Symbolic Imitation Learning: From Black-Box to Explainable Driving Policies

Iman Sharifi and Saber Fallah

***Abstract*—Current methods of imitation learning (IL), primarily based on deep neural networks, offer efficient means for obtaining driving policies from real-world data but suffer from significant limitations in interpretability and generalizability. These shortcomings are particularly concerning in safety-critical applications like autonomous driving. In this paper, we address these limitations by introducing Symbolic Imitation Learning (SIL), a groundbreaking method that employs Inductive Logic Programming (ILP) to learn driving policies which are transparent, explainable and generalisable from available datasets. Utilizing the real-world highD dataset, we subject our method to a rigorous comparative analysis against prevailing neural-network-based IL methods. Our results demonstrate that SIL not only enhances the interpretability of driving policies but also significantly improves their applicability across varied driving situations. Hence, this work offers a novel pathway to more reliable and safer autonomous driving systems, underscoring the potential of integrating ILP into the domain of IL.**[1]

***Index Terms*—Symbolic Imitation Learning, First-Order Logics, Autonomous Driving**

I. Introduction

The development of autonomous driving (AD) technologies has brought the transportation systems to the next level, promising safer and more efficient roadways. Among the various techniques used to enable autonomous vehicles (AVs), imitation learning (IL) has emerged as a promising approach due to its ability to learn complex driving behaviors from expert demonstrations. By leveraging large-scale driving datasets and deep neural networks (DNNs), IL has demonstrated remarkable success in training autonomous agents to emulate humans' driving behaviors [1]–[3].

Despite its impressive performance, DNN-based IL(DIL) inherits the critical limitations of DNNs, hindering its widespread adoption in real-world autonomous systems. A prominent drawback is the lack of interpretability in the learned driving policies due to the black-bax nature of DNNs [4], making the decision-making process of AD challenging to understand. This lack of interpretability not only limits our ability to diagnose failures or errors but also hampers the trust and acceptance of autonomous systems by society and regulatory bodies [5]. Moreover, the generalizability of DIL driving policies remains a concern. Although they can learn to imitate expert drivers in specific scenarios (with a given state distribution), adapting these policies to unseen situations (out of state distribution) can be problematic since DIL can just learn the demonstrated behaviors [6], thereby having little knowledge about novel situations when mismatching test and train state distributions [7]. The rigid nature of learned policies often leads to suboptimal or unsafe behavior when encountering unfamiliar conditions, such as adverse weather or complex traffic scenarios. Finally, though more somple-efficient as compared to reinforcement learning, DIL methods still suffer from sample-inefficiency since they need millions of state-action pairs to learn efficiently [8], [9].

Recently, state-of-the-art explainable AI (XAI) methods seek to improve the transparancy and interpretability of DNNs by providing various criteria [10], [11], most cutting-edge criterion of which is a neuro-symbolic learning method aiming at combining the learning capabilities of DNNs with symbolic reasoning [12], [13]. Since most neuro-symbolic methods employ symblic logic-based reasoning to extract domain-specific first-order logical (FOL) rules using different rule-generation techniques [14], [15], they are identified as sample-efficient techniques that have a credible sense of generalizability [13]. For instance, [15] proposed a differentiable logic machine (DLM), as a neural-logic approach, to extract FOL rules using inductive logic programming (ILP) [16], as a technique aiming at inducing new rules from human-provided examples. Similarly, DLM was employed by [17] to learn an interpretable controller for AD in the form of a first-order program. Likewise, our work aims to replace DIL with ILP-based programs, proposing a sample-efficient and transparent method while transferable to unseen scenarios.

In this paper, we propose a novel solution to address the challenges of the current DIL methods by introducing the concept of SIL, as an approach aiming to convert black-box driving policies into explainable policies by incorporating ILP to extract symbolic rules from humans' background knowledge. Using ILP, we seek to unlock the potential for transparent, interpretable, and adaptive driving policies, laying the foundation for safer and more reliable AD systems. The primary objective of this paper is twofold: first, to enhance the transparency of learned driving policies by inducing human-readable and interpretable rules that capture essential safety, legality, and smoothness aspects of driving behavior. Second, we strive to improve the generalizability of these policies, enabling AVs to handle diverse and challenging driving scenarios with reliability.

The rest of the paper is divided into several sections. Section II introduces the prerequisites of the method. Section

I. Sharifi and S. Fallah {i.sharifi, s.fallah}@surrey.ac.uk are with the Connected and Autonomous Vehicles Lab (www.cav-lab.io) at the Department of Mechanical Engineering Sciences, University of Surrey, Guildford, UK.

[1]The source code and data are available on this link: https://github.com/iman-sharifi-ghb/SIL.

III describes the proposed method in general and sention IV particularly discusses it for the AD system. Section V presents the simulation environment, results, and discussion. Finally, section VI draws a conclusion.

II. Preliminaries

In this section, we provide the necessary background for the proposed method.

A. First-Order Logic

FOL is a formalism that uses facts and rules to represent knowledge and perform logical inferences [9]. Rules in this framework consist of a head and a body like the following format:

$$\texttt{head :- body.} \tag{1}$$

where `:-` indicates entailment operator ($\models$) in this context [16]. `head` signifies an output predicate expressing a relationship between concepts, and `body` outlines conditions for the `head` predicate's validity. Predicates consist of a functor and a number of arguments, depicted as $\texttt{functor(arg}_1\texttt{,arg}_2\texttt{,...,arg}_n\texttt{)}$ or `functor/n`, where arguments are variables or constants, and `n` is an arity. FOL rules usually follow Horn clause patterns, having a single head literal and zero or more literals in the body:

$$\texttt{H :- B}_1\texttt{,B}_2\texttt{,...,B}_n\texttt{.} \tag{2}$$

$\texttt{B}_i, i = 1, n$ show premises in `body` where comma (`,`) stands for conjunction ($\wedge$) in Prolog programming. Also, `H`, the head, serves as a conclusion for the rule. This rule implies that if $\texttt{B}_1$ to $\texttt{B}_n$ conjunctively hold, then `H` is `true`; otherwise, `H` is `false`. FOL-based rules enable the definition of intricate relationships and logical inferences by employing the existing knowledge.

B. Inductive Logic Programming (ILP)

ILP is a machine learning paradigm that combines FOL with inductive reasoning to learn symbolic rules from examples, including a set of positive examples and, optionally, negative examples [16]. These examples are typically represented as tuples of ground literals, where a ground literal is an atom with all its variables instantiated to constants.

The ILP setting involves three key components: language bias set $\mathcal{B}$, background knowledge set $\mathcal{BK}$, and a set of examples $\mathcal{E}$. Set $\mathcal{B}$ specifies the space of possible hypotheses which are rules that the ILP system considers during learning. Using a desired head predicate and multiple potential body predicates, it guides the search for candidate rules that could potentially explain the given examples. Being domain-specific, set $\mathcal{BK}$ provides additional information about the domain in which the ILP operates. This knowledge may include known relationships, constraints, or regularities that can be used to infer new rules. Examples $\mathcal{E}$, including positive examples $\mathcal{E}^+$ and negative examples $\mathcal{E}^-$, are the observed instances that the ILP system aims to learn from.

The main goal of an ILP system is to discover hypotheses $\mathcal{H}$ that explain $\mathcal{E}^+$ while avoiding $\mathcal{E}^-$. ILP scans $\mathcal{E}^+$ and attempts

Algorithm 1 Inductive Logic Programming (ILP)

```
Require: $\mathcal{B}$, $\mathcal{BK}$, $\mathcal{E}^+$, $\mathcal{E}^-$
Ensure: $\mathcal{H}$
 1: $\mathcal{H}_{old}$ = []
 2: for $e^+$ in $\mathcal{E}^+$ do
 3:     for $b$ in $\mathcal{B}$ do
 4:         if $b$ can cover $e^+$ according to $\mathcal{BK}$ then
 5:             Add $b$ to $\mathcal{H}_{old}$
 6:         end if
 7:     end for
 8: end for
 9: for $e^-$ in $\mathcal{E}^-$ do
10:     for $h$ in $\mathcal{H}_{old}$ do
11:         if $h$ covers $e^-$ according to $\mathcal{BK}$ then
12:             Remove $h$ from $\mathcal{H}_{old}$
13:         end if
14:     end for
15: end for
16: for $e^+$ in $\mathcal{E}^+$ do
17:     $\mathcal{H}_{new}$ = []
18:     for $h$ in $\mathcal{H}_{old}$ do
19:         if $h$ can cover $e^+$ according to $\mathcal{BK}$ then
20:             $h_r$ is $h$ after refinement
21:             Add $h_r$ to $\mathcal{H}_{new}$
22:         end if
23:     end for
24: end for
25: $\mathcal{H}$ = $\mathcal{H}_{new}$
```

to match them with all $b \in \mathcal{B}$. If b can cover an example based on $\mathcal{BK}$, it is added to the list of learned hypotheses. In the second step, hypotheses that cover $\mathcal{E}^-$ are removed from the original hypotheses list. Next, the algorithm refines hypotheses based on $\mathcal{E}^+$ and $\mathcal{BK}$. For each hypothesis that covers a positive example $e^+ \in \mathcal{E}^+$, refinement is done by generating a more specific hypothesis with new literals, extending its coverage on examples. Algorithm 1 indicates the pseudo-code of ILP.

ILP leverages the power of FOL and inductive reasoning to learn interpretable and generalizable symbolic rules from a small example space $\mathcal{E}$, even a single example [18]. By iteratively refining candidate rules, guided by $\mathcal{E}^+$ and $\mathcal{BK}$, ILP offers a principled approach for learning explainable policies in complex domains like AD.

C. Imitation Learning

IL is a prominent technique in the field of machine learning and robotics, which enables an agent to learn complex behaviors by imitating expert demonstrations. In DIL, the agent employs DNNs to learn from a rich dataset $\mathcal{D}$ containing expert demonstrations, represented as state-action pairs: $\{(s_1, a_1), (s_2, a_2), \ldots, (s_T, a_T)\}$. Here, s_t denotes the state of the environment at time step t, and a_t represents the action taken by the expert in that state. The goal of DIL is to learn a policy $\pi_\theta(a|s)$, parameterized by θ, that can map states to appropriate actions in a manner consistent with the expert behavior. One common approach to train the imitation policy is through supervised learning. The learning process involves minimizing the discrepancy between the actions selected by the learned policy and the actions taken by the expert in the demonstration data. This discrepancy is usually measured using a mean squared error (MSE) loss function, as shown in Eq. 3. The objective is to find the optimal parameters θ^* that minimize the loss over the dataset $\mathcal{D}$:

$$\theta^* = \arg\min_{\theta} \sum_{(s,a)\in\mathcal{D}} (\pi_\theta(a|s) - a)^2 \tag{3}$$

DIL has shown remarkable success in various domains, including AD. By learning from expert drivers' demonstrations, DIL can effectively navigate complex traffic scenarios. However, one significant challenge with DIL approaches is the lack of interpretability. As the policies are typically encoded in DNNs, understanding the decision-making process of the autonomous agent becomes difficult. Additionally, DIL cannot perform well out of the state distribution, thereby lacking generalization in unseen situations.

III. Methodology: Symbolic Imitation Learning

In this section, we introduce the fundamental concept of the SIL framework, which represents a novel approach leveraging ILP to extract symbolic policies from human-generated background knowledge. This process aims to replicate human behaviors by uncovering explicit rules governing complex actions demonstrated by humans.

As illustrated in Fig. 1, the SIL framework includes three main components: knowledge acquisition, rule induction, and rule aggregation. In the knowledge acquisition phase, we provide essential inputs using our background knowledge about the environment for ILP, including $\mathcal{B}$, $\mathcal{BK}$, and $\mathcal{E}$ (see algorithm 1), facilitating the induction of a single rule in the rule induction phase. For all required rules, this process is repeated to accumulate all necessary rules induced by ILP, gradually assembling individual rules at each stage to constitute $\mathcal{H}$ set. Finally, the rule aggregation component employs and supplements these rules to extract desired actions. These actions are inherently human-like and interpretable since the logical rules are derived from human demonstrations in the rule induction phase. Leveraging ILP's ability to deduce logical rules from examples, SIL effectively captures intricate human-like behaviors, a task that conventional DIL methods often struggle to achieve.

SIL offers several advantages, notably its capacity for sample-efficient learning. It demonstrates the ability to construct a set of interpretable rules with a relatively small number of expert demonstrations. These rules not only enhance human-comprehensible decision-making but also contribute to the model's overall generalizability. The explicit nature of these symbolic rules equips SIL-based agents to adapt more effectively to previously unseen scenarios, surpassing the performance of black-box policies, which lack transparency and struggle with generalization.

IV. SIL for Autonomous Driving

In this section, we employ SIL to derive unknown rules for autonomous highway driving, a use-case scenario. Our objective is to imitate the behaviors of human drivers by utilizing SIL using multiple sets of provided examples. Firstly, we provide the reader with the necessary context about the highway environment by introducing useful predicates. Then, we delve into the essential components of the SIL framework within the context of autonomous highway driving, explaining how we abtain the desired actions in various states.

It is noted that the primary goal of employing SIL in autonomous highway driving is to find rules from human-derived background knowledge and extract actions that emulate human-like behavior. These actions include lane changes and adjustments of the AV's longitudinal velocity, all of which are employed to ensure safe, efficient, and smooth driving. Keep note that when the AV executes lane changes, it typically employs one of three actions: lane keeping (LK), right lane change (RLC), or left lane change (LLC).

Introduction: human drivers frequently change their vehicle's lane and velocity based on the positions and speeds of neighboring vehicles, also known as target vehicles (TVs). Accordingly, we leverage the relative positions and velocities of TVs around the AV to make informed decisions regarding lane changes and velocity adjustments. To achieve this, we define the sections surrounding the AV where vehicles may be present, dividing the vicinity into 8 distinct `sectors`: `front`, `frontRight`, `right`, `backRight`, `back`, `backLeft`, `left`, and `frontLeft`. Each of these sectors can either be occupied by a vehicle, denoted by the `sector_isBusy` predicate set to `true`, or unoccupied, where the predicate is set to `false`. For instance, `right_isBusy` is `true` if the there is a vehicle in the `right` section, otherwise, it is set to `false`.

In terms of the relative longitudinal velocities of the TVs, we calculate the difference between their velocities and that of the AV, then assigning predicates accordingly. Three predicates are considered for relative velocity within each sector. When the relative velocity is a positive (negative) value more than a predefined threshold, `bigger` (`lower`) is used in the corresponding predicate. Also, when the absolute value of the relative velocity falls below the threshold, the velocities are categorized as `equal`. For example, `frontVel_isBigger`, `frontVel_isEqual`, and `frontVel_isLower` show the relative velocity between the `front` TV and the AV. In addition, human drivers should stay between the highway lanes (valid sections) and avoid driving in off-road areas (invalid sections). To incorporate this contextual knowledge, each sector is marked as either valid (`sector_isValid` is set to `true`) or invalid (`sector_isValid` is set to `false`).

A. Knowledge Acquisition

AD systems need various rules to perform well on the roads, and each rule requires identical settings and examples. Thus, to extract diverse rules, it is necessary to define distinct settings and employ unique examples for each rule. Thus, we start

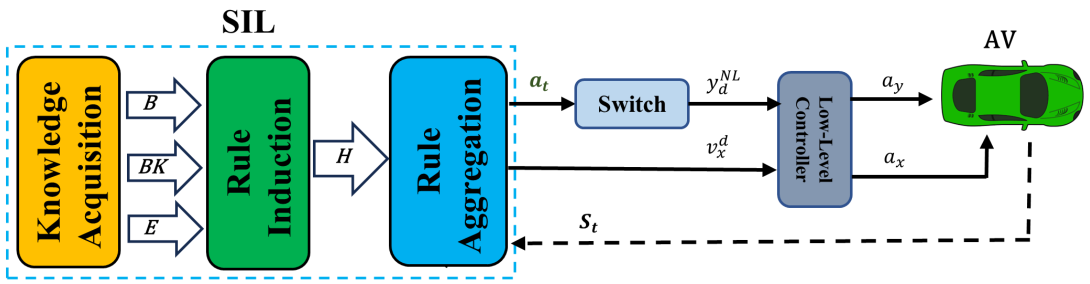

Fig. 1: Decision-making using symbolic imitation learning framework in autonomous driving

with a process that defines the scope of each desired rule by defining their head and candidate body predicates in a specific $\mathcal{B}$ set. The selection of candidate body predicates depends on their potential influence on the accuracy of the head predicate. So, we remove the body predicates that have no effects on the head predicats, thereby reducing the dimension of $\mathcal{B}$ set, which could otherwise lead to long processing times.

Generally, our objective revolves around the identification of unsafe, dangerous, and efficient lane change actions, coupled with effective adjustments in longitudinal acceleration (smoothness). As shown in Table I, our aim is to find rules associated with *unsafe*, *dangerous*, *efficient*, and *smooth* actions relating to both lane and acceleration changes. Each rule, similar to the rule form in Eq. 2, owns an exclusive head predicate defined in $\mathcal{B}$ using `head_pred/1` and a set of candidate body predicates specified with the `body_pred/1` predicate. Candidate body predicates include all literals having an impact on the head predicate. ILP should find the best body predicates that satisfy all $e^+ \in \mathcal{E}^+$ while simultaneously rejecting all $e^- \in \mathcal{E}^-$. To achieve this, we define multiple possible states using our background knowledge and assign a unique ID to each state. Based on our knowledge about each action, we categorize each state as either e^+ using the `pos/1` predicate or e^- using the `neg/1` predicate. This process is repeated for each rule to collect sufficient knowledge-based data for inducting previously-unknown logical rules. Therefore, it becomes necessary to provide specific $\mathcal{B}$, $\mathcal{BK}$, and $\mathcal{E}$ for the effective acquisition of each rule.

The knowledge acquisition process creates various real-world states that include all possible driving scenarios. Each state consists of an AV surrounded by 8 sections, each of which categorized as either occupied by a TV or vacant. Thus, we add 8 `sector_isBusy` literals to the candidate body predicates (`sector` is a representative for 8 sections around the AV). Furthermore, we consider the relative velocity of each TV concerning the AV within each state, adding 24 relative velocity literals (3 for each `sector`) to candidate body predicates. Additionally, we incorporate the validity of the right and left sections in each state to ensure compliance with legal driving areas, adding 2 literals accordingly. Table I indicates the total number of candidate body predicates for each head predicate. After the definition of states, we assign positive or negative labels to the head predicates of the unknown rules, according to our driving expertise and corresponding to the rule being extracted. For instance, to instruct an ILP program in deriving unsafe RLC rules, states wherein taking RLC action is unsafe receive positive labels; otherwise, such scenarios receive negative labels.

The quantity of positive examples ($N_{\mathcal{E}^+}$) and negative examples ($N_{\mathcal{E}^-}$) have a big impact on rule accuracy and robustness. While the inclusion of $\mathcal{E}^-$ is optional, its presence improve the robustness and generalizability of the derived rules across a broader distribution of states. Notably, as defined in Table I, distinct values for $N_{\mathcal{E}^+}$ and $N_{\mathcal{E}^-}$ are assigned to each rule.[2]

B. Rule Induction

After obtaining sufficient amount of knowledge-driven data for each unknown rule, we elaborate on the rule induction stage of the SIL framework, discussing the extracted rules for autonomous highway driving. Notably, we extract the desired rules using Popper, a powerful ILP system that combines answer set programming[3] (ASP) [19], [20] and Prolog to enhance the learning accuracy and speed. One of the main advantages of Popper is the ability to learn from failures (LFF) by defining, constraining, and searching the hypothesis space using ASP (generate stage), testing hypotheses against $\mathcal{E}^+$ and $\mathcal{BK}$ using Prolog (test stage), and pruning the hypothesis space using the failed hypotheses (constrain stage) [21].

Due to the dynamic nature of driving decision-making, various rules can be induced for different tasks. According to what we need for the highway driving task, we just induce the required general rules using ILP, of which *safety*, *efficiency*, and *smoothness* ones are discussed.

1) Safety: We predominantly focus on safety rules concerning lane change scenarios, which can be categorized into two primary parts: unsafe rules and dangerous rules. Each of the actions, RLC and LLC, can potentially fall into the categories of unsafe or dangerous in various states, while LK is generally

[2]All knowledge-based data are available at github.com/iman-sharifi-ghb/Symbolic-Imitation-Learning/tree/main/SIL/data

[3]Being particularly useful for solving combinatorial and knowledge-intensive problems, ASP is a declarative programming paradigm that allows encoding complex problems as logic-based rules and constraints.

TABLE I: Important information about the rules in the knowledge acquisition and rule induction components

RULE	ACTION	HEAD	N_{BP}	$N_{\mathcal{E}+}$	$N_{\mathcal{E}-}$	HYPOTHESIS	ACCURACY	$T_e(sec)$
UNSAFE	RLC	`rlc_isUnsafe`	38	768	256	h1	1.00	0.365
	LLC	`llc_isUnsafe`	38	768	256	h2	1.00	0.345
DANGEROUS	LK	`lk_isDangerous`	43	87	937	h3	1.00	0.762
	RLC	`rlc_isDangerous`	38	320	704	h4	1.00	0.416
	LLC	`llc_isDangerous`	38	308	717	h5	1.00	1.007
EFFICIENCY	RLC	`rlc_isBetter`	25	32	992	h6	1.00	4.108
	LLC	`llc_isBetter`	25	128	896	h7	1.00	0.892
SMOOTHNESS	CATCH-UP	`reachDesiredSpeed`	43	512	512	h8	1.00	0.336
	FOLLOW-UP	`reachFrontSpeed`	43	253	771	h9	1.00	0.703
	BRAKE	`brake`	43	253	771	h10	1.00	0.326

considered safe, except for situations where it could pose potential danger.

Unsafe Actions: the objective here is to reveal previously unknown rules by employing relevant examples for states where taking RLC or LLC would be unsafe and lead to collisions or crashes. Employing ILP, we have deduced two rules, one relating to unsafe RLC (rule `h1`) and the other to unsafe LLC (rule `h2`). For instance, rule `h1` indicates when taking RLC is unsafe (`rlc_isUnsafe`):

```
%% Rule h1: unsafe RLC
rlc_isUnsafe:-
  right_isBusy;not(right_isValid).
```

where `;` ($\vee$) and `not` ($\neg$) represent disjunction and negation operators in FOL formalism, respectively. Similarly, rule `h2` stands for the moments that taking LLC (`llc_isUnsafe`) is unsafe for the AV:

```
%% Rule h2: unsafe LLC
llc_isUnsafe:-
  left_isBusy;not(left_isValid).
```

To elaborate, in a given state, if a TV is present in the `right` (`left`) side of the AV or the `right` (`left`) section is invalid, taking the RLC (LLC) action is considered unsafe, as it would lead to collisions. Therefore, it is essential to avoid taking RLC (LLC) action in such states.

Dangerous Actions: dangerous actions include situations where the AV takes actions that potentially jeopardize the safety of its passengers. For instance, consider a state where a car occupies the `backRight` (`backLeft`) section of the AV, and its velocity surpasses that of the AV. In such scenarios, taking RLC (LLC) action is permissible but may result in an accident with the car in the `backRight` (`backLeft`) section. While not completely unsafe, it is advisable to avoid taking dangerous actions in these circumstances.

Using the Popper ILP system, we have derived three rules for dangerous actions in each state. The first one (rule `h3`) indicates that if there is a vehicle in the `backRight` section with a velocity exceeding that of the AV, or if there is a vehicle in the `frontRight` section with a velocity lower than that of the AV, then executing the RLC action is dangerous:

```
%% Rule h3: dangerous RLC
rlc_isDangerous:-
  backRight_isBusy,backRightVel_isBigger;
  frontRight_isBusy,frontRightVel_isLower.
```

Similarly, rule `h4` has been established for the LLC action:

```
%% Rule h4: dangerous LLC
llc_isDangerous:-
  backLeft_isBusy,backLeftVel_isBigger;
  frontLeft_isBusy,frontLeftVel_isLower.
```

Lastly, rule `h5` takes into account scenarios in which LK may pose danger:

```
%% Rule h5: dangerous LK
lk_isDangerous:-
  back_isBusy,
  not(backDist_isSafe),backVel_isBigger.
```

This rule asserts that if there is a TV in the `back` section, the distance between the AV and the TV is critical, and the TV's velocity exceeds that of the AV, staying in the current lane (LK) is considered dangerous and could lead to a collision.

2) Efficiency: Safety rules identify lane change actions that are either unsafe or dangerous in specific states. However, they do not provide guidance on which action is efficient when none of them are unsafe or dangerous. To enhance the decision-making process, we introduce prioritization for actions. Generally, to create time-efficient driving scenarios, it is necessary to change lanes appropriately in order to minimize the driving time. Yet it is important to avoid unnecessary and sudden lane changes to otherwise cause health-related issues for passengers. Therefore, the first priority is to avoid changing lanes unless it is necessary. If the AV needs to change lanes or overtake a vehicle in the `front` section, we introduce another priority. When the AV intends to change lanes and has the option to choose either RLC or LLC, it is preferable to choose the LLC action, as the left lane typically has a higher speed. Additionally, RLC should be chosen when LLC is not allowed. Overall, LLC takes precedence over RLC, helping to prioritize actions when both are equally safe. Following these priorities, we label scenarios extracted from knowledge-driven data, with the aim of determining the preferable actions using

ILP. Rule `h6` indicates a scenario where LLC takes precedence over RLC:

```
%% Rule h6: efficient LLC
llc_isBetter:-
  front_isBusy,not(left_isBusy),
  not(frontLeft_isBusy).
```

This rule states that when there is a vehicle in front of the AV, and the `left` as well as `frontLeft` sections are not occupied (free), then LLC is better than RLC. Conversely, rule `h7` represents situations where the RLC action is preferable:

```
%% Rule h7: efficient RLC
rlc_isBetter:-
  front_isBusy,left_isBusy,
  frontLeft_isBusy,not(right_isBusy),
  not(frontRight_isBusy).
```

This rule specifies that if the `front`, `left`, and `frontLeft` sections are occupied, and the `right` and `frontRight` sections are not occupied, then RLC is favored over LLC. Notably, `rlc_isBetter` and `llc_isBetter` are mutually exclusive, meaning that if RLC is better, then LLC cannot be better, thereby preventing conflicts among the rules.

3) Smoothness: To make sure about passengers' health and overall safety, the AV, similar to human drivers, must change their vehicles' velocity smoothly. Human drivers typically employ a limited set of actions to control their vehicle's velocity while experiencing smooth driving. One such action involves increasing their velocity to reach their desired speed slowly, while another is to adjust their speed according to the lead vehicle (`front` TV) while closing the distance between them. In emergency situations, such as when the distance between their vehicle and the `front` TV is unsafe and their vehicle's velocity exceeds that of the `front` TV, they resort to brake pedal to avoid potential collisions. Based on these real-world driving experiences, our aim is to identify three fundamental rules that significantly impact driving safety and smoothness.

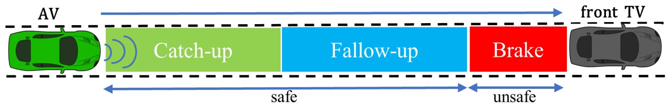

Fig. 2: Acceleration phases: catch-up (for reaching driver's desired speed), follow-up (for adjusting speed the front TV's speed), and brake (for necessary deceleration).

In our prior work [22], we introduced a rule-based acceleration methodology for velocity control in autonomous vehicles (AVs). The results were promising, with the AV successfully avoiding collisions with the `front` TV while maintaining smooth velocity adjustments. We achieved this by employing a low-level controller to ensure velocity commands lack discontinuities. As represented in Eq. 4, our approach includes three distinct phases, inspired by human driver behavior when driving in a lane, similar to [23]. As illustrated in Fig. 2, the `catch-up` phase corresponds to scenarios where the `front` section is unoccupied by a TV. In such cases, each driver has the freedom to decide whether to accelerate, decelerate, or maintain their current velocity. To calculate the required acceleration in this scenario, we incorporate a desired longitudinal velocity term (V_x^d), which may vary among individual drivers, to determine the necessary acceleration for reaching the desired velocity. When a vehicle is present in front of human drivers' vehicles, they often adjust their velocities to match the `front` TV's speed, thereby avoiding collisions. The `follow-up` phase applies to situations where a TV occupies the `front` section of the AV and the distance is safe. As shown in Eq. 4, the `follow-up` equation adjusts the AV's velocity to the `front` TV's velocity ($v_{x,TV}$) while reducing the distance (D) between them. Finally, as shown in Fig. 2, when drivers find themselves at an unsafe distance (the distance that is less than a critical distance C) from the `front` TV and at a high speed, they must decelerate to avoid collision with the lead vehicle. In such cases, they utilize the brake pedal to decelerate until a safe following distance is achieved. The `brake` condition in Eq. 4 associates with the scenarios wherein braking is the best policy for velocity control.

$$a_x = \begin{cases} \frac{V_x^d - v_{x,AV}}{\Delta t} & \texttt{catch-up} \\ \frac{v_{x,TV}^2 - v_{x,AV}^2}{2(D-C)} & \texttt{follow-up} \\ \frac{-v_{x,AV}^2}{2D} & \texttt{brake} \end{cases} \tag{4}$$

As stated before, we aim to find rules to formulate accelerations in Eq. 4. Since existing ILP methods cannot directly capture literals with numerical values (except for 0 and 1), the extraction of rules represented in Eq. 4 poses a challenge. However, we can identify the `true` phase among `catch-up`, `follow-up`, and `brake`. Using the identified phase, we can determine the desired acceleration and then the velocity for each state. To establish corresponding rules to each phase, we consider the relative distance between the AV and the `front` TV and classify states as positive or negative examples for each rule, as the rules specifications shown in Table I. This approach enables us to extract an unknown rule for each phase using Popper. Three rules have been induced for adjusting a_x in each phase; the first rule, for instance, indicates that when the `front` section is unoccupied (free), accelerations should be adjusted to obtain the desired velocity.

```
%% Rule h8: Catch-up
reachDesiredSpeed:-not(front_isBusy).
```

The second rule indicates that when the `front` section is busy and the relative longitudinal distance between the two vehicles is safe, then the AV should change the acceleration to reach the `front` TV's speed.

```
%% Rule h9: Follow-up
reachFrontSpeed:-
  front_isBusy,frontDist_isSafe.
```

Finally, the third extracted rule indicates that when there is a vehicle in front of the AV and the relative distance between them is lower than the allowed distance, then the AV should brake to avoid collision with the `front` TV.

```
%% Rule h10: Brake (deceleration)
brake:-
  front_isBusy,frontVel_isLower,
  not(frontDist_isSafe).
```

As shown in Table I, we have used an *accuracy* criterion to show how rules fit on the corresponding $\mathcal{E}^+$ and $\mathcal{E}^-$. *Accuracy* is the average of *precision* and *recall*. According to the table, *accuracy* of all rules is 1,00, meaning that all rules fit the data in the best manner. Moreover, the execution time (T_e), also known as the processing time, for each rule presented in Table I. Needless to say, T_e is very short for each rule, meaning that the rule induction process is fast in the SIL framework.

C. Rule aggregation

In the previous section, we induced the desired driving rules using Popper, forming a set of hypotheses from `h1` to `h10`. However, these induced rules are raw and require processing to yield human-like actions for lane changes and velocity adjustments. Thus, after extracting the desired driving rules, we feed them into the rule aggregation component. The primary goal here is to post-process these rules to extract the best lane change and velocity actions for each specific state. Within this component, the extracted rules are integrated with provided background knowledge to construct a comprehensive decision-making program. Additionally, we introduce supplementary rules to establish strong connections between the induced rules, facilitating the extraction of the desired actions and enabling AV control with a low-level controller.

To identify the best action for state S_t, we initially employ `h1`to `h5` rules to eliminate unsafe or dangerous actions, thereby reducing the action space. Then, we determine the best action from the remaining options using `h6` and `h7`, which assess the quality of each action. Ultimately, we extract the best action a_t, which can be LK, LLC, or RLC. As illustrated in Fig. 1, once a_t is determined, the AV's lane ID adjusts accordingly using a switch box (Fig. 1), defining a pre-defined lateral position for the new lane y_d^{NL}. We then employ a low-level controller to generate a lateral acceleration command a_y.

To derive the desired velocity command (v_x^d) for the AV at each time step (see Fig. 1), we utilize the current state S_t to identify the rule among `h8`, `h9`, and `h10` rules where the head predicate holds `true`. Using the identified rule and Eq. 4, we calculate the longitudinal acceleration a_x. Afterward, we determine the desired velocity using $v_x^d = v_x + a_x \Delta t$. This desired velocity is then fed to a low-level controller as the input for the velocity controller. The controller, in turn, adjusts the thrust to obtain the desired velocity in a continuous and smooth manner. In summary, the SIL framework employs the learned hypotheses to determine high-level control commands and send them to the low-level controller component. This process yields human-like actions while adhering to safety, efficiency, and smoothness rules.

V. Results and Discussions

In this section, we discuss the results of the proposed method and compare them with a DIL baseline. HighD dataset [24] is used to indicate the implementations for both methods.

A. Baseline: DIL

We utilized a fully-connected black-box DNN as the basis for DIL, with the hyperparameters determined through preliminary tests on the data. The network was structured with three fully-connected layers: the first two layers held 128 nodes each, while the final layer contained three nodes, implementing softmax activation functions to represent the action space. The PyTorch library was employed for neural network computations, and the ADAM algorithm was utilized for the optimization process. The learning rate and state-action batch size were set to $\alpha = 1e-4$ and 256, respectively.

To provide enough data for the network, we gathered over 160,000 state-action pairs from the highD dataset by tracking all vehicles from the initial frames to the final ones. Each state included 8 normalized relative positions of the TVs around the AV, in addition to the AV's velocity divided by a predefined maximum velocity. Within each frame, we detected lane-change actions of different vehicles using their initial and final lane positions. After action detection in a given state, we assigned a binary list, consisting of 0s and 1s, as a representative for the action. Finally, the network was trained on these state-action pairs, with the loss function converging to its minimal point. It is important to note that we solely considered imitating the lane-change actions, thus avoiding the imitation of the velocity of each vehicle. Instead, we employed rule-based methods to control the AV's velocity in the DIL framework.

B. SIL Implementation

In contrast to the DIL method, we employed a small number of positive and negative examples to learn unknown rules, as displayed in Table I. Following the construction of the SIL framework, it can make interpretable decisions in each state thanks to symbolic FOL. They are also generalizable, as they are not limited to specific circumstances. Moreover, as shown in Table I, this method is fast since each rule can be induced in a fraction of a second (see T_e column). We implemented the SIL framework using the highD dataset, where a virtual AV agent with an arbitrary initial lane and velocity is considered. The objective is to drive safely, efficiently, and smoothly on the highway. It's important to note that, like DIL, the SIL training process is conducted offline and should be completed before the commencement of the driving adventure. Again, the process does not require much time to extract the rules, compared to DIL to learn a policy, as ILP programs are typically sample-efficient and can operate effectively even with just one sample.

C. Comparison

We implemented both methods in similar situations to draw an analogy between their overall performance. Both DIL and

TABLE II: Comparison of both methods in 50 test episodes

DIRECTION	LEFT-TO-RIGHT		RIGHT-TO-LEFT	
METHOD	**DIL**	**SIL**	**DIL**	**SIL**
N_{LC}	3	**40**	4	**38**
N_{hits}	6	**0**	8	**0**
$T_{avg}(sec)$	65.28	64.84	65.13	64.93
$D_{avg}(m)$	1,998	2,100	1,976	2,100
$V_{avg}(km/h)$	110.18	**116.59**	109.22	**116.43**

SIL agents start with similar locations, velocities, and directions. To compare their performance, we established certain criteria that compare safety, efficiency, and smoothness in both methods. Safety is indicated by the number of collisions (N_{hits}), while efficiency and smoothness are assessed by the number of lane changes (N_{LC}). Moreover, average velocity is used as an efficiency measure, calculated as $V_{avg} = \frac{D_{avg}}{T_{avg}}$, where D_{avg} and T_{avg} represent the average traveled distance and time per episode, respectively.

Using the highD dataset, we tested both methods in 50 episodes, each consisting of a track with a length of 2,100 meters. In each scenario, the AV drove either in the left-to-right (L2R) direction or in the right-to-left (R2L) direction. We trained the DIL agent in the L2R direction and aimed to assess its generalizability to the opposite direction. However, the proposed SIL method is inherently generalizable, as it uses general knowledge-based examples from the environment's domain to find rules. Table II presents the results of each method after completing the test scenarios. According to this table, the SIL agent successfully completed the scenarios without a single collision, while the DIL agent collided with other vehicles in 12% of the test episodes at best. Furthermore, when we employed the DIL agent in the R2L direction, the number of collisions increased, and the agent could not perform lane changes effectively. In contrast, the performance of the SIL agent did not degrade in the this direction, demonstrating better generalizability. Additionally, the number of lane changes (N_{LC}) was near zero for the DIL agent, indicating that it struggled to learn lane changes efficiently. Though the DIL agent drives very smoothly, it cannot change lane efficiently and waste time staying in a single lane, causing traffic congestion on the road. Using a large amount of data, this lack of lane change efficiency reflects the sample-inefficiency nature of DIL methods. In contrast, the proposed SIL changed the lane approximately one time per episode due to the presence of *efficiency* rules, aiding the agent in making effective lane changes and reducing driving time. To further evaluate the performance of both methods, we leveraged V_{avg}, which represents the average speed of each vehicle per episode, indicating lane change efficiency. We computed this value for each agent in both directions, and then considered the average value for comparison. Accordingly, V_{avg} of DIL and SIL methods was 109.7 and 116.51 km/h, respectively, showing that V_{avg} of the SIL agent was higher than that of the DIL agent, which implies that the SIL agent was successful at finding free lanes by changing lanes. This resulted in a freedom to increase the velocity, reducing driving time.

It's worth noting that learning each rule in the SIL framework takes a relatively short time, as shown in Table I, while training a DIL neural network over a large dataset demands significant time and computational resources. Furthermore, a large amount of data may not be available for different tasks, whereas the proposed method can learn effectively with just a few examples derived from human expertise. However, we should not overlook the drawbacks of the proposed method, one of which is that most ILP systems, including Popper, struggle with noise handling. If some positive and negative examples are mislabeled, it can be challenging for these systems to reject such noise. Moreover, to extract each unknown rule, we need to define specific settings and examples, as mentioned in subsection IV-A. Therefore, we encourage future research to address these issues and develop new methods based on this novel research line.

VI. CONCLUSION

In this paper, we proposed a novel symbolic imitation learning method which takes advantage of humans' driving background knowledge and examples to extract new rules for autonomous driving using inductive logic programming. This method consists of three main components: knowledge acquisition, rule induction, and rule aggregation, all of which are closely connected to each other. Using the highD dataset, we have shown that our method outperforms the DNN-based imitation learning method in the case of safety, efficiency, and smoothness. Moreover, the proposed method offers a completely interpretable approach using first-order logics and can perform well in unseen environments, making it generalizable as well. Furthermore, our method is more sample-efficient as compared to the DIL since we use significantly fewer state-action pairs for learning rules.

REFERENCES

[1] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. Theodorou, and B. Boots, "Agile autonomous driving using end-to-end deep imitation learning," *arXiv preprint arXiv:1709.07174*, 2017.

[2] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. A. Theodorou, and B. Boots, "Imitation learning for agile autonomous driving," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 286–302, 2020.

[3] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end autonomous driving," *arXiv preprint arXiv:1605.06450*, 2016.

[4] Y. Zhang, P. Tiňo, A. Leonardis, and K. Tang, "A survey on neural network interpretability," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 5, no. 5, pp. 726–742, 2021.

[5] J. Kim and J. Canny, "Interpretable learning for self-driving cars by visualizing causal attention," in *Proceedings of the IEEE international conference on computer vision*, pp. 2942–2950, 2017.

[6] S. K. S. Ghasemipour, R. Zemel, and S. Gu, "A divergence minimization perspective on imitation learning methods," in *Conference on Robot Learning*, pp. 1259–1277, PMLR, 2020.

[7] Z. Zhu, K. Lin, B. Dai, and J. Zhou, "Off-policy imitation learning from observations," *Advances in Neural Information Processing Systems*, vol. 33, pp. 12402–12413, 2020.

[8] B. Fang, S. Jia, D. Guo, M. Xu, S. Wen, and F. Sun, "Survey of imitation learning for robotic manipulation," *International Journal of Intelligent Robotics and Applications*, vol. 3, pp. 362–369, 2019.

[9] C. Yu, X. Zheng, H. H. Zhuo, H. Wan, and W. Luo, "Reinforcement learning with knowledge representation and reasoning: A brief survey," *arXiv preprint arXiv:2304.12090*, 2023.

[10] F. K. Došilović, M. Brčić, and N. Hlupić, "Explainable artificial intelligence: A survey," in *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)*, pp. 0210–0215, IEEE, 2018.

[11] E. Tjoa and C. Guan, "A survey on explainable artificial intelligence (xai): Toward medical xai," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 11, pp. 4793–4813, 2020.

[12] M. K. Sarker, L. Zhou, A. Eberhart, and P. Hitzler, "Neuro-symbolic artificial intelligence," *AI Communications*, vol. 34, no. 3, pp. 197–209, 2021.

[13] P. Hitzler and M. K. Sarker, "Neuro-symbolic artificial intelligence: The state of the art," 2022.

[14] D. Kimura, M. Ono, S. Chaudhury, R. Kohita, A. Wachi, D. J. Agravante, M. Tatsubori, A. Munawar, and A. Gray, "Neuro-symbolic reinforcement learning with first-order logic," *arXiv preprint arXiv:2110.10963*, 2021.

[15] M. Zimmer, X. Feng, C. Glanois, Z. Jiang, J. Zhang, P. Weng, L. Dong, H. Jianye, and L. Wulong, "Differentiable logic machines," *arXiv preprint arXiv:2102.11529*, 2021.

[16] A. Cropper and S. Dumančić, "Inductive logic programming at 30: a new introduction," *Journal of Artificial Intelligence Research*, vol. 74, pp. 765–850, 2022.

[17] Z. Song, Y. Jiang, J. Zhang, P. Weng, D. Li, W. Liu, and J. Hao, "An interpretable deep reinforcement learning approach to autonomous driving," in *IJCAI Workshop on Artificial Intelligence for Automous Driving*, 2022.

[18] A. Cropper, S. Dumančić, and S. H. Muggleton, "Turning 30: New ideas in inductive logic programming," *arXiv preprint arXiv:2002.11002*, 2020.

[19] V. Lifschitz, *Answer set programming*. Springer Heidelberg, 2019.

[20] D. Corapi, A. Russo, and E. Lupu, "Inductive logic programming in answer set programming," in *International conference on inductive logic programming*, pp. 91–97, Springer, 2011.

[21] A. Cropper and R. Morel, "Learning programs by learning from failures," *Machine Learning*, vol. 110, pp. 801–856, 2021.

[22] I. Sharifi, M. Yildirim, and S. Fallah, "Towards safe autonomous driving policies using a neuro-symbolic deep reinforcement learning approach," *arXiv preprint arXiv:2307.01316*, 2023.

[23] J. Sun, H. Sun, T. Han, and B. Zhou, "Neuro-symbolic program search for autonomous driving decision module design," in *Conference on Robot Learning*, pp. 21–30, PMLR, 2021.

[24] R. Krajewski, J. Bock, L. Kloeker, and L. Eckstein, "The highd dataset: A drone dataset of naturalistic vehicle trajectories on german highways for validation of highly automated driving systems," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2118–2125, IEEE, 2018.