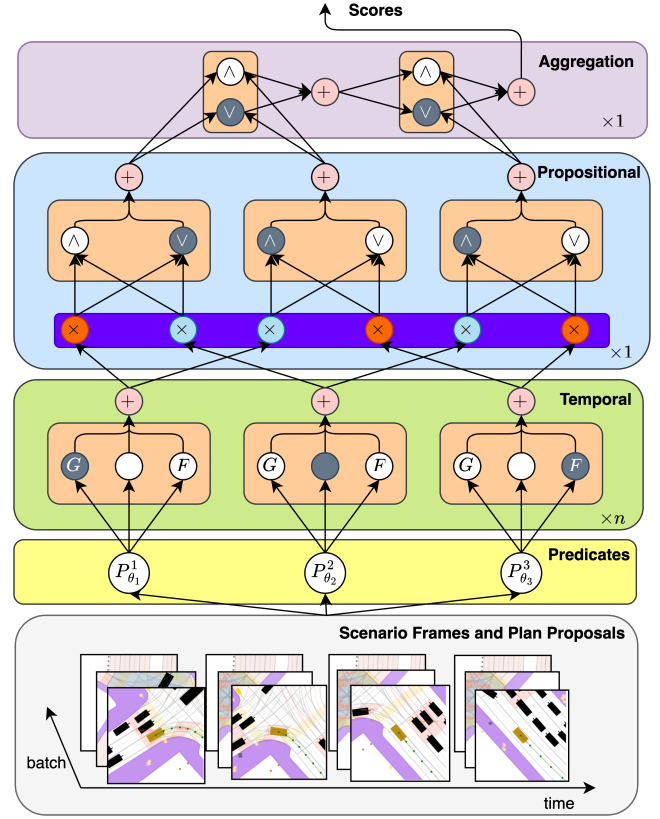

Fig. 2. The logic structure $\bar{\mathcal{L}}$ consists of three types of layers: **Temporal, Propositional**, and **Aggregation**. The **Temporal** layer processes the initial predicates, applying temporal operators. The **Propositional** layer generates all possible pairs of predicates connected by logical operators. The **Aggregation** layer aggregates the output of the **Propositional** layer into one cluster by deciding the logic operator to connect neighboring clusters. **Temporal** layers can be stacked. a layer's formal definition is in Sec. IV-B1. 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 ($\bigcirc$) in these gates represents a single value *weight*. In the selection gate, the $\bullet$ circle represents the operator with the largest weight, meaning the operator is selected. In the negation gate, the $\bigcirc$ circle represents the negation of the input (i.e., multiply with a negative number), while the $\bullet$ circle represents the original input (i.e., a positive number). The gate implementation is described in Sec. IV-B2. Supposing we only consider one layer of Temporal layer ($n = 1$), and given a set of predicates $\mathcal{P} = \{P_{\theta_1}^1, P_{\theta_2}^2, P_{\theta_3}^3\}$, $P_{\theta_2}^2 \in \bar{\mathcal{P}}$ and $P_{\theta_1}^1, P_{\theta_3}^3 \in \dot{\mathcal{P}}$, this learnable logic structure represents the logic formula $(\mathbf{G}P_{\theta_1}^1 \vee \neg P_{\theta_2}^2) \vee (\neg \mathbf{G}P_{\theta_1}^1 \wedge \mathbf{F}P_{\theta_3}^3) \vee (\neg P_{\theta_2}^2 \wedge \mathbf{F}P_{\theta_3}^3)$ This formula can be further reduced to $P_{\theta_2}^2 \rightarrow (\mathbf{G}P_{\theta_1}^1 \vee \mathbf{F}P_{\theta_3}^3)$.

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 $\bigcirc$ 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:

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

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 blends operators using softmax to enable continuous gradient flow during training.

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.

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

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 be 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. $\square$

This theorem proves that a single **Propositional** layer is sufficient to express all condition-action pairs, justifying our computationally efficient design choice.

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,

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

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:

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

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:

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

A concrete example is shown in Fig. 2 by iteratively applying (10) and (11) to the selection and negation gates. For an ensemble 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 [27] 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).

C. Learning From 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 θ^* that grade demonstrations with the highest score and unseen behaviors with lower scores. Directly optimizing on (4) would simply make the score $\bar{\mathcal{L}}(S; \theta)$ to be 1 for all demonstrations. The optimizer could find "shortcuts" and result in learning trivial formulas like $P_{\theta_1}^1 \vee \neg P_{\theta_1}^1$, which are always true and do not provide meaningful rules. To address this issue, we introduce two regularization terms for θ and $\bar{\mathcal{L}}$. The full learning algorithm is presented in Algorithm 1.

The learning system evaluates driving behaviors in a state space $S = \{(\mathcal{E}_t, \tau_t)\}_{t=0}^{T}$, where behaviors with $\bar{\mathcal{L}}(S; \theta) > 0$ are considered acceptable. Without constraints, the system might learn shortcuts that accept almost any behavior. We prevent this through two types of regularization:

1) Preventing Shortcut Learning Through Regularization: The learning system evaluates driving behaviors in a state space $S = \{(\mathcal{E}_t, \tau_t)\}_{t=0}^{T}$, where behaviors with $\bar{\mathcal{L}}(S; \theta) > 0$ are considered acceptable. Without constraints, the system might learn shortcuts that accept almost any behavior. Consider a simple example where we want to learn rules for safe lane changes. Without regularization, the system might learn the rule: InLane $\vee$ SafeDistance that accepts behaviors in which the car is either in a lane OR maintains safe distance, which is clearly too permissive. We prevent such shortcuts through two types of regularization: First, we gradually tighten the parameters θ of each rule using: $\theta = \theta - \alpha \cdot \text{sign}\left(\frac{\partial \bar{\mathcal{L}}}{\partial \theta}\right)$. In our example, this might gradually increase the required safe distance threshold from 1 meter to a more reasonable 2 meters. Second, when combining rules, we encourage the use of AND ($\wedge$) over OR ($\vee$) operations using: $w_{op}^{\wedge} = \min(w_{op}^{\wedge} + \beta, w_{\max})$

---

[2] $\bar{\mathcal{L}}$ means the learnable logic structure, while $\mathcal{L}$ means a concrete logic formula defined by (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 $\boldsymbol{\theta}^*$

1 **Function** Update($\bar{\mathcal{L}}, \boldsymbol{\theta}, \mathcal{J}, \gamma$):
2   Update $\boldsymbol{\theta}$ with $\nabla_{\boldsymbol{\theta}} \mathcal{J}$;
3   Update $\bar{\mathcal{L}}$ (i.e., gate weights) with $\nabla_{\bar{\mathcal{L}}} \mathcal{J}$;
4   **return** $\bar{\mathcal{L}}, \boldsymbol{\theta}$;

5 **Function** Regularize($\bar{\mathcal{L}}, \boldsymbol{\theta}, \alpha, \beta, w_{\max}$):
6      $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \text{sign}(\frac{\partial \bar{\mathcal{L}}}{\partial \boldsymbol{\theta}})$;
7   **foreach** *Aggregation layer in* $\bar{\mathcal{L}}$ **do**
8          $w_\wedge \leftarrow \min(w_\wedge + \beta, w_{\max})$;
9   **return** $\bar{\mathcal{L}}, \boldsymbol{\theta}$;

10 **for** *epoch* $\leftarrow 1$ **to** E **do**
11   **for** *each batch* $\{S_1, \ldots, S_B\} \sim \mathcal{D}^+$ **do**
12          $\mathcal{J} \leftarrow \frac{1}{B} \sum_{i=1}^{B} \bar{\mathcal{L}}(S_i; \boldsymbol{\theta})$;
13          $\bar{\mathcal{L}}, \boldsymbol{\theta} \leftarrow$ Update($\bar{\mathcal{L}}, \boldsymbol{\theta}, \mathcal{J}, \gamma$);
14          $\bar{\mathcal{L}}, \boldsymbol{\theta} \leftarrow$ Regularize($\bar{\mathcal{L}}, \boldsymbol{\theta}, \alpha, \beta, w_{\max}$);

15 **return** $\bar{\mathcal{L}}^* \leftarrow \bar{\mathcal{L}}, \boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$

This helps the system learn more appropriate rules like: InLane $\wedge$ SafeDistance requiring both conditions to be met for a safe lane change.

Importantly, these constraints only eliminate spurious shortcuts - any rule structure or parameter that genuinely captures important driving behavior will remain intact, as it will be consistently reinforced by the demonstration data despite the regularization pressure.

V. EXPERIMENTS

A. General Experiment Setup

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., will the plan result in obstacle collision?) (2) comfort-related (e.g., is the acceleration in a reasonable range?); (3) efficiency-related (e.g., is the car in a slow lane?); (4) environment-related (e.g., is the curvature of the current lane too high?). Implementing the predicate set $\mathcal{P}$ involves LLM-aided design [28], but the final predicates were manually checked to ensure correctness and sensibility.

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. Following PDM [2], we use the log replay to obtain the 4-second future trajectories of other agents in both training and evaluation.

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. We used 15 trajectory candidates for evaluation in closed-loop simulation.

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 [2], 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 $\top \rightarrow \mathbf{G}$*InDrivable*. The red plan is penalized for exceeding comfort constraints (the blue dashed line) on lateral acceleration, which violates the rule $\mathbf{G}$*SafeTTC* $\rightarrow \mathbf{G}$*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* $\rightarrow \mathbf{F}$*Overtaking*. These rules are almost always discovered in our experiment. One exception is that the rule $\neg$*SpeedLimit* $\rightarrow \mathbf{F}$*Overtaking* could sometimes devolve to *SpeedLimit* or $\mathbf{G}$*SpeedLimit* in scenarios where overtaking happens rarely (e.g., following other cars). More detailed case studies are provided on our website.
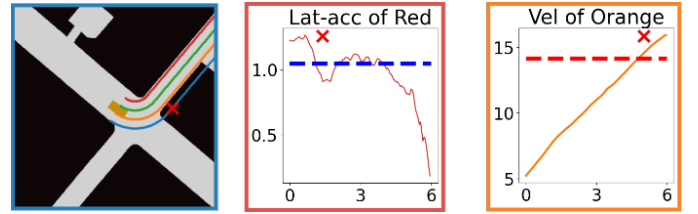

Fig. 3. Case study on discovered rules. While our method evaluates 15 trajectory candidates in practice, we show only 4 representative trajectories here for visual clarity. From left to right, these sub-figures explain why the blue, red, and orange plans received lower scores. The blue plan violates the drivable area rule, the red plan exceeds comfort constraints on lateral acceleration, and the orange plan breaks the speed limit without overtaking context.

2) Parameter Optimization: *Comfortable*$_{\boldsymbol{\theta}}$ is a predicate measure if a motion plan is comfortable, which has parameters $\boldsymbol{\theta} = [\theta_{a_f}, \theta_{a_b}, \theta_{a_l}, \theta_{a_r}]$, representing thresholds for acceleration in forward, backward, left, and right directions. It can be defined as:

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

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 threshold parameters in (12) are differentiable and can be learned from the training demonstration data. The learned parameters $\boldsymbol{\theta}$ are shown in Table I.

The **Standard** parameters of comfortable acceleration are provided in [29] from a user study. We observe the **Learned** parameters are close to **Standard**. In practice, it is noticeable that the acceleration thresholds are hard to estimate. However, the learned parameters are comparable to the standard values

	θ_{a_f}	θ_{a_b}	θ_{a_l}	θ_{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 computed from 5 independent runs.

by learning from data, which indicates the proposed method can learn the parameters characterizing the demonstration data well. The notable standard deviations in Table I reflect inherent variations in driving behavior, from right turns showing higher variation than left turns, to forward acceleration varying more than deceleration. These variations persist despite analyzing over 100 hours of driving data, suggesting they stem from inherent differences in driving environments (e.g., Boston's narrow streets versus Las Vegas's wide boulevards) rather than insufficient data. Potentially, better scenario-based classification could address these variations, but we leave this for future work.

C. Evaluation in Closed-loop Planning

The learned rules are evaluated in a closed-loop planning system using NuPlan under two settings: Closed-Loop Score Non-Reactive (CLS-NR), where other agents follow recorded trajectories, and Closed-Loop Score Reactive (CLS-R), where agents are controlled by an Intelligent Driver Model (IDM) policy. Performance is measured using normalized scores (0-1) across safety (collision avoidance, following distances), rule compliance (lane keeping, speed limits), progress along the route, and comfort (acceleration, jerk), assessing the rules' ability to balance driving requirements in both predictable and interactive scenarios. The results are shown in Table II. Here, we compare our learned Scoring Logic Network (SLN) with the Expert Rules (ER) in PDM [2] and the Neural Critic (NC) [5]. The last two rows show the overall performance on "All" the scenarios and the Val14 split originally used for evaluating the ER [2].

TABLE II
CLOSED-LOOP PLANNING PERFORMANCE

	Rule Complexity			CLS-NR ↑			CLS-R ↑		
	$|\bar{\mathcal{P}}|$	$|\dot{\mathcal{P}}|$	#. 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 proposal approach we used here is from PDM [2].

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 CLS-NR setting, SLN outperforms ER in 6 out of 9 scenario types, ties in 2, and underperforms in 1, while in CLS-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 the "All" scenarios and the "Val14" [2] split is used to evaluate ER, in both reactive and non-reactive settings. Following NuPlan's log frequency [1], we run the closed-loop planning system at 20 Hz. To conserve computational resources, we also evaluated performance at 10 Hz and 5 Hz. Results show minimal performance impact, with scores the same on the "All" scenarios and minor degradation (<0.2, CLS-NR: 0.93, CLS-R: 0.91) on the "Val14" split for both reduced frequencies.

D. Ablation Studies

1) Proposal Approach: A robust rule should above all be able to filter out undesirable plans for different proposal approaches. We evaluate the learned rules on different proposal approaches, including PDM [2], AT-Sampler [5], Hybrid-PDM [2], and ML-Prop [30]. For both rule-based and learning-based proposal approaches, 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 [2]	0.90	0.79	**0.92**	0.91	0.81	**0.93**
AT-Sampler [5]	0.83	0.81	**0.89**	0.84	0.80	**0.90**
Hybrid-PDM [2]	0.90	0.79	**0.92**	0.91	0.79	**0.92**
ML-Prop* [30]	0.81	0.75	**0.86**	0.82	0.76	**0.85**

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

2) Number of Proposal Candidates: As shown in 4, the performance of both CLS-NR and CLS-R improves significantly when increasing proposals from 3 to 15 (CLS-NR: 0.81 → 0.92, CLS-R: 0.79 → 0.93), but plateaus beyond 15 proposals, showing only minimal gains up to 30. Therefore, 15 proposals were used in the main experiments.
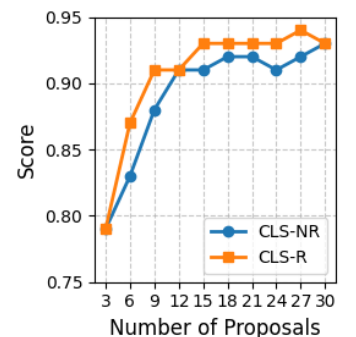

Fig. 4. Proposal Number Ablation

3) Regularization Hyperparameters: We conducted an ablation study to examine the impact of regularization hyperparameters α and β on learning performance. "Conv" means the rate of convergence measured by the number of epochs before the validation loss stops decreasing for 10 epochs. "Triv" is the ratio of learned rules that are trivial (e.g., $\top \rightarrow \top$) across

α	Conv	Triv	β	Conv	Triv
10^{-4}	99	0	10^{-2}	131	0
10^{-5}	37	0	10^{-3}	37	0
10^{-6}	24	0.4	10^{-4}	27	0.5
10^{-7}	13	0.9	10^{-5}	12	1
0	11	1	0	11	1

10 runs in the "Following" scenario. While varying α (β), the value of β (α) is fixed to 10^{-3} (10^{-5}). When $\alpha = 0$ or $\beta = 0$ (i.e. no regularization), the model converges quickly (11 epochs) but learns only trivial rules. Larger α (10^{-7} to 10^{-4}) and β (10^{-5} to 10^{-2}) values increase convergence time but reduce trivial rules. We found $\alpha = 10^{-5}$ and $\beta = 10^{-3}$ an optimal choice, achieving convergence in 37 epochs with no trivial rules, balancing training efficiency and rule quality.

VI. CONCLUSION, LIMITATIONS, AND FUTURE WORK

This paper introduces 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 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 thus 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. A current limitation of FLoRA is that it can only discover relationships between provided predicates, which are manually designed. Future work will focus on extending FLoRA with LLM-aided predicate discovery and incorporating accident data as they become available through real-world autonomous driving deployments.

REFERENCES

[1] 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.

[2] 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.

[3] 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 *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, 2023.

[4] 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.

[5] 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.

[6] L. Chen, P. Wu, K. Chitta, B. Jaeger, A. Geiger, and H. Li, "End-to-end autonomous driving: Challenges and frontiers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[7] S. Akhauri, L. Y. Zheng, and M. C. Lin, "Enhanced transfer learning for autonomous driving with systematic accident simulation," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[8] 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 *ACM/IEEE 12th Int. Conf. on Cyber-Physical Sys.*, 2021.

[9] 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 Int. J. of Connect. & Automat. Vehicles*, 2020.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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 *CoRL*, 2021.

[15] 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.

[16] 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.

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

[18] 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*, 2021.

[19] 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.

[20] 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," *Int. J. of Robotics Research*, 2017.

[21] 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.

[22] E. Bartocci, C. Mateis, E. Nesterini, and D. Nickovic, "Survey on mining signal temporal logic specifications," *Inf. Comput.*, 2022.

[23] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *IJCAI*, 2013.

[24] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theor. Comput. 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] K. Leung, N. Aréchiga, and M. Pavone, "Backpropagation through signal temporal logic specifications: Infusing logical structure into gradient-based methods," *Int. J. of Robotics Research*, 2020.

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

[28] B. Chen, F. Zhang, A. Nguyen, D. Zan, Z. Lin, J.-G. Lou, and W. Chen, "CodeT: Code generation with generated tests," *ArXiv*, 2022.

[29] 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.

[30] 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.