An Interpretable Deep Reinforcement Learning Approach to Autonomous Driving

Zhihao Song^1 , **Yunpeng Jiang**^{1} , **Jianyi Zhang**^{1} , **Paul Weng**^{1} , **Dong Li**^{2} , **Wulong Liu**^{2} and **Jianye Hao**^{2,3}
^{1}University of Michigan Joint Institute, Shanghai Jiao Tong University
^{2}Huawei Noah's Ark Lab
^{3}School of Computing and Intelligence, Tianjin University
{18202196246, jyp9961, zhangjy97, paul.weng}@sjtu.edu.cn, {lidong106, liuwulong, haojianye}@huawei.com

Abstract

This paper investigates a deep reinforcement learning (RL) approach for autonomous driving. Since interpretability is essential for such a high-stake domain, in contrast to previous deep RL work, we exploit a recent neuro-symbolic model called differentiable logic machine to learn an interpretable controller in the form of a first-order logic program. As a proof of concept, we demonstrate the feasibility of our approach on two classical decision-making scenarios in autonomous driving: lane changing and intersection management. Our preliminary results obtained in a simple simulator suggest that learning an interpretable controller does not penalize performance. Moreover, since the controller is a logic program, it is understandable and is amenable to analysis.

1 Introduction

Deep Reinforcement learning (RL) has shown great potential in various applications [Silver *et al.*, 2017; Rong, 2017; Levine *et al.*, 2016] thanks to the generality of RL and the approximation power of deep learning. Thus, it has started to be also investigated in autonomous driving [Kiran *et al.*, 2020]. Although deep RL is a promising machine learning technique, the learned controllers (or policies) are generally black boxes since they are usually large neural networks.

However, for a successful application in a high-stake domain such as autonomous driving, the interpretability of the learned controller becomes essential (see Section 2). Indeed, before any deployment, a controller should be inspectable and verifiable, which would help to achieve trust in the system. During its operation, its decision-making process should be robust and understandable. Moreover, in case of an accident, the decisions made by the autonomous car should be traceable and explainable.

To achieve interpretability, some recent neuro-symbolic methods have been developed where an interpretable model is learned in the form of a first-order logic program. These approaches usually consists in formulating the problem as an inductive logic programming (ILP) problem [Muggleton, 2019] (see Section 3.2) In this paper, using a model called differentiable logic machine (DLM) [Zimmer *et al.*, 2021] (see Section 3.4), we evaluate the feasibility of learning a logic-based controller in two classical decision-making scenarios in autonomous driving [Urmson *et al.*, 2009]: lane changing and intersection management.

Main Contributions Our contributions can be summarized as follows. We formulate the autonomous driving scenarios as ILP problems (see Section 4). We evaluate DLM as a solution method to solve those problems. We compare it with several relevant state-of-the-art methods and demonstrate its good performance (see Section 5). Since the solution yielded by DLM is interpretable, we provide it as a reference and discuss it.

2 Related Work

Numerous methods based on RL [Kiran *et al.*, 2021] and imitation learning [Pomerleau, 1988; Bojarski *et al.*, 2016] with neural network-based controllers have been proposed for autonomous driving. Due to the black-box nature of neural networks, recent research has started to investigate interpretable methods in this domain. For instance, Schmidt *et al.* [2021] investigate an interpretable and verifiably-safe reinforcement learning approach for self-driving cars using decision trees. Kim and Canny [2017] develop an interpretable end-to-end reinforcement learning by visualizing causal attention. Chen *et al.* [2020] propose a joint learning of environment model and driving policy using graphical models in sequential latent reinforcement learning. In contrast, to the existing work, we aim to learn a policy in the form of a logic program, assuming that the observations can be encoded in first-order logic. Such a program could then be debugged if necessary, verified, used to explain the decisions of the autonomous system.

Given the importance of explainable and interpretable artificial intelligence [Barredo Arrieta *et al.*, 2019], various other approaches have been proposed for interpretable RL. For instance, PIRL [Verma *et al.*, 2018] or its extension [Anderson *et al.*, 2020] learn a policy in the form of a computer program starting from a domain-specific language and a program sketch. Similar neuro-symbolic approaches have also been tried in autonomous driving [Sun *et al.*, 2020]. In contrast to those approaches, the DLM model [Zimmer *et al.*, 2021] that we adopt in this work can learn a logic controller that is independent of the number of objects (e.g., vehicles). In particular, it can learn from situations with a small number of objects and generalize to cases with a much larger number of objects. Although, we do not exploit this feature in our current work, it may be important for autonomous driving, since the number of objects that the self-driving car will interact with cannot be known in advance. Furthermore, DLM can be trained via online RL, whereas other neuro-symbolic work often relies on imitation learning.

3 Background

In this section, we present the necessary background knowledge and notations used in later sections.

3.1 First Order Logic

First Order Logic (FOL) [Barwise, 1977] is a formal language that describes the world (i.e., domain of discourse) in terms constants (or objects) and their relations. Formally, it is defined with the following elements:

- A *constant* corresponds to an object of the world. A set $\mathcal{C}$ of constant is given, which contains all the possible objects. Its size is denoted $|\mathcal{C}| = m$.
- A *variable* represents an unspecified constant.
- An r-ary *predicate* $P(x_1, \dots, x_r)$ corresponds to a relation between the r terms x_1 to x_r, which can be constants or variables. A predicate with all its arguments provided as constants is called grounded. Its value can be evaluated to True or False.

FOL generally also contains functions (i.e., mappings from constants to constants), which are a way to define constants from other constants. Following previous work, we only consider a fragment of FOL, where functions are not allowed. Using predicates, one can recursively define more complex formulas using (existential $\exists$ or universal $\forall$) quantifiers and logic operators (e.g., conjunction $\wedge$, disjunction $\vee$, negation $\neg$, implication $\rightarrow$ or biconditional $\leftrightarrow$).

3.2 Inductive Logic Programming

Inductive Logic Programming (ILP) [Muggleton, 2019] is a machine learning framework where given a set of positive examples and negative examples expressed in first-order logic, the goal is to learn a logic formula that entails all positive examples and none of the negative ones. ILP is a natural approach for explainable AI since understanding logic programs is generally much easier than neural network models; it is also applicable in transfer learning thanks to its symbolic representation. Solving an ILP problem is in general hard since it amounts to searching in a combinatorial discrete space of programs whose size grows exponentially with the depth of the program. In this work, we use a recent model, Differentiable Logic Machine, to solve our ILP problems.

3.3 Reinforcement Learning

Reinforcement learning (RL) tasks can be formulated with a Markov Decision Process (MDP) model [Bellman, 1957], which is defined as a tuple $(\mathcal{S}, \mathcal{A}, T, r, \mu, \gamma)$ where $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, transition function T specifies the probability of the next state $s' \in \mathcal{S}$ from the previous state and action $s \in \mathcal{S}, a \in \mathcal{A}$, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function, μ is a distribution of all initial states, and $\gamma \in [0, 1)$ is a discount factor for calculating the return of a state-action sequence. A policy π is a mapping from states to actions (or more generally probability distributions over actions), specifying how actions are selected based on current states. The goal in RL is generally expressed as learning a policy π^* that maximizes the expected discounted total reward:

$$\pi^* = \arg\max_{\pi} \left\{ \mathbb{E}_{\mu,T,\pi} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \right\} \tag{1}$$

where t is the time step in one episode of simulation, $\mathbb{E}_{\mu,T,\pi}$ is the expectation w.r.t. initial state distribution μ, transition function T and policy π.

Deep RL is a combination of reinforcement RL and deep learning. In deep RL, a policy is usually represented by a differentiable model (e.g., neural network), which is trained to obtain a good performance in terms of expected discounted total reward. Such a model corresponds in fact to a policy $\pi_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta}$, which corresponds to trainable weights. One popular state-of-the-art deep RL algorithm for training $\pi_{\boldsymbol{\theta}}$ is the Proximal Policy Optimization (PPO) algorithm [Schulman *et al.*, 2017], which we use in our experiments.

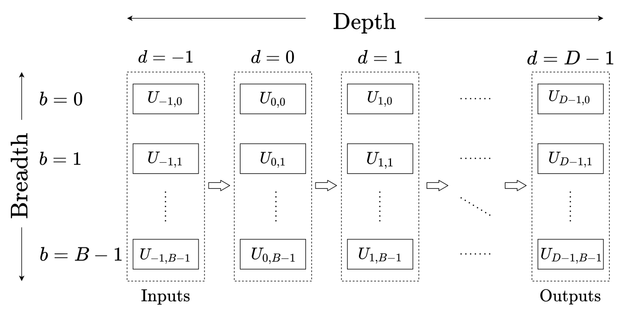
3.4 Differentiable Logic Machine

Differentiable Logic Machine (DLM) [Zimmer *et al.*, 2021] is a recent differentiable neural-symbolic architecture. It builds on Neural Logic Machine (NLM) [Dong *et al.*, 2019], which is neural network architecture with a strong logic architectural bias to simulate forward chaining. In contrast to NLM, a trained DLM model (if trained successfully) can provide a solution as a FOL formula. We overview how DLM works below.

Architecture. A DLM model (illustrated in Figure 1) is specified by a max depth D and a max breadth B. It consists of $(D + 1) \times B$ units organized into layers. The units are indexed by both depth $d \in \{-1, 0, 1, \cdots, D - 1\}$ ($d = -1$ represents the inputs and $d = D - 1$ represents the last layer, which provides the output of the model) and breadth $b \in \{0, 1, \cdots, B - 1\}$.

For depth $d \geq 0$, a unit (see Figure 1, right) at breadth b, denoted by $U_{d,b}$, takes as inputs the b-ary predicates obtained from layer $d - 1$ and outputs $N_{d,b}$ new (invented) b-ary predicates whose truth values depend on the input predicates. Those output predicates can be denoted as $P_{b,d,i}$ with $i \in \{0, 1, \cdots, N_{d,b} - 1\}$. Concretely, those units process predicates as tensors. The shape of the tensor output by $U_{d,b}$ is $[N_{d,b}, m, m-1, \cdots, m-b+1]$ where m is the total number of objects in the problem and $[m, m-1, \cdots, m-b+1]$ corresponds to all the permutations of b objects. It can be seen as $N_{d,b}$ tensors with size $[m, m-1, \cdots, m-b+1]$, each represents the truth values of a b-ary predicate in a fixed permutation order of input objects.

The components of a tensor correspond to the probability that a corresponding predicate is true for some constants of the problem. In training process, the probabilities are float numbers between 0 and 1 for gradient descent method. In



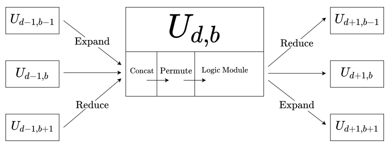


Figure 1: DLM model with max breadth B and max depth D: (left) overall architecture; (right) a unit in DLM and its workflow: If $b = 0$, then no inputs from expansion and no outputs to reduction; if $b = B - 1$, then no inputs from reduction and no outputs to expansion; if $d = -1$, then no previous layer and no inputs; if $d = D - 1$, then no next layer and no outputs.

test environment and real usage, the probabilities in every unit are rounded to Boolean values, 0 for false and 1 for true. For conciseness, we refer to those probabilities simply as truth values below.

For $d \geq 0$, the b-ary input predicates of unit $U_{d,b}$ come from the predicates output by units $U_{d-1,b-1}$ (if $b > 0$), $U_{d-1,b}$, and $U_{d-1,b+1}$ (if $b < B - 1$) of the previous layer. Those predicates are either computed by the corresponding units ($d - 1 \geq 0$) or directly corresponds to the initial predicates ($d-1 = -1$). Since $U_{d-1,b-1}$ (resp. $U_{d-1,b+1}$) outputs $b-1$-ary (resp. $b+1$) predicates, they are first processed by an *expansion* (resp. *reduction*) operator (see below) to increase (resp. decrease) the arity of those predicates. Moreover, all the input predicates are also processed by a *permutation* operator (see below) to increase the potential expressivity of the output predicates.

Expansion. Expansion is the operator changing an r-ary predicate P^r to an $r + 1$-ary predicate P^{r+1}, where the additional object does not influence the truth value. The formula can be given as follows:

$$P^{r+1}(X_1, \cdots, X_{r+1}) = P^r(X_1, \cdots, X_r) \tag{2}$$

and we can denote it by $\text{Expand}\,(P^r)$.

Reduction. Reduction is the operator changing an r-ary predicate P^r to an $r - 1$-ary predicate P^{r-1}. There are two methods of reduction, using existential quantifier or universal quantifier respectively, with respect to the eliminated variable. The formula can be given as follows:

$$P_1^{r-1}(X_1, \cdots, X_{r-1}) = \exists X_r P^r(X_1, \cdots, X_r) \tag{3}$$
$$P_2^{r-1}(X_1, \cdots, X_{r-1}) = \forall X_r P^r(X_1, \cdots, X_r) \tag{4}$$

and we can denote them as $\exists\,(P^r)$ and $\forall\,(P^r)$ respectively.

Permutation. Permutation is an operator applied to all input predicates. The number of predicates generated by permutation is equal to the number of permutation of the variables, i.e., for arity r, it is $r!$. One permuted predicate can be written as follows:

$$P^r_\sigma(X_1, \cdots, X_r) = P^r(X_{\sigma^1}, \cdots, X_{\sigma^r}) \tag{5}$$

where $(\sigma^1, \cdots, \sigma^r)$ is one permutation of $\{1, \cdots, r\}$.

In the experiments of this paper, the max breadth B is chosen to be $B = 3$, and the highest arity of predicates is $B - 1 = 2$. For each 2-ary predicate P^2, the permutation operator produces P^2 and another predicate where the two arguments are swapped, which we denote:

$$\text{Rotate}(P^2)(X_1, X_2) = P^2(X_2, X_1). \tag{6}$$

Logic Module. A unit at depth $d \geq 0$ computes its output using a logic module from the input predicates obtained after expansion, reduction, and permutation. A logic module is a fully differentiable learning component parameterized by learnable weights. The weights mapped into $[0, 1]$ using Gumbel-softmaxes [Jang *et al.*, 2017] are then used to select input predicates to be combined with logic operators such as conjunction, disjunction, and negation. In our experiments, each unit $U_{d,b}$ for $d \geq 0$ is designed to learn output predicates of the following forms:

$$P_{d,b,i} = \begin{cases} P_1 \wedge P_2 \\ P_1 \vee P_2 \\ P_1 \wedge \neg P_2 \\ P_1 \vee \neg P_2 \end{cases} \tag{7}$$

with $i \in \{0, 1, \cdots, N_{d,b} - 1\}$ and P_1, P_2 are in the form of

$$P_1, P_2 = \begin{cases} P_{d-1,b,k}(X_{\sigma_t^1}, \cdots, X_{\sigma_t^b}) \\ (\forall P_{d-1,b+1,k})\,(X_{\sigma_t^1}, \cdots, X_{\sigma_t^b}) \\ (\exists P_{d-1,b+1,k})\,(X_{\sigma_t^1}, \cdots, X_{\sigma_t^b}) \\ \text{Expand}\,(P_{d-1,b-1,k})\,(X_{\sigma_t^1}, \cdots, X_{\sigma_t^b}) \end{cases} \tag{8}$$

where k represents the index of the selected predicate and $(\sigma_t^1, \cdots, \sigma_t^b)$ is the index of objects in a selected order.

Concluding Remarks. We summarize the operators used in DLM in Table 1. For more details about DLM, we refer the

Operator	Type	Meaning
$\wedge$	binary	Boolean and
$\vee$	binary	Boolean or
$\neg$	unary	Boolean negation
$\forall$	unary	Reduction
$\exists$	unary	Reduction
Expand	unary	Expansion
Rotate	unary	Permutation

Table 1: Notations for operators in DLM

readers to Zimmer *et al.* [2021] where several training techniques are proposed to find an interpretable solution. In our experiments, we use their online RL training method, which is based on PPO [Schulman *et al.*, 2017] using their novel critic adapted to the logical inputs. This method is designed to make the model converge to an interpretable solution using notably a controlled reduction of Gumbel softmax noise, dropout, and softmax temperatures. After training, the trained DLM model can be significantly reduced by extracting from it the logical formula it encodes.

4 General Approach

In this section, we describe our general approach and discuss our modeling decisions for the experiments.

4.1 Methodology

To apply DLM in autonomous driving, the following steps should be followed:

- List the objects that can appear in the environment,
- List the relevant predicates that are necessary to describe the world,
- Construct a perception module that transform observations to predicate valuations,
- Build and train a DLM,
- Analyze the logic program extracted from the trained DLM model.

Below we discuss those steps in the context of autonomous driving. Notably, we highlight the assumptions made in our preliminary work and discuss possible extensions, which we leave for future work.

For autonomous driving, objects could consist of vehicles, pedestrians, or other potential obstacles. Since DLM requires the predicates to be grounded over all combinations of objects, this can lead to very large inputs if many objects are considered. To alleviate this issue, DLM could be extended to use typed first-order logic where objects can be of different types (e.g., vehicle vs pedestrian). Doing so would allow to restrict the grounding of a predicate only to the relevant combinations of objects depending on the types that are required by that predicate.

In our experiments, we use a simple simulator where only vehicles are present. In this context, our agent simply focuses on the vehicles that are the closest to it. Moreover, we only use predicates that are nullary or unary for the initial predicates to describe the observation of the agent. Binary predicates can be avoided by considering relative distances and speeds. Those simplifying assumptions allow the inputs of the DLM model to be of small size, which is sufficient to demonstrate the feasibility of our approach.

Predicates for autonomous driving need to describe the kinematic information of the agent and its surrounding vehicles. For simplicity, we discretize the ranges of relative positions and speeds. Our approach (and DLM) could be generalized to use an extension of first-order logic where predicates could be for instance parameterized inequalities (e.g., to describe if some safety distance holds), which would avoid the discretization.

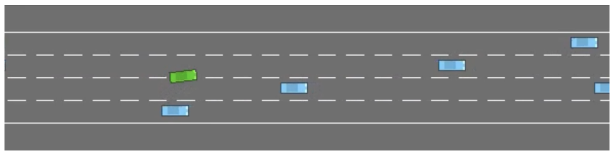


Figure 2: Lane-Changing Scenario: All vehicles drive from left to right. The lanes $L_1 \ldots L_4$ are from top to bottom.

In the description of our experiments that we provide below, we give the complete list of the predicates that we used. Since we need to describe kinematic information and relations between vehicles, we can limit the arity of predicates to a maximum of two. For our experiments, we adapt the list of predicates to the specific scenario under consideration, but more generally, we could consider a generic set of predicates to cover all driving scenarios.

For the DLM model to learn and reason about the road environment, it needs to be provided with predicate groundings over the objects that exist around the agent. The values of those groundings have to be computed from sensors. In practice, this step can be more or less complex depending on the types of sensors (e.g., tachometers or lidar vs video inputs). Since we rely on a simulator, for simplicity, we extract useful information directly from the simulation to compute the grounding values.

For the last two steps, we rely on the DLM model. We build and train it according to the online training method suggested by Zimmer *et al.* [2021], which we overviewed in the previous section. Once the model is trained, we extract the obtained logic program and analyze it.

4.2 Modeling Decisions

The main goal of this work is to show as a proof of concept that an interpretable controller can be learned for autonomous driving using RL training. To demonstrate the feasibility of the approach, we use the Highway-env [Edouard, 2018] environment, which is an autonomous driving simulator for tactical decision-making tasks. Two scenarios, *lane changing* and *intersection management*, are selected in the experiments. We explain next in details how we modeled the two scenarios. Refer to Appendix C in appendix for detailed values of parameters used in initial predicates.

Lane-Changing Scenario

In *lane changing* (see Figure 2 for an illustration), the goal is to drive the vehicle as fast as possible without colliding with other social vehicles. We detail next what the observations, actions, objects, and predicates are in this scenario.

Observations. There are 4 parallel lanes in total, denoted as $L_1, \cdots, L_4$. One vehicle is controllable, denoted as agent A^t, and other social vehicles are denoted as vehicles V_i^t where t is the time step and i is the index of vehicles.

For each vehicle V_i^t, the simulated environment can produce the x-axis coordinate $V_i^t[x]$, the y-axis coordinate $V_i^t[y]$, the x-axis speed $V_i^t[v_x]$, the y-axis speed $V_i^t[v_y]$, and the occupied lane $V_i^t[L]$. The relative distance $V_i^t[\Delta x]$, $V_i^t[\Delta y]$ and relative speed $V_i^t[\Delta v_x]$, $V_i^t[\Delta v_y]$ w.r.t. the agent are also calculated.

Actions. The agent can operate one of the four following actions at each time step, which are acceleration, deceleration, left turn, and right turn.

Objects. To satisfy the inputs requirement of DLM, $m = 4$ vehicles are selected at each time step t to be the input objects, denoted as $O_k^t = V_i^t$ where $k \in \{1, \cdots, m\}$ for some existing vehicle V_i^t in the environment. Intuitively, the V_i^t's are the vehicles that are the closest to the agent in each lane. More specifically, the selection rule is shown below.

1. Categorize vehicles by lanes using $V_i^t[L] = k$.
2. Filter vehicles to only keep those in front of the agent with $V_i^t[\Delta x] > -(d_{\text{safe}} + d_{\text{vehicle}})$, where d_{safe} is the safe distance of social vehicles and d_{vehicle} is the length of social vehicles and the agent.
3. Choose the nearest vehicle w.rt. $V_i^t[\Delta x]$ for each k.
4. The four selected vehicles are denoted as O_k^t in the order of $O_k^t[L] = k$.

Note that it may happen that in some lane k, no vehicles are found. In that case, a dummy object is used for O_k^t. Predicates evaluated on dummy objects yield False.

Predicates. Since all the lanes are parallel with the y-axis, so the relative distance in y-axis represents the same meaning as the relation of lane indexes.

Nullary initial predicates are features related to the whole environment. Predicates 0-2 act as sensors on the agent, detecting whether there is no nearby vehicle on its left lane, on its current lane and on its right lane.

0. $P_{-1,0,0} \leftarrow \neg\exists k\big[(k = A^t[L] - 1) \wedge (-(d_{\text{safe}} + d_{\text{vehicle}}) < O_k^t[\Delta x] < d_{\text{medium}})\big]$
1. $P_{-1,0,1} \leftarrow \neg\exists k\big[(k = A^t[L] + 1) \wedge (-(d_{\text{safe}} + d_{\text{vehicle}}) < O_k^t[\Delta x] < d_{\text{medium}})\big]$
2. $P_{-1,0,2} \leftarrow \neg\exists k\big[(k = A^t[L]) \wedge (-(d_{\text{safe}} + d_{\text{vehicle}}) < O_k^t[\Delta x] < d_{\text{medium}} + d_{\text{vehicle}})\big]$

Predicates 3-5 measure the speed of the agent. v_{s1}, v_{s2} and v_{s3} are three target speed values for the agent defined in the environment.

3. $P_{-1,0,3} \leftarrow A^t[v_x] < \frac{1}{2}v_{\text{s1}} + \frac{1}{2}v_{\text{s2}}$
4. $P_{-1,0,4} \leftarrow \frac{1}{2}v_{\text{s1}} + \frac{1}{2}v_{\text{s2}} \leq A^t[v_x] < \frac{1}{2}v_{\text{s2}} + \frac{1}{2}v_{\text{s3}}$
5. $P_{-1,0,5} \leftarrow A^t[v_x] \geq \frac{1}{2}v_{\text{s2}} + \frac{1}{2}v_{\text{s3}}$

Unary initial predicates are features that are evaluated on one selected object. Predicate 0 is the validation label, indicating whether the vehicle exists or not.

0. $P_{-1,1,0} \leftarrow \text{True}$

Predicates 1-3 determine the relation of lanes w.r.t. the agent.

1. $P_{-1,1,1} \leftarrow k = A^t[L] - 1$
2. $P_{-1,1,2} \leftarrow k = A^t[L]$
3. $P_{-1,1,3} \leftarrow k = A^t[L] + 1$

Predicates 4-8 measures the relative distance in the x-axis. d_{near}, d_{medium}, d_{far} and d_{max} are relative distance thresholds defined from near to the agent to far away from the agent.

4. $P_{-1,1,4} \leftarrow -(d_{\text{safe}} + d_{\text{vehicle}}) < O_k^t[\Delta x] < d_{\text{safe}} + d_{\text{vehicle}}$
5. $P_{-1,1,5} \leftarrow d_{\text{safe}} + d_{\text{vehicle}} \leq O_k^t[\Delta x] < d_{\text{near}}$
6. $P_{-1,1,6} \leftarrow d_{\text{near}} \leq O_k^t[\Delta x] < d_{\text{medium}}$
7. $P_{-1,1,7} \leftarrow d_{\text{medium}} \leq O_k^t[\Delta x] < d_{\text{far}}$
8. $P_{-1,1,8} \leftarrow d_{\text{far}} \leq O_k^t[\Delta x] < d_{\text{max}}$

Predicates 9-13 measures the relative speed in the x-axis. $\Delta[v_x]_{\text{slow}}$, $\Delta[v_x]_{\text{same}}$ and $\Delta[v_x]_{\text{fast}}$. The vehicles slower than the agent should be emphasized, so the range $[\Delta[v_x]_{\text{slow}}, \Delta[v_x]_{\text{same}}]$ is proportionally divided into three subranges.

9. $P_{-1,1,9} \leftarrow O_k^t[\Delta v_x] < \Delta[v_x]_{\text{slow}}$
10. $P_{-1,1,10} \leftarrow \Delta[v_x]_{\text{slow}} \leq O_k^t[\Delta v_x] < \frac{2}{3}\Delta[v_x]_{\text{slow}} + \frac{1}{3}\Delta[v_x]_{\text{same}}$
11. $P_{-1,1,11} \leftarrow \frac{2}{3}\Delta[v_x]_{\text{slow}} + \frac{1}{3}\Delta[v_x]_{\text{same}} \leq O_k^t[\Delta v_x] < \frac{1}{3}\Delta[v_x]_{\text{slow}} + \frac{2}{3}\Delta[v_x]_{\text{same}}$
12. $P_{-1,1,12} \leftarrow \frac{1}{3}\Delta[v_x]_{\text{slow}} + \frac{2}{3}\Delta[v_x]_{\text{same}} \leq O_k^t[\Delta v_x] < \Delta[v_x]_{\text{same}}$
13. $P_{-1,1,13} \leftarrow \Delta[v_x]_{\text{same}} \leq O_k^t[\Delta v_x] < \Delta[v_x]_{\text{fast}}$

Predicates 14-16 measures the relative speed in the y-axis, representing whether the relation of lanes w.r.t. the agent is changing. ε is a small value which is enough to test whether a vehicle is changing its lane.

14. $P_{-1,1,14} \leftarrow O_k^t[\Delta v_y] < -\varepsilon$
15. $P_{-1,1,15} \leftarrow -\varepsilon \leq O_k^t[\Delta v_y] < \varepsilon$
16. $P_{-1,1,16} \leftarrow O_k^t[\Delta v_y] \geq \varepsilon$

Predicates 17-21 measures the predicted relative distance in the x-axis, calculated by the relative distance after two time steps if both the agent and the selected vehicle keep their speed and direction. d'_{near} and d'_{far} represents a dangerous distance and a safe distance of the predicted relative distance. The ranges are divided proportionally.

17. $P_{-1,1,17} \leftarrow O_k^t[\Delta x] + 2O_k^t[\Delta v_x] < d'_{\text{near}}$
18. $P_{-1,1,18} \leftarrow d'_{\text{near}} \leq O_k^t[\Delta x] + 2O_k^t[\Delta v_x] < \frac{2}{3}d'_{\text{near}} + \frac{1}{3}d'_{\text{far}}$
19. $P_{-1,1,19} \leftarrow \frac{2}{3}d'_{\text{near}} + \frac{1}{3}d'_{\text{far}} \leq O_k^t[\Delta x] + 2O_k^t[\Delta v_x] < \frac{1}{3}d'_{\text{near}} + \frac{2}{3}d'_{\text{far}}$
20. $P_{-1,1,20} \leftarrow \frac{1}{3}d'_{\text{near}} + \frac{2}{3}d'_{\text{far}} \leq O_k^t[\Delta x] + 2O_k^t[\Delta v_x] < d'_{\text{far}}$
21. $P_{-1,1,21} \leftarrow O_k^t[\Delta x] + 2O_k^t[\Delta v_x] \geq d'_{\text{far}}$

Intersection Management Scenario

In *intersection management* (see Figure 3 for an illustration, the goal is to cross an intersection without traffic lights as fast as possible while avoiding any collision with other vehicles. For simplicity, we focus on the case where the agent aims to turn left across the intersection, which is the hardest situation. We explain next what the observations, actions, objects, and predicates are in this scenario.

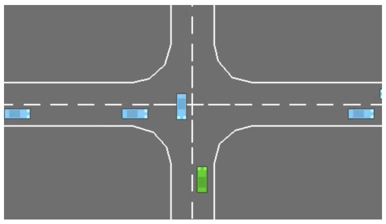


Figure 3: Intersection Management Scenario: The agent arrives from the south and aims to make a left turn.

Observations. There are 4 directions in total, south, east, north, and west, denoted as $D_1, \cdots, D_4$. One vehicle starting in D_1 is controllable, denoted as agent A^t, and other vehicles are social vehicles controlled by the simulator, denoted as vehicles V_i^t where t is the time step and i is the index of vehicles. Similarly to the highway scenario, for each vehicle V_i^t, the environment can produce the x-axis and y-axis coordinates $V_i^t[x]$, $V_i^t[y]$, the x-axis and y-axis speeds $V_i^t[v_x]$, $V_i^t[v_y]$, the direction it comes from $V_i^t[D]$, the relative distance to the agent $V_i^t[\Delta x]$, $V_i^t[\Delta y]$, and relative speed to the agent $V_i^t[\Delta v_x]$, $V_i^t[\Delta v_y]$.

Actions. The agent can execute one of the three longitudinal actions, which are acceleration, keeping the current speed, and deceleration.

Objects. To satisfy the inputs requirement of NLM and DLM, $m = 8$ vehicles are selected at each time step t to be the input objects, denoted as $O_k^t = V_i^t$ where $k \in \{1, \cdots, m\}$ for some existing vehicle V_i^t in the environment. In words, the eight objects represent the nearest vehicles in each direction with respect to the agent. Formally, the selection rule is shown below.

1. Select object only if the agent is close to the intersection, judging by $A^t[y] \leq d_{\text{intersection}}$, where $d_{\text{intersection}}$ is the distance between the entry and the center of the intersection.
2. Categorize vehicles by directions using $V_i^t[D]$ (2 vehicles for each direction).
3. Choose the two nearest vehicles according to $\sqrt{V_i^t[\Delta x]^2 + V_i^t[\Delta y]^2}$ for each direction.
4. The eight selected vehicles are denoted as O_k^t in the order of $O_k^t[D] = \lceil \frac{k}{2} \rceil$.

Moreover, for a specific k, if no vehicle is found, dummy objects are used. As for the other scenario, predicates evaluates to False on dummy objects.

Predicates. Nullary predicates are defined as follows. Predicates 0-7 detect whether there is any vehicle located in a specified nearby area w.r.t. the agent.

0. $P_{-1,0,0} \leftarrow \exists i\big[(-2d_{\text{vehicle}} < O_k^t[\Delta x] \leq -d_{\text{vehicle}}) \wedge (-d_{\text{intersection}} < O_k^t[\Delta y] < d_{\text{intersection}})\big]$
1. $P_{-1,0,1} \leftarrow \exists i\big[(-d_{\text{vehicle}} < O_k^t[\Delta x] \leq 0) \wedge (-d_{\text{intersection}} < O_k^t[\Delta y] < d_{\text{intersection}})\big]$
2. $P_{-1,0,2} \leftarrow \exists i\big[(0 < O_k^t[\Delta x] \leq d_{\text{vehicle}}) \wedge (-d_{\text{intersection}} < O_k^t[\Delta y] < d_{\text{intersection}})\big]$
3. $P_{-1,0,3} \leftarrow \exists i\big[(d_{\text{vehicle}} < O_k^t[\Delta x] < 2d_{\text{vehicle}}) \wedge (-d_{\text{intersection}} < O_k^t[\Delta y] < d_{\text{intersection}})\big]$
4. $P_{-1,0,4} \leftarrow \exists i\big[(-d_{\text{intersection}} < O_k^t[\Delta x] < d_{\text{intersection}}) \wedge (-2d_{\text{vehicle}} < O_k^t[\Delta y] \leq -d_{\text{vehicle}})\big]$
5. $P_{-1,0,5} \leftarrow \exists i\big[(-d_{\text{intersection}} < O_k^t[\Delta x] < d_{\text{intersection}}) \wedge (-d_{\text{vehicle}} < O_k^t[\Delta y] \leq 0)\big]$
6. $P_{-1,0,6} \leftarrow \exists i\big[(-d_{\text{intersection}} < O_k^t[\Delta x] < d_{\text{intersection}}) \wedge (0 < O_k^t[\Delta y] \leq d_{\text{vehicle}})\big]$
7. $P_{-1,0,7} \leftarrow \exists i\big[(-d_{\text{intersection}} < O_k^t[\Delta x] < d_{\text{intersection}}) \wedge (d_{\text{vehicle}} < O_k^t[\Delta y] < 2d_{\text{vehicle}})\big]$

Predicates 8-10 measure the speed of the agent. v_{slow} and v_{fast} separates the speed range to slow, medium and fast for intersection-management scenario.

8. $P_{-1,0,8} \leftarrow A^t[v_x] \leq v_{\text{slow}}$
9. $P_{-1,0,9} \leftarrow v_{\text{slow}} < A^t[v_x] \leq v_{\text{fast}}$
10. $P_{-1,0,10} \leftarrow A^t[v_x] > v_{\text{fast}}$

Predicates 11-13 measure the position of the agent, indicating whether it is near the intersection or not.

11. $P_{-1,0,11} \leftarrow 0 \leq A^t[y] \leq d_{\text{intersection}}$
12. $P_{-1,0,12} \leftarrow -d_{\text{intersection}} \leq A^t[x] \leq 0$
13. $P_{-1,0,13} \leftarrow (A^t[x] \geq -d_{\text{intersection}}) \wedge (A^t[y] \leq d_{\text{intersection}})$

Unary input predicates are related to properties of the selected objects. Predicates 0-3 specify the coming direction.

0. $P_{-1,1,0} \leftarrow O_k^t[D] = 1$
1. $P_{-1,1,1} \leftarrow O_k^t[D] = 2$
2. $P_{-1,1,2} \leftarrow O_k^t[D] = 3$
3. $P_{-1,1,3} \leftarrow O_k^t[D] = 4$

Predicates 4-9 measure the predicted relative distance in the x-axis after one time step. $d^{\text{face}}_{\text{near}}$, $d^{\text{face}}_{\text{medium}}$ and $d^{\text{face}}_{\text{far}}$ specify four ranges of predicted relative distance, for those vehicles facing with the direction of the agent (i.e., vehicles from north and west). $d^{\text{back}}_{\text{near}}$, $d^{\text{back}}_{\text{medium}}$ and $d^{\text{back}}_{\text{far}}$ are chosen for those vehicles orienting the back of the agent (i.e., vehicles from south and east).

4. $P_{-1,1,4} \leftarrow -d^{\text{face}}_{\text{far}} \leq O_k^t[\Delta x] + O_k^t[\Delta v_x] \leq 0$
5. $P_{-1,1,5} \leftarrow -d^{\text{face}}_{\text{medium}} \leq O_k^t[\Delta x] + O_k^t[\Delta v_x] \leq 0$
6. $P_{-1,1,6} \leftarrow -d^{\text{face}}_{\text{near}} \leq O_k^t[\Delta x] + O_k^t[\Delta v_x] \leq 0$
7. $P_{-1,1,7} \leftarrow 0 \leq O_k^t[\Delta x] + O_k^t[\Delta v_x] \leq d^{\text{back}}_{\text{near}}$
8. $P_{-1,1,8} \leftarrow 0 \leq O_k^t[\Delta x] + O_k^t[\Delta v_x] \leq d^{\text{back}}_{\text{medium}}$
9. $P_{-1,1,9} \leftarrow 0 \leq O_k^t[\Delta x] + O_k^t[\Delta v_x] \leq d^{\text{back}}_{\text{far}}$

Predicates 10-15 measure the predicted relative distance in the y-axis after one time step.

10. $P_{-1,1,10} \leftarrow -d^{\text{face}}_{\text{far}} \leq O_k^t[\Delta y] + O_k^t[\Delta v_y] \leq 0$
11. $P_{-1,1,11} \leftarrow -d^{\text{face}}_{\text{medium}} \leq O_k^t[\Delta y] + O_k^t[\Delta v_y] \leq 0$
12. $P_{-1,1,12} \leftarrow -d^{\text{face}}_{\text{near}} \leq O_k^t[\Delta y] + O_k^t[\Delta v_y] \leq 0$

13. $P_{-1,1,13} \leftarrow 0 \leq O_k^t[\Delta y] + O_k^t[\Delta v_y] \leq d^{\text{back}}_{\text{near}}$
14. $P_{-1,1,14} \leftarrow 0 \leq O_k^t[\Delta y] + O_k^t[\Delta v_y] \leq d^{\text{back}}_{\text{medium}}$
15. $P_{-1,1,15} \leftarrow 0 \leq O_k^t[\Delta y] + O_k^t[\Delta v_y] \leq d^{\text{back}}_{\text{far}}$

5 Experimental Results

Regarding the architecture size of DLM, we set the breadth of the model to be $B = 3$ and the depth to be $D = 7$ to ensure enough expressivity. Thus, since $B = 3$, the invented predicates can be nullary, unary, or binary predicates, although the initial predicates are only nullary and unary. Moreover, $D = 7$ means that apart from the initial predicates, 7-layer predicates will be invented iteratively. A larger depth than necessary is not problematic because a logic unit can preserve one of its input predicates P as one of its outputs by learning expressions such as $P \wedge P$, $P \vee P$. For lane-changing scenario (resp. intersection-management scenario), the number of output predicates for each unit is set to $N_{d,b} = 8$ (resp. $N_{d,b} = 12$). For each logic unit, there are two (resp. three) predicates for each type of connections (see Section 3.4).

As baselines to compare with our DLM-based approach, we consider two different models. First, we selected a standard neural network (NN) to serve as a basic baseline. To be fair, the NN is given as inputs the same information as DLM, but in the form of a feature vector. Second, we selected the state-of-the-art neuro-symbolic model, NLM [Dong *et al.*, 2019]. Since NLM and DLM share a similar architecture, we used the same network size (i.e., smae B, D, and $N_{d,b}$) and the same initial predicates for both of them. All the models are trained independently. For the two baselines, we used the PPO algorithm [Schulman *et al.*, 2017] while DLM is trained using the procedure proposed by Zimmer *et al.* [2021].

We present below the experimental results in the two scenarios we selected. Notably, for each of them, we first describe the performance of the different models with respect to several metrics, then we provide and discuss the simplified logic formulas learned by DLM.

5.1 Lane-Changing Scenario

Performance. After the training process, the three models are tested in the same test environment, with a maximum of 100 time steps in each episode. The comparison is shown in Table 2.

Quantities	NN	NLM	DLM
avg time steps	93.43	96.8	96.77
crash rate	7.2%	4.8%	3.7%
avg speed	22.22	25.20	27.26
avg distance	2060.91	2418.56	2615.34
avg acceleration	43.43	69.50	71.78
avg deceleration	42.58	20.26	16.59
avg left turn	3.65	3.65	4.49
avg right turn	3.77	3.39	3.91

Table 2: Test evaluation on the lane-changing scenario

The quantities named **avg time steps**, **avg speed** and **avg distance** correspond to the average lasting time steps, average speed, and average passed distance for each episode respectively. The **crash rate** is the percentage of episodes that end in a crash. Note that some crashes are unavoidable because the other cars are controlled by a simple fixed policy. The quantities named **avg acceleration**, **avg deceleration**, **left turn**, and **right turn** provide statistics about how often actions are selected by the trained model.

Rule Extraction. After training, we extracted the learned rules and simplified them (see Appendix A for more details). The obtained controller can then be written as follows:

$$
\begin{aligned}
P_{\text{dec}} &= \neg P_{-1,0,2} \wedge \neg P_{-1,0,0} \wedge \neg P_{-1,0,1} \\
P_{\text{left}} &= \neg P_{-1,0,2} \wedge P_{-1,0,0} \\
P_{\text{right}} &= \neg P_{-1,0,2} \wedge \neg P_{-1,0,0} \wedge P_{-1,0,1} \\
P_{\text{acc}} &= P_{-1,0,2}
\end{aligned}
$$

Interpretation. The learned rule for lane-changing scenario is fairly simple. The decision making is only related to three inputs, $P_{-1,0,0}, P_{-1,0,1}, P_{-1,0,2}$, which detect whether there is a nearby vehicle on the left lane, on the current lane and on the right lane in front of the agent. Under the trained model, the agent accelerates if no front vehicle on the current lane is detected; if that is not possible, it tries to turn left if no front vehicle on the left lane is detected; otherwise, it tries to turn right if no front vehicle on the right lane is detected; as a last option, it decelerates if the three lanes are all blocked.

Analysis. The learned rule is reassuringly simple and reasonable. Although the logic rule is intuitively obvious, it may have some limitations. Notably, it does not try to plan ahead by considering cars farther. This may be due to our simplifying modeling assumptions. More importantly, the model decides the action without any information about speed or relative speed. This implies that the model cannot tell apart the difference between fast vehicles and slow vehicles, which may potentially lead to unexpected problem like colliding with a very slow car.

Motivated by the success of these preliminary experiments, some of our simplifying modeling decisions could be relaxed and made more realistic. For example, in this experiment, the information of the current lane of the agent could be added, so that the relation of lanes between social vehicles and the agent could be learned. The speed range of all cars could be increased and the discretization could be made finer. Moreover, more cars could be selected so that the decision-making of the agent could be less myopic.

5.2 Intersection Management Scenario

Performance. After training, the three models are tested in the same test environment, with a maximum of 100 time steps in each episode. The comparison is shown in Table 3.

The quantity named **avg time steps** gives the average lasting time steps for each episode respectively. The quantities, **crash rate**, **timeout rate**, and **success rate**, are the percentages of episodes that the agent respectively ends in a crash, reaches the maximum steps, or passes the intersection. Due to the difficulty of the intersection management scenario, the issue of crashes and timeouts due to other cars is even more acute than in the previous scenario, as also noted by the author

Quantities	NN	NLM	DLM
avg time steps	103.72	83.06	96.76
crash rate	10.4%	9.5%	6.5%
timeout rate	16.4%	7.5%	11.6%
success rate	73.2%	83.0%	82.0%
avg acceleration	59.47	45.10	45.04
avg deceleration	6.92	16.80	51.73
avg keep speed	37.33	21.16	0.00

Table 3: Test evaluation on the intersection management scenario

of the highway-env simulator. The quantities, **avg acceleration**, **avg deceleration**, and **keep speed**, reflect the general distribution of actions selected by the trained model.

Rule Extraction. Using the same techniques of rule extraction as in the lane-changing scenario, the predicates for actions in FOL form can be derived:

$$P_{\text{dec}} = P_1 \vee P_2 \vee P_3 \tag{9}$$

$$P_{\text{keep}} = \text{False} \tag{10}$$

$$P_{\text{acc}} = \neg P_{\text{dec}} \tag{11}$$

$$\begin{aligned} P_1 =& \big[P_{-1,0,0} \vee \exists P_{-1,1,5} \vee \forall\,(P_{-1,1,11} \vee P_{-1,1,3})\big] \wedge \\ & \big[\neg P_{-1,0,0} \wedge P_{-1,0,7} \wedge \neg P_{-1,0,8}\big] \qquad (12)\\ P_2 =& \big[\forall(P_{-1,1,11} \vee P_{-1,1,3}) \vee P_{-1,0,0} \vee \exists P_{-1,1,5}\big] \wedge \\ & \big[\exists P_{-1,1,3} \vee \exists P_{-1,1,4}\big] \qquad (13)\\ P_3 =& \big[\forall(P_{-1,1,11} \vee P_{-1,1,3}) \vee P_{-1,0,0} \vee \exists P_{-1,1,5}\big] \wedge \\ & \big[(\exists P_{-1,1,11} \wedge P_{-1,0,1}) \vee \\ & \quad [P_{-1,0,0} \wedge (\exists P_{-1,1,12} \vee \neg P_{-1,0,6})]\big] \qquad (14) \end{aligned}$$

Interpretation. The logic program for the intersection scenario is much more complex. From the overview of the three actions, we can notice that action keeping speed is never considered by the model. A long logic expression determine whether the acceleration or the deceleration is selected.

The logic expression of P_{dec} is a disjunctive clause composed of three components. It shows that three situations, represented by P_1, P_2, P_3 leads to a deceleration operation. Moreover, one specific term $P_{-1,0,0} \vee \exists P_{-1,1,5} \vee \forall\,(P_{-1,1,11} \vee P_{-1,1,3})$ occurs in all the three predicates. It can be concluded that this term of great importance is a necessary condition for deceleration. Predicate $P_{-1,0,0}$ detects whether there is a nearby vehicle in the west in some distance range ($-10 < \Delta x \leq -5$) w.r.t. the agent. Predicate $\exists P_{-1,1,5}$ detects whether there is any vehicle that may be located in the west of the agent after one time step. Predicate $\forall\,(P_{-1,1,11} \vee P_{-1,1,3})$ detects whether all the vehicles are from the west or may be located in the north of the agent. The compound term can be seen as a detection of surrounding vehicles. The simplest case is that no vehicle is detected around the agent, so that the model accelerates until passing the intersection.

The remaining parts of logic programs filter the situation after detecting nearby vehicles. The simplest situation is $\exists P_{-1,1,3} \vee \exists P_{-1,1,4}$ (from P_2), which selects the deceleration action if any vehicle is from the west or in the west or drives to the west w.r.t. to the agent. The situation $\neg P_{-1,0,0} \wedge P_{-1,0,7} \wedge \neg P_{-1,0,8}$ represents another simple case that detects whether there is a nearby vehicle in the north (from $P_{-1,0,7}$) when the vehicle is in a relatively high speed (from $\neg P_{-1,0,8}$). The last situation $(\exists P_{-1,1,11} \wedge P_{-1,0,1}) \vee [P_{-1,0,0} \wedge (\exists P_{-1,1,12} \vee \neg P_{-1,0,6})]$ (from P_3) can be divided into two parts. One is $\exists P_{-1,1,11} \wedge P_{-1,0,1}$ checking if there is an extremely close car in the west. The other is $P_{-1,0,0} \wedge (\exists P_{-1,1,12} \vee \neg P_{-1,0,6})$, which is not very meaningful. Predicate $P_{-1,0,0}$ is discussed above and $\exists P_{-1,1,12} \vee \neg P_{-1,0,6}$ measures if a vehicle is predicted to be on the north side of the agent without having a close range.

Analysis. The DLM model has a relatively good performance in this scenario, meaning that the input predicates selected by the logic programs are representative. These inputs may help better understand some important features of the environment. From the explanations above, it is apparent that vehicles in the west and north are much more important. This is because the driving direction of the agent is to north-west, and vehicles from the north side and west side are more likely to have a collision with the agent. Moreover, the selected predicates have a longer thresholds for detecting vehicles located in the west or in the north, because those vehicles may block the road. Both the agent and the social vehicles are not able to drive backward, so they can get stuck in the middle of the intersection until the end of an episode. The model learns to wait until the intersection becomes free enough in case of such bad situations.

6 Conclusion

This preliminary work provides a basic proof of concept for interpretable reinforcement learning in autonomous driving, in particular for tactical decision-making. By describing the autonomous vehicle's environment in first-order logic (FOL), a pure FOL program can be learned using the DLM model. Our experiments demonstrate that our approach is competitive against standard deep reinforcement learning and state-of-the-art neuro-symbolic methods, while having the added benefit of yielding an interpretable controller. Representative information and potential problems could then be detected by analyzing the learned logic programs.

Our discussion of our modeling decisions and our analysis of the learned interpretable controllers already suggest various potential improvements of our methodology, such as the introduction of finer-grained predicates. Furthermore, future work may also include the use of verification tools to guarantee the good properties of the obtained logic programs or the generation of explanations in natural language to articulate the decision-making process of the agent while it is driving.

Acknowledgments

This work is funded by Huawei. We thank anonymous reviewers for their helpful comments. We also thank Junqi Chen, Shensong Zhao, Yingtao Shen and Yulin Gao for meaningful discussions at the very beginning of the research.

References

Greg Anderson, Abhinav Verma, Isil Dillig, and Swarat Chaudhuri. Neurosymbolic Reinforcement Learning with Formally Verified Exploration. In *NeurIPS*, 2020.

Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador Garcia, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. *Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI.* 2019.

Jon Barwise. An introduction to first-order logic. In *Studies in Logic and the Foundations of Mathematics*, volume 90, pages 5–46. 1977.

Richard Bellman. A markov decision process. *Journal of mathematics and mechanics*, 6:679–684, 1957.

Mariusz Bojarski, David W. del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *ArXiv*, abs/1604.07316, 2016.

Jianyu Chen, Shengbo Eben Li, and Masayoshi Tomizuka. Interpretable end-to-end urban autonomous driving with latent deep reinforcement learning. *ArXiv*, abs/2001.08726, 2020.

Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. *ArXiv*, abs/1904.11694, 2019.

Leurent Edouard. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env, 2018.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

Jinkyu Kim and John F. Canny. Interpretable learning for self-driving cars by visualizing causal attention. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969, 2017.

Bangalore Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Kumar Yogamani, and Patrick P'erez. Deep reinforcement learning for autonomous driving: A survey. *ArXiv*, abs/2002.00444, 2020.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 2016.

Stephen Muggleton. Inductive logic programming. *New Generation Computing*, 267:1–38, 2019.

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

Jin Rong. Deep learning at Alibaba. In *International Joint Conference on Artificial Intelligence*, 2017.

Lukas M. Schmidt, Georgios Kontes, Axel Plinge, and Christopher Mutschler. Can you trust your autonomous car? interpretable and verifiably safe reinforcement learning. In *2021 IEEE Intelligent Vehicles Symposium (IV)*, pages 171–178, 2021.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, pages 354–359, 2017.

Jiankai Sun, Hao Sun, Tian Han, and Bolei Zhou. Neurosymbolic program search for autonomous driving decision module design. In *coRL*, 2020.

Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas M. Howard, Sascha Kolski, Alonzo Kelly, Maxim Likhachev, Matt McNaughton, Nick Miller, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, William Whittaker, Ziv Wolkowicki, Jason Ziglar, Hong Bae, Thomas Brown, Daniel Demitrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha Sadekar, Wende Zhang, Joshua Struble, Michael Taylor, Michael Darms, and Dave Ferguson. *Autonomous driving in urban environments: Boss and the urban challenge*, page 1–59. Springer, 2009.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. *ArXiv*, abs/1804.02477, 2018.

Matthieu Zimmer, Xuening Feng, Claire Glanois, Zhaohui Jiang, Jianyi Zhang, P. Weng, Hao Jianye, Li Dong, and Liu Wulong. Differentiable logic machines. *ArXiv*, abs/2102.11529, 2021.

A Equations for simplifying extracted rules

Let $P = P^r(x_1, \cdots, x_r)$ is an r-ary predicate, $F(P)$ is a predicate composed by P and other predicates, with operators excluding dimensional operators $(\forall, \exists, \mathrm{Expand}, \mathrm{Rotate})$. With respect to the operation and notation mentioned in Table 1 Notation, the following statements hold true.

- $P \wedge F(P) = P \wedge F(\mathrm{True})$
- $P \vee F(P) = P \vee F(\mathrm{False})$
- $\forall(\mathrm{Expand}P) = \exists(\mathrm{Expand}P) = P$
- for $r \geq 1$, $\neg(\forall P) = \exists(\neg P)$
- for $r \geq 1$, $\neg(\exists P) = \forall(\neg P)$
- for $r = 0$, $\mathrm{Rotate}\,(\mathrm{Expand}(\mathrm{Expand}P)) = \mathrm{Expand}(\mathrm{Expand}P)$
- for $r = 2$, $\forall\,(\mathrm{Rotate}(\mathrm{Expand}P)) = \mathrm{Expand}(\forall P)$
- for $r = 2$, $\exists\,(\mathrm{Rotate}(\mathrm{Expand}P)) = \mathrm{Expand}(\exists P)$
- for $r \geq 1$, $P \wedge \mathrm{Expand}(\exists P) = P$
- for $r \geq 1$, $P \wedge \mathrm{Expand}(\forall P) = \mathrm{Expand}(\forall P)$
- for $r \geq 1$, $P \vee \mathrm{Expand}(\exists P) = \mathrm{Expand}(\exists P)$
- for $r \geq 1$, $P \vee \mathrm{Expand}(\forall P) = P$

Let $P_1 = P_1^r(x_1, \cdots, x_r)$, $P_2 = P_2^r(x_1, \cdots, x_r)$ are two r-ary predicate. With respect to the operation and notation mentioned in Table 1 Notation, the following statements hold true.

- $\neg(P_1 \wedge P_2) = \neg P_1 \vee \neg P_2$
- $\neg(P_1 \vee P_2) = \neg P_1 \wedge \neg P_2$
- $\mathrm{Expand}(P_1 \wedge P_2) = \mathrm{Expand}P_1 \wedge \mathrm{Expand}P_2$
- $\mathrm{Expand}(P_1 \vee P_2) = \mathrm{Expand}P_1 \vee \mathrm{Expand}P_2$
- for $r \geq 1$, $\forall(P_1 \wedge P_2) = \forall P_1 \wedge \forall P_2$
- for $r \geq 1$, $\exists(P_1 \vee P_2) = \exists P_1 \vee \exists P_2$

Let $P_1 = P_1^r(x_1, \cdots, x_r)$ is an r-ary predicate and $P_2 = P_2^{r+1}(x_1, \cdots, x_{r+1})$ is an $r+1$-ary predicate. With respect to the operation and notation mentioned in Table 1 Notation, the following statements hold true.

- $\forall(\mathrm{Expand}P_1 \vee P_2) = P_1 \vee \forall P_2$
- $\exists(\mathrm{Expand}P_1 \wedge P_2) = P_1 \wedge \exists P_2$

B Hyperparameters in training

For lane-changing scenario, the detailed hyperparameters are shown in Table 4. Specifications for highway-env [Edouard, 2018] is shown in Table 5. Hyperparameters and specifications for intersection-management scenario is shown in Table 6 and Table 7.

Quantities	Values
DLM depth	7
DLM breadth	3
DLM output dimension	8
learning rate	0.005
maximum steps	100000

Table 4: Hyperparameters in lane-changing scenarios

Quantities	Env1	Env2
maximum episode steps	50	100
speed reward coefficient	0.4	0.4
collision reward coefficient	0.3	0.3
collision penalty	0	-50
policy frequency	1	1

Table 5: Specifications of environments in lane-changing scenarios

Quantities	Values
DLM depth	7
DLM breadth	3
DLM output dimension	12
learning rate	0.005
maximum steps	100000

Table 6: Hyperparameters in intersection-management scenarios

Quantities	Env1	Env2
maximum episode steps	100	100
acceleration reward	0.2	0
arrival reward	50	50
collision penalty	-500	-500
policy frequency	4	4

Table 7: Specifications of environments in lane-changing scenarios

C Parameters

Table 8 presents the detailed values for parameters used in initial predicates for our experiments.

lane-changing		**intersection management**	
Quantities	Values	Quantities	Values
d_{safe}	1	$d_{\text{intersection}}$	20
d_{vehicle}	5	d_{vehicle}	5
v_{s1}	20	v_{slow}	2
v_{s2}	25	v_{fast}	4
v_{s3}	30	$d_{\text{near}}^{\text{back}}$	5
d_{near}	18		
d_{medium}	25	$d_{\text{medium}}^{\text{back}}$	15
d_{far}	50		
d_{max}	80	$d_{\text{far}}^{\text{back}}$	30
$\Delta[v_x]_{\text{slow}}$	-9		
$\Delta[v_x]_{\text{same}}$	0	$d_{\text{near}}^{\text{face}}$	15
$\Delta[v_x]_{\text{fast}}$	1		
ε	0.2	$d_{\text{medium}}^{\text{face}}$	28
d'_{near}	4		
d'_{far}	10	$d_{\text{far}}^{\text{face}}$	50

Table 8: Parameters used in initial predicates for two scenarios