Toward Interpretable-AI Policies Using Evolutionary Nonlinear Decision Trees for Discrete-Action Systems

Yashesh Dhebar, Kalyanmoy Deb, Fellow, IEEE, Subramanya Nageshrao, Ling Zhu, and Dimitar Filev, Fellow, IEEE

Abstract—Black-box artificial intelligence (AI) induction methods such as deep reinforcement learning (DRL) are increasingly being used to find optimal policies for a given control task. Although policies represented using a black-box AI are capable of efficiently executing the underlying control task and achieving optimal closed-loop performance—controlling the agent from the initial time step until the successful termination of an episode, the developed control rules are often complex and neither interpretable nor explainable. In this article, we use a recently proposed nonlinear decision-tree (NLDT) approach to find a hierarchical set of control rules in an attempt to maximize the open-loop performance for approximating and explaining the pretrained black-box DRL (oracle) agent using the labeled state–action dataset. Recent advances in nonlinear optimization approaches using evolutionary computation facilitate finding a hierarchical set of nonlinear control rules as a function of state variables using a computationally fast bilevel optimization procedure at each node of the proposed NLDT. In addition, we propose a reoptimization procedure for enhancing the closed-loop performance of an already derived NLDT. We evaluate our proposed methodologies (open- and closed-loop NLDTs) on different control problems having multiple discrete actions. In all these problems, our proposed approach is able to find relatively simple and interpretable rules involving one to four nonlinear terms per rule, while simultaneously achieving on par closed-loop performance when compared to a trained black-box DRL agent. A postprocessing approach for simplifying the NLDT is also suggested. The obtained results are inspiring as they suggest the replacement of complicated black-box DRL policies involving thousands of parameters (making them noninterpretable) with relatively simple interpretable policies. The results are encouraging and motivating to pursue further applications of proposed approach in solving more complex control tasks.

Index Terms—Bilevel, interpretable, nonlinear decision tree (NLDT), reinforcement learning (RL).

I. INTRODUCTION

CONTROL system problems are increasingly being solved by using modern reinforcement learning (RL) and other machine learning (ML) methods to find an autonomous agent (or controller) to provide an optimal action $A_t$ for every state variable combination $S_t$ in a given environment at every time step $t$. The execution of the output action $A_t$ takes the object to the next state $S_{t+1}$ in the environment. This process is repeated until a termination criteria is met. The mapping between input state $S_t$ and output action $A_t$ in most real-world problems is usually complex, and is, therefore, captured through an artificial intelligence (AI) method. In the RL literature, this mapping is referred to as policy, and sufficient literature exists in efficient training of these RL policies [1]–[7]. While these methods are efficient at training the AI policies for a given control system task, the developed AI policies, often represented using complicated artificial neural networks, are usually complex and noninterpretable.

Interpretability of AI policies is important to a human mind due to several reasons: 1) interpretable policies provide better insight and understanding of the working principles of the derived control-logic; 2) they can be easily deployed with a low fidelity hardware; and 3) they may also allow an easier way to extend the control policies for more complex versions of the problem. While defining interpretability is a subjective matter, a number of past efforts have attempted to find interpretable AI policies with limited success [8], [9]. In this work, our aim is to find policies that are relatively interpretable and simple as compared to the black-box AI counterparts such as DNN. For this purpose, we propose to use a nonlinear rule with limited complexity at each node of the nonlinear decision tree (NLDT). At each time step, the state variable vector is passed through the derived NLDT’s root rule and the control flows through the decision tree (DT) to arrive at a specific leaf node, describing one of the allowable discrete actions. Well-known difficulties of learning nonlinear rules are alleviated by using a bilevel optimization approach in which the upper level handles the structure of a rule and the lower level determines the coefficients of the terms of the rule. Moreover,
a final efficient search of all coefficients of rules in the NLDT is employed to arrive at a reliable rule base.

In the remainder of this article, we first present the main motivation behind finding interpretable policies in Section II. In Section III, a brief overview on existing works related to interpretable AI policies is provided. In Section IV, we review a recently proposed NLDT approach in the context of arriving at relatively interpretable AI policies. The overall open-loop and closed-loop NLDT policy generation methods are described in Section V. The results on four control system problems are presented in Section VI. Next, a new benchmark problem is proposed to conduct a scale-up study of our algorithm in Section VII. Finally, conclusions and future studies are presented in Section VIII. The supplementary material provides further details.

II. Motivation for the Study

Various data analysis tasks, such as classification, controller design, regression, image processing, etc., are increasingly being solved using AI methods. The suggested methods have been demonstrated to solve complex data analysis tasks without much change in their usual frameworks. However with more such studies over the past few decades [9], these AI generation methods are faced with a huge challenge — interpreting and understanding the inner working of AI. Achieving a high-accuracy solution does not necessarily satisfy a curious domain expert, particularly if the solution is not interpretable or explainable. A technique (whether AI-based or otherwise) to handle data well is no more enough. The researchers now or explainable. A technique (whether AI-based or otherwise) needs to be quite complex in order to have its policy function \( \pi \), one way to achieve this would be through an interpretable policy function \( \pi_{\text{int}}(S) \) as follows:

\[
\pi_{\text{int}}(S) = \begin{cases} 
0, & \text{if } \phi_0(S) \text{ is true} \\
1, & \text{if } \phi_1(S) \text{ is true} \\
2, & \text{if } \phi_2(S) \text{ is true}
\end{cases}
\] (1)

where \( \phi_i(S) : R^2 \rightarrow \{0, 1\} \) is a Boolean function that partitions the state space \( S \) into two subdomains based on its output value. For a given state \( S \), exactly one of \( \phi_i(S) \) is true, thereby making the policy \( \pi_{\text{int}} \) deterministic. If we relook at Fig. 1(a), we notice that the three actions are quite mixed at the bottom part of the \( x-v \) plot (state space). Thus, the partitioning Boolean functions \( \phi_i \) need to be quite complex in order to have \( \phi_0(S) = \text{true} \) for all blue points, \( \phi_1(S) = \text{true} \) for all orange points, and \( \phi_2(S) = \text{true} \) for all green points.

What we address in this study is an attempt to find an approximated policy function \( \pi_{\text{int}}(S) \), which may not replicate and explain all 100% time instance data corresponding to the oracle black-box policy \( \pi_{\text{oracle}}(S) \) [Fig. 1(a)], but it is fairly interpretable to explain close to 100% data. Consider the state–action plot in Fig. 1(b), which is generated with a relatively simple and relatively interpretable policy \( \pi_{\text{int}}(S) = \{i | \phi_i(S) \text{ is true}, i = 1, 2, 3\} \) obtained by our proposed procedure shown as follows:

\[
\phi_0(S) = \neg \psi_1(S) \\
\phi_1(S) = (\psi_1(S) \land \neg \psi_2(S)) \\
\phi_2(S) = (\psi_1(S) \land \psi_2(S))
\] (2)

where \( \psi_1(S) = |0.96 - 0.63\hat{x}^2 - 0.28\hat{v}_i - 0.22\hat{v}_f| \leq 0.36 \) and \( \psi_2(S) = |1.39 - 0.28\hat{x}^2 - 0.30\hat{v}_f^2| \leq 0.53 \) represent black and orange boundaries, respectively. Here, \( \hat{x} \) and \( \hat{v} \) are normalized state variables (see the supplementary material for details). The action \( A_t \) predicted using the above policy does not match the output of \( \pi_{\text{oracle}} \) at some states (about 8.1%), but from our experiments we observe that it is still able to drive the mountain-car to the destination goal located on the right hill in 99.8% episodes.

Importantly, the \( \pi_{\text{int}} \) policy is relatively simpler (involving two simple nonlinear functions \( \psi_1 \) and \( \psi_2 \)) than the corresponding black-box policy \( \pi_{\text{oracle}} \) and is amenable to an easier understanding of the relationships between \( x_t \) and \( v_t \) to make a near perfect control. Since the explanation process used the data from \( \pi_{\text{oracle}} \) as the universal truth [Fig. 1(a)], the derived relationships will also provide an explanation of the working of the black-box policy \( \pi_{\text{oracle}} \). A more gross approximation to Fig. 1(a) by more simplified relationships (\( \phi_i \)) may reduce the overall open-loop accuracy of matching the output.
of $\pi_{\text{oracle}}$. Hence, a balance between a good interpretability and a high open-loop accuracy in searching for Boolean functions $\phi_i(S_i)$ becomes an important matter for such an interpretable AI-policy development study.

In this article, we focus on developing a search procedure for arriving at the $\psi$-functions [see (2)] and their combinations for different discrete action systems. The structure of the policy $\pi_{\text{int}}(s_i)$ shown in (1) resembles a DT. However, unlike a standard DT, it involves a nonlinear function at every nonleaf node. This requires an efficient nonlinear optimization method to arrive at reasonably succinct and accurate nonlinear functionals. The procedure we propose here is generic and is independent of the AI method used to develop the black-box policy $\pi_{\text{oracle}}$.

III. RELATED PAST STUDIES

In [15], an interpretable orchestrator is developed to choose from two RL-policies: 1) $\pi_C$ for maximizing reward and 2) $\pi_R$ for maximizing an ethical consideration. The orchestrator is dependent on only one of the state variables and despite it being interpretable, the policies: $\pi_C$ and $\pi_R$, are still black-box and convoluted. Maes et al. [16] constructed a set of interpretable index-based policies and used a multiarm bandit procedure to select a high performing index-based policy. The search space of interpretable policies is much smaller and the procedure suggested for finding an interpretable policy is computationally heavy, taking about hours to several days of computational time on simple control problems. In [17], genetic programming (GP) is used to obtain interpretable policies on control tasks involving continuous actions space through model-based policy learning. However the interpretability was not captured in the design of the fitness function and a large archive was created passively to store every policy for each complexity encountered during the evolutionary search. A linear DT-based model is used in [18] to approximate the $Q$-values of the trained neural network. In that work, the split in DT occurs based on only one feature, and at each terminal node, the $Q$-function is fitted using a linear model on all features. Verma et al. [19] used a program sketch $S$ to define the domain of interpretable policies $e$. Interpretable policies are found using a trained black-box oracle $e_N$ as a reference by first conducting a local search in the sketch space $S$ to mimic the behavior of the oracle $e_N$ and then fine-tuning the policy parameters through online Bayesian optimization. The bias toward generating interpretable programs is done through controlled initialization and local search rather than explicitly capturing interpretability as one of the fitness measure. Particle swarm optimization [20] is used to generate the interpretable fuzzy rule set in [21] and is demonstrated on classic control problems involving continuous actions. Works on DT [22]-based policies through imitation learning have been carried out in [23]. Bastani et al. [24] extended this to utilize $Q$-values and eventually render DT policies involving < 1000 nodes on some toy games and CartPole environment with an ultimate aim to have the induced policies verifiable. Bastani et al. [25] used axis-aligned DTs to develop interpretable models for black-box classifiers and RL-policies. They first derive a distribution function $P$ by fitting the training data through axis-aligned Gaussian distributions. $P$ is then used to compute the loss function for splitting the data in the DT. Vandewiele et al. [26] attempted to generate interpretable DTs from an ensemble using a genetic algorithm. In [27], regression trees are derived using classical methods, such as CART [22] and Kd-tree [28], to model $Q$-function through supervised training on batch of experiences and comparative study is made with ensemble techniques. In [29], a gradient-based approach is developed to train the DT of pre-fixed topology involving linear split rules. These rules are later simplified to allow only one feature per split node and resulting DTs are pruned to generate simplified rule set.

While the above methods attempt to generate an interpretable policy, the search process does not use complexity of policy in the objective function, instead, they rely on initializing the search with certain interpretable policies. The studies on DT-based interpretable policies so far mostly involve either axis-parallel trees (where split rules are of type: $x_i \leq \tau$) or oblique trees (where split rules are linear functions of state variables). In some approaches, the topology of DT is fixed. Our approach is more flexible in deriving an interpretable policy, since it grows the hierarchical set of interpretable rules naturally (i.e., topology of DT is not fixed or prespecified) and the inherent split rules of DT can assume a nonlinear structure. The problem of finding nonlinear rule set to solve a given control task is a challenging problem involving efficient nonlinear optimization algorithms (and one of the key reasons why previous approaches probably did not address it). The complex task of arriving at nonlinear set of control logic and their hierarchical connection is the focus of this article. In our approach described below, we build an efficient search algorithm to directly find relatively interpretable policies as compared to the black-box policies represented using DNN (or tile encoding [12]) using recent advances in nonlinear optimization.

IV. NONLINEAR DECISION TREE APPROACH

In this study, we use a direct mathematical rule generation approach [presented in (2)] based on NLDT [30]. While the overall goal of this work is to induce an interpretable policy $\pi_{\text{int}}$ that can serve as a controller, the first step (discussed at length in Section V) involves the induction of $\pi_{\text{int}}$ to approximate and explain the behavior of a pretrained black-box policy $\pi_{\text{oracle}}$. This can be achieved by deriving $\pi_{\text{int}}$ to match the labeled state–action data that is generated using $\pi_{\text{oracle}}$.

DTs are considered a popular choice due to their interpretability aspects. They are intuitive and each decision can be easily interpreted. However, in a general scenario, regular DTs often have complicated topology since the rules at each conditional node can assume only axis parallel structure $x_i \leq \tau$ to make a split. On the other end, single rule-based classifiers such as support vector machines (SVMs) have just one rule, which is complicated and highly nonlinear. Keeping these two extremes in mind, we develop an NLDT framework where each conditional node can assume a nonlinear functional form while the tree is allowed to grow by recursively splitting the data in conditional nodes, similar to the procedure.
used to induce regular DTs. In our case of replicating a policy \( \pi_{\text{oracle}} \), a conditional node in NLDT represents a nonlinear control logic and terminal leaf nodes indicate an action. This is schematically shown in Fig. 2. An NLDT, which can be viewed as an assembly of conditional nodes, represents an overall controller/policy \( \pi_{\text{int}} \).

In the binary-split NLDT, used in this study, a conditional node is allowed to have exactly two splits as shown in Fig. 2. The nonlinear split rule \( f(x) \) at each conditional node is expressed as a weighted sum of powerlaws

\[
 f(x) = \begin{cases} 
 \sum_{i=1}^{m} w_i B_i + \theta_1, & \text{if } m = 0 \\
 \sum_{i=1}^{m} w_i B_i + \theta_1 - |\theta_2|, & \text{if } m = 1 
\end{cases}
\]

where power laws \( B_i \) are given as \( B_i = \prod_{j=1}^{d} x_j^{b_{ij}} \) and \( m \) indicates if an absolute operator should be present in the rule or not. In Section V-A, we discuss procedures to derive values of exponents \( b_{ij} \), weights \( w_i \), and biases \( \theta_1 \).

V. OVERALL APPROACH

The overall approach is illustrated in Fig. 3. First, a dedicated black-box policy \( \pi_{\text{oracle}} \) is trained from the actual environment/physics of the problem. This aspect is not the focus of this article. Next, the trained policy \( \pi_{\text{oracle}} \) (Block 1 in the figure) is used to generate labeled training and testing datasets of state–action pairs from different time steps. We generate two types of training datasets: 1) Regular—as they are recorded from multiple episodes,\(^2\) and 2) Balanced—selected from multiple episodes to have almost equal number of states for each action, where an episode is a complete simulation run wherein an AI policy is used to control the object over multiple time steps. Third, the labeled training dataset (Block 2) is used to find the NLDT (Block 3) using a recursive bilevel evolutionary algorithm described in Section V-A. We call this an open-loop NLDT (or, NLDT\(_{OL} \)) since it is derived from a labeled state–action dataset generated from \( \pi_{\text{oracle}} \) and does not involve any direct interaction with the control environment. Use of a labeled state–action data in a supervised manner allows for a faster search of NLDT even with a large dataset as compared to constructing the NLDT using traditional RL, which involves interaction with the environment [19]. Next, in an effort to make the overall NLDT interpretable while simultaneously ensuring better closed-loop performance, we prune the NLDT by taking only the top part of NLDT\(_{OL} \) [we call NLDT\(_{OL}^{(p)} \) in Block 4] and reoptimize all nonlinear rules within it for the weights and biases using an efficient evolutionary optimization procedure to obtain final NLDT\(_* \) (Block 5). The reoptimization is done here with closed-loop objectives, such as the cumulative reward function or closed-loop completion rate. We briefly discuss the open-loop training procedure of inducing NLDT\(_{OL} \) and the closed-loop training procedure to generate NLDT\(_* \) in next sections.

A. OPEN-LOOP TRAINING

A labeled state–action dataset is first created using a pretrained black-box policy \( \pi_{\text{oracle}} \). Since we are dealing with discrete-action control problems, the underlying imitation task of replicating the behavior of \( \pi_{\text{oracle}} \) using the labeled state–action data translates to a classification problem. We train NLDT discussed in Section IV to fit the state–action data through supervised learning. Nonlinear split rule \( f(x) \) at each conditional node [Fig. 2 and (3)] is derived using a dedicated bilevel optimization algorithm [31], where the upper level searches the template of the nonlinear rule and the corresponding lower level focuses at estimating optimal values of weights/coefficients for optimal split of data present in the conditional node. The optimization formulation for deriving a nonlinear split rule \( f(x) (3) \) at a given conditional node is given as follows:

\[
\text{Minimize } F_{U}(B, m, w^*, \theta^*) \\
\text{subject to } (w^*, \theta^*) \in \text{argmin} \left\{ F_L(w, \theta) |_{(B, m)} \right\}
\]

\[
\begin{align*}
F_L(w, \theta) |_{(B, m)} &\leq \tau_l \\
-1 &\leq w_i \leq 1 \quad \forall l \\
\theta &\in [-1, 1]^{m+1}
\end{align*}
\]

where \( m \in \{0,1\} \), \( b_{ij} \in Z \)

where \( Z \) is a set of exponents allowed to limit the complexity of the derived rule structure. In this study, we use \( Z = \{-3, -2, -1, 0, 1, 2, 3\} \). The objective \( F_U \) quantifies the complexity of the nonlinear rule by enumerating the number of terms present in the equation of rule \( f(x) \) shown as follows:

\[
F_U(B, m, w^*, \theta^*) = \sum_{i=1}^{p} \sum_{j=1}^{d} g(b_{ij})
\]

where \( g(a) = 1 \), if \( a \neq 0 \), zero otherwise. \( m \) indicates the presence or absence of a modulus operator and \( w \) and \( \theta \) encode rule weights \( w_i \) and biases \( \theta_1 \), respectively. The lower level objective function \( F_L \) quantifies the net impurity of child nodes resulting from the split. Impurity \( I \) of a node \( P \) is computed using a Gini-score: \( \text{Gini}(P) = 1 - \sum_{i=1}^{c} (N_i/N)^2 \), where \( N \) is the total number of points present in the node and \( N_i \) represents number of points belonging to class \( i \). Datapoints present in node \( P \) gets distributed into two nonoverlapping subsets based on their split function value. Datapoints with \( f(x) \leq 0 \) go to the left child node \( L \) and rest go to the right child node \( R \). The lower level objective function \( F_L \), which quantifies the quality of this split, is then given by

\[
F_L(w, \theta) |_{(B, m)} = \left( \frac{N_L}{N_P} \text{Gini}(L) + \frac{N_R}{N_P} \text{Gini}(R) \right)_w \theta (B, m)
\]
The $\tau_f$ parameter in (4) represents the maximum allowable net-impurity (6) of child nodes. The resulting child nodes obtained after the split undergo another split and the process continues until one of the termination criteria is met.

More details regarding the bilevel-optimization algorithm [31] to derive split rule $f_i(x)$ at the $i$th conditional node in NLDT can be found in [30].

After the entire NLDT is found, in this study, a pruning and tree simplification strategy (see the supplementary material for more details) is applied to reduce the size of NLDT in an effort to improve on the interpretability of the overall rule set. This entire process of inducing NLDT from the labeled state–action data results into the open-loop NLDT—NLDT$^{OL}$. NLDT$^{OL}$ can then be used to explain the behavior of the oracle $\pi_{oracle}$. We will see in Section VI that despite being not 100% accurate in imitating $\pi_{oracle}$, NLDT$^{OL}$ manages to achieve respectable closed-loop performance with 100% completion rate and a high cumulative reward value. Next, we discuss the closed-loop training procedure to obtain NLDT$^*$.

B. Closed-Loop Training

The intention behind closed-loop training is to enhance the closed-loop performance of NLDT. It will be discussed in Section VI that while closed-loop performance of NLDT$^{OL}$ is at par with $\pi_{oracle}$ on control tasks involving two to three discrete actions, such as CartPole and MountainCar, the NLDT$^{OL}$ struggles to autonomously control the agent for control problems involving more states and actions (e.g., LunarLander). In closed-loop training, we fine-tune and reoptimize the weights $W$ and biases $\Theta$ of an entire NLDT$^{OL}$ (or pruned NLDT$^{OL}$—block 4 in Fig. 3) to maximize its closed-loop fitness ($F_{CL}$). The closed-loop fitness ($F_{CL}$) is expressed as the average of the cumulative reward collected across $M$ episodes

$$\text{Maximize } F_{CL}(W, \Theta) = \frac{1}{M} \sum_{i=1}^{M} R_i(W, \Theta)$$

Subject to $W \in [-1, 1]^{n_w}, \Theta \in [-1, 1]^{n_\theta}$

(7)

where $n_w$ and $n_\theta$ are the total number of weights and biases appearing in entire NLDT. In our case, we set $M = 20$.

VI. RESULTS

In this section, we present results obtained using our approach on following control problems: 1) CartPole; 2) CarFollowing; 3) MountainCar; and 4) LunarLander. The first two problems have two discrete actions, third problem has three discrete actions, and the fourth problem has four discrete actions. Open-loop statistics are reported using scores of training and testing accuracy on labeled state-action data generated from $\pi_{oracle}$. For quantifying the closed-loop performance, we use two metrics: 1) completion rate, which gives a measure on the number of episodes which are successfully completed and 2) cumulative reward, which quantifies how well an episode is executed. For each problem, ten runs of open-loop training are executed using 10 000 training datapoints. Open-loop statistics obtained from these runs are reported wherein the testing dataset also comprises of 10 000 datapoints. We choose the median performing NLDT$^{OL}$ for closed-loop analysis. We run 50 batches of 100 episodes each and report statistics regarding completion rate and cumulative reward for NLDT$^*$ obtained after closed-loop training of median performing NLDT$^{OL}$.

A. CartPole Problem

This problem comprises of four state variables and two discrete actions. Details regarding this problem are provided in the supplementary document. The oracle DNN controller is trained using the PPO algorithm [2]. Table I shows the performance of NLDT on training datasets of different sizes. It is observed that NLDT trained with 5000 and 10 000 datapoints shows a robust open-loop performance and also produces 100% closed-loop performance. Based on this observation, we set the default value of training data to 10 000 for all control problems discussed in this article. The obtained NLDT$^{OL}$ has about two rules with an average of about four terms in the derived nodal functions.

Interestingly, the same NLDT (without closed-loop training) also produces 100% closed-loop performance by achieving the maximum cumulative reward value of 200.

1) NLDT for CartPole Problem: One of the NLDT$^{OL}$s obtained for the CartPole problem is shown in Fig. 4 in terms of normalized state variable vector $\tilde{x}$. 

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
\[ x^{\text{max}} = [1.37, 0.88, 0.10, 0.45] \text{CartPole Rules.} \]

Normalization Constants Are: \( x^{\text{mix}} = [-0.91, -0.43, -0.05, -0.40] \), \( x^{\text{max}} = [1.37, 0.88, 0.10, 0.45] \) if

\[
\begin{align*}
&| -0.18 \hat{x}_0 \hat{s}_2^{-2} - 0.63 \hat{x}_3^{-2} + 0.67 | - 0.24 \leq 0 \text{ then} \\
&\quad \text{Action} = 0 \\
&\text{else} \quad \text{Action} = 1
\end{align*}
\]

The respective policy can be alternatively stated using a programmable if-then-else rule-structure as shown in Algorithm 1.

A little manipulation will reveal that for a correct control strategy, Action 0 must be invoked if the following condition is true:

\[
2.39 \leq \left( \frac{\hat{x}_0}{\hat{x}_2^2} + \frac{3.50}{\hat{x}_3^2} \right) \leq 5.06
\]

otherwise, Action 1 must be invoked. First, notice that the above policy does not require the current velocity (\( \hat{s}_1 \)) to determine the left or right action movement. Second, for small values of angular position (\( \hat{s}_2 \approx 1 \)) and angular velocity (\( \hat{s}_3 \approx 1 \)), that is, the pole is falling toward left, the above condition is always true. That is, the cart should be pushed toward left, thereby trying to stabilize the pole to vertical position. On the other hand, if the pole is falling toward right (large values of \( \hat{s}_2 \approx 2 \) and \( \hat{s}_3 \approx 2 \)), the term in bracket will be smaller than 2.39 for all \( \hat{x}_0 \in [1, 2] \), and the above policy suggests that Action 1 (push the cart toward right) must be invoked. When the pole is falling right, a push of the cart toward right helps to stabilize the pole toward its vertical position. These extreme case analyses are intuitive and our policy can be explained for its proper working, but what our NLDT approach is able to find is a precise rule for all situations of the state variables to control the CartPole to a stable configuration, mainly using the black-box-AI data.

### Algorithm 1

\[
\begin{align*}
&x^{\text{max}} = [1.37, 0.88, 0.10, 0.45] \text{CartPole Rules.} \\
&\text{Normalization Constants Are: } x^{\text{mix}} = [-0.91, -0.43, -0.05, -0.40], \ x^{\text{max}} = [1.37, 0.88, 0.10, 0.45] \text{ if} \\
&| -0.18 \hat{x}_0 \hat{s}_2^{-2} - 0.63 \hat{x}_3^{-2} + 0.67 | - 0.24 \leq 0 \text{ then} \\
&\quad \text{Action} = 0 \\
&\text{else} \quad \text{Action} = 1
\end{align*}
\]

The results of NLDT’s performance on problems with two discrete actions (Tables I–III) indicate that despite having a noticeable mismatch with the open-loop output of the oracle black-box policy \( \pi_{\text{oracle}} \), the closed-loop performance of NLDT is at par or at times better than \( \pi_{\text{oracle}} \). This observation suggests that certain state–action pairs are not of crucial importance when it comes to executing a closed-loop control and, therefore, errors made in predicting these state–action events do not affect and deteriorate the closed-loop performance.

### Table I

| Training Data Size | Training Accuracy | Test Accuracy (Open-loop) | # Rules | Rule Length | Cumulative Reward, Mat=200 | Compl. Rate |
|--------------------|-------------------|--------------------------|---------|------------|-----------------------------|------------|
| 100                | 97.00 ± .55       | 82.79 ± 2.40             | 1.50 ± 0.50 | 3.30 ± 0.93 | 199.73 ± 0.32 | 95.00 ± 5.10 |
| 500                | 95.54 ± 1.53      | 79.66 ± 3.10             | 1.90 ± 0.54 | 3.88 ± 0.60 | 175.38 ± 2.81 | 51.00 ± 5.10 |
| 1,000              | 91.90 ± 0.87      | 90.59 ± 1.87             | 1.80 ± 0.40 | 4.05 ± 1.04 | 200.00 ± 0.00 | 100 ± 0.00   |
| 5,000              | 92.07 ± 1.28      | 92.02 ± 1.27             | 1.70 ± 0.46 | 4.25 ± 0.90 | 200.00 ± 0.00 | 100 ± 0.00   |
| 10,000             | 91.86 ± 1.25      | 92.05 ± 1.10             | 1.30 ± 0.46 | 4.45 ± 1.56 | 200.00 ± 0.00 | 100 ± 0.00   |

### Table II

| Train. Acc. | Test. Acc. | Depth | # Rules | Rule Length | Compl. Rate |
|-------------|------------|-------|---------|-------------|-------------|
| 96.41 ± 1.97 | 96.53 ± 1.90 | 1.90 ± 0.30 | 2.40 ± 0.66 | 3.28 ± 0.65 | 100 ± 0.00 |

### Table III

| AI | Cumulative Reward | Compl. Rate |
|----|-------------------|-------------|
| DNN | 174.16k 173.75k ±20.95 | 100 ± 0.00 |
| NLDT<sub>OL</sub> | 174.15k 173.87k ±16.48 | 100 ± 0.00 |
| NLDT<sup>*</sup> | 179.76k 179.71k ±0.95 | 100 ± 0.00 |

### B. CarFollowing Problem

We have developed a discretized version of the car following problem discussed in [32]. The objective here is to control the rear car and follow a randomly moving car using acceleration or braking actions. This problem involves three state variables and two discrete actions. More details regarding this problem are provided in the Supplementary document. The oracle policy was obtained using a double Q-learning algorithm [33]. The reward function for the CarFollowing problem is shown in the supplementary material, indicating that a relative distance close to 30 m produces the highest reward.

The results for the CarFollowing problem are shown in Table II. An average open-loop accuracy of 96.53% is achieved with at most three rules, each having 3.28 terms on an average.

For this problem, we apply the closed-loop reoptimization (Blocks 4 and 5 to produce Block 6 in Fig. 3) on the entire NLDT<sub>OL</sub>. As shown Table III, NLDT<sup>*</sup> is able to achieve better closed-loop performances (100% completion rate and slightly better average cumulative reward). Fig. 5 shows that NLDT<sup>*</sup> adheres to the 30-m gap between the cars more closely than original DNN or NLDT<sub>OL</sub>.

The NLDT<sub>OL</sub> obtained for the CarFollowing problem is shown in Fig. 6 and its physical interpretation is provided in the supplementary material.

The results of NLDT’s performance on problems with two discrete actions (Tables I–III) indicate that despite having a noticeable mismatch with the open-loop output of the oracle black-box policy \( \pi_{\text{oracle}} \), the closed-loop performance of NLDT is at par or at times better than \( \pi_{\text{oracle}} \). This observation suggests that certain state–action pairs are not of crucial importance when it comes to executing a closed-loop control and, therefore, errors made in predicting these state–action events do not affect and deteriorate the closed-loop performance.
Fig. 5. Relative distance plot for CarFollowing.

0.63\xi_0^2 - 0.87\xi_1^2\xi_2^2 - 1.00 \quad \text{Node 0}

0.96\xi_1^3 - 0.58\xi_0 + 1.00 \quad \text{Node 1}

\leq 0 \quad \text{Node 2 Action 1} \quad \text{Node 3 Action 0}

\leq 0 \quad \text{Node 4 Action 1}

Fig. 6. NLDT \text{OL} for the CarFollowing problem. Normalization constants are: \( x^{\min} = [-1.25, -7.93, -1.00] \) and \( x^{\max} = [30.30, 0.70, 1.00] \).

C. MountainCar Problem

This problem comprises of two state variables to capture \( x \) position and velocity of the car. The task is to use three actions and drive an underpowered car to the destination (see the supplementary document for more details).

Compilation of results of the NLDT_{OL} induced using training datasets comprising of different data distributions (\textit{regular} and \textit{balanced}) is presented in Table IV. A state–action plot obtained using \( \pi_{\text{oracle}} \) and one of the NLDT policy corresponding to the first row of Table IV is provided in Fig. 1(a) and (b), respectively. It is observed that about 8% mismatch in the open-loop performance (i.e., testing accuracy in Table IV) comes from the lower left region of state–action plot [Fig. 1(a) and (b)] due to highly nonlinear nature of \( \pi_{\text{oracle}} \). Despite having this mismatch, the NLDT policy is able to achieve close to 100% closed-loop control performance with an average of 2.4 rules having about three terms.

Also, NLDT trained on balanced dataset (2nd row of Table IV) is able to achieve 100% closed-loop performance and involves about three control rules with an average of 3.07 terms in each rule.

D. LunarLander Problem

The task in this problem is to control the lunar-lander using four discrete actions and successfully land it on a lunar terrain. The state of the lunar lander is expressed with eight state variables, six of which are continuous, and two are categorical. More details of this problem are provided in the supplementary material.

Table V provides the compilation of results obtained using NLDT_{OL}. In this problem, while a better open-loop performance occurs for regular dataset, a better closed-loop performance is observed when the NLDT open-loop training is done on the balanced dataset. Also, NLDT_{OLS} with depth three are not adequate to achieve high closed-loop performance. The best performance is observed using balanced dataset where NLDT_{OL} of depth 6 achieves 93% episode completion rate. A specific NLDT_{OL} of depth 6 with 26 rules, each having about 4.15 terms, is shown in the supplementary material.

It is understandable that a complex control task involving many state variables cannot be simplified or made interpretable with just one or two control rules. Next, we use a part of the NLDT_{OL} from the root node to obtain a pruned NLDT^{(P)} (step “B” in Fig. 3) and reoptimize all weights (\( W \)) and biases (\( \Theta \)) using the procedure discussed in Section V-B (shown by orange box in Fig. 3) to find closed-loop NLDT^*. Table VI shows that for the pruned NLDT-3, which comprises of the top three layers and involves only four rules of original 26-rule NLDT_{OL} (i.e., NLDT-6), the closed-loop performance increases from 51% to 96% (NLDT^*-3 results in Table VI) after reoptimizing its weights and biases with closed-loop
### TABLE VI

**Closed-Loop Performance on LunarLander Problem With and Without Reoptimization on 26-Rule NLDT_{OL}. Number of Rules Is Specified in Brackets for Each NLDT and Total Parameters for the DNN Are Marked.**

| Re-Opt. | NLDT-2 (2)       | NLDT-3 (4)       | NLDT-4 (7)       | NLDT-5 (13)      | NLDT-6 (26)      | DNN (4,996)     |
|---------|------------------|------------------|------------------|------------------|------------------|-----------------|
|         | Cumulative Reward|                  |                  |                  |                  |                 |
| Before  | $-1675.77 \pm 164.29$ | $42.96 \pm 13.83$ | $54.24 \pm 27.44$ | $56.16 \pm 23.50$ | $169.43 \pm 23.96$ | $247.27 \pm 3.90$ |
| After   | $-133.95 \pm 2.51$   | $234.12 \pm 17.95$ | $234.98 \pm 22.25$ | $182.87 \pm 21.92$ | $214.94 \pm 17.31$ | $93.00 \pm 3.30$  |
|         | Completion Rate    |                  |                  |                  |                  |                 |
| Before  | $0.00 \pm 0.00$     | $51.00 \pm 3.26$  | $82.00 \pm 9.80$  | $79.00 \pm 7.66$  | $93.00 \pm 3.30$  | $94.00 \pm 1.96$  |
| After   | $18.00 \pm 7.38$    | $90.00 \pm 2.77$  | $99.00 \pm 1.71$  | $93.00 \pm 7.59$  | $91.00 \pm 4.15$  |                 |

**Fig. 8.** Final NLDT*-3 for LunarLander prob. $\hat{x}_i$ is a normalized state variable (see the supplementary material).

**Fig. 9.** Closed-loop training plot for finetuning the rule-set corresponding to depth-3 NLDT\(_{OL}\) to obtain NLDT\_* for LunarLander problem.

**Fig. 10.** PSM benchmark problem.

The resulting NLDT with its associated four rules is shown in Fig. 8.

As shown in Table VI, the NLDT\_* with just two rules (NLDT-2) is too simplistic and does not recover well after reoptimization. However, the NLDT\_*s with four and seven rules achieve a near 100% closed-loop performance. Clearly, an NLDT\_* with more rules (NLDT-5 and NLDT-6) are not worth considering since both closed-loop performances and the size of rule sets are worse than NLDT\_*-4. Note that DNN produces a better reward, but not enough completion rate, and the policy is more complex with 4996 parameters.

Fig. 9 shows the closed-loop training curve for generating NLDT\_* from Depth-3 NLDT\(_{OL}\). The objective here is to maximize the closed-loop fitness (reward) $F_{CL}$ (7), which is expressed as the average of cumulative reward $R_c$ collected over $M$ episodes. It is evident that the cumulative reward for the best-population member climbs to the target reward of 200 at around 25th generation and the average cumulative reward of the population also catches up the best cumulative reward value with generations.

To check the repeatability of our approach, another run for generating NLDT\(_{OL}\) and NLDT\_* is executed. The resulting NLDT and corresponding equations are provided in the supplementary material. A visualization of the real-time closed-loop performance obtained using this new NLDT for two different rule-sets (i.e., before applying reoptimization and after applying the reoptimization through closed-loop training) is shown in https://youtu.be/DByYWTQ6X3E. It can be observed in the video that the closed-loop control executed using the Depth-3 NLDT\(_{OL}\) comprising of rules obtained directly from the open-loop training (i.e., without any reoptimization) is able to bring the LunarLander close to the target. However, the LunarLander hovers above the landing pad and the Depth-3 NLDT\(_{OL}\) is unable to land it in most occasions. Episodes in these cases are terminated after the flight-time runs out. On the other hand, the Depth-3 NLDT\_* comprising of rule sets obtained after reoptimization through closed-loop training is able to successfully land the LunarLander.

### VII. Scale-Up Study and Improvisation on Acrobot Control Problem

In this section, we investigate how the overall algorithm can be made more efficient in terms of—training time and scalability. To this purpose, we introduce a benchmark problem of planar serial manipulator (PSM). This problem is inspired from the classical Acrobot control problem [12]. A schematic of the PSM problem is provided in Fig. 10.
The state space for the PSM problem (Fig. 10) comprises of angular position $\theta_i$ and angular velocity $\omega_i$ of each joint. Thus, for a $n$-link manipulator involving $n$ revolute joints, the state space would be $2n$ dimensional. The motor is located at the last joint of the manipulator and is actuated using three torque values: $-\tau$, 0, and $\tau$. Each link is of 1 unit length and has its center-of-mass at its geometric center. The motor is assumed to be massless for the sake of simplicity. The base of the manipulator is located at (0, 0, 0) and the motion of the PSM is limited to the XZ plane. There is a downward gravitational pull ($g$) of 10 units (i.e., $-10$ along vertical $z$-axis). Torque is applied along the $y$-axis. The task in this problem is to take the end effector (i.e., tip of the last link, $Z$-coordinate $z_E$) of the PSM to a desired height of $H$ units by supplying torque to the motor located at the last joint [joint between ($n$ – 1)th and $n$th link]. The difficulty of this benchmark problem can be adjusted by the following.

1. Changing the number of links.
2. Changing the value of desired height level $H$.
3. Changing the value of torque $\tau$.
4. Placing extra motors at other joints.

We simulate the mechanics of the planar serial manipulator using PyBullet [34]: a Python-based physics engine.

In our work, we provide two case scenarios by focusing at the first three points of the above list. As mentioned before, by changing the number of links, the dimension of the state-space changes. The dimension of the action-space depends on the number of motors used. In the present work, we keep the number of motors fixed to one, having three discrete actions.

The details regarding two environments which are created and studied in this section are summarized in Table VII.

The reward function $r(x, A)$ is given by the following equation:

$$r(x) = \begin{cases} 
-1 - \left( \frac{H + 1 - z_E}{n + H + 1} \right)^2, & \text{if } z_E < H \\
100, & \text{if } z_E \geq H.
\end{cases} \quad (8)$$

The minimum value for $z_E$ is $-n$ when the entire manipulator is stretched to its full length and all joint angles (i.e., $\theta_i$) are at 0 deg.

At the beginning of an episode, joint angles $\theta_i$ of the manipulator are randomly initialized between $-5$ deg and $+5$ deg, and the angular velocities $\omega_i$ are initialized to a value between $-0.5$ rad/sec and $+0.5$ rad/sec.

In next sections, we discuss results obtained on the above two custom designed environments by using different procedures of inducing NLDE. The black-box AI (DNN) is trained using the PPO algorithm [2]. The resulting DNN has two hidden layers of 64 nodes each and has total 5699 parameters thereby making it massively uninterpretable.

### A. Ablation Study for Open-Loop Training

In this section, we launch two separate studies related to open-loop training. Procedure (see Fig. 3). It is seen in Section V-A that the open-loop training is conducted using a hierarchical bilevel-optimization algorithm, which is discussed at length in [30]. A dedicated bilevel-optimization algorithm is invoked to derive the split-rule $f(x)$ at a given conditional node. The upper level search is executed using a discrete version of a genetic algorithm and the lower level search is realized through an efficient real-coded genetic algorithm (RGA). Evolutionary algorithms are in general considered robust and have a potential to conduct more global search. However, being population driven, their search speed is often less than that of classical optimization algorithms. In this section, we study the effect of replacing the RGA with the classical sequential quadratic programming (SQP) optimization algorithm in the lower level of the overall bilevel algorithm for obtaining NLDT$_{OL}$. Later, closed-loop training based on RGA is applied to NLDT$_{OL}$ to obtain NLDT* by reoptimizing real valued coefficients of NLDT$_{OL}$ (Section V-B). We use the SciPy [35] implementation of SQP. The initial point required for SQP is obtained using the mixed dipole concept [30], [36]–[38].

For analysis, we induce an open-loop NLDT$_{OL}$ of depth-3 using a balanced training dataset of 10,000 datapoints. The testing dataset also comprises of 10,000 datapoints. The comparison of accuracy scores and average training time of inducing NLDT$_{OL}$ by using SQP and RGA algorithm at lower level is provided in Table VIII. For a given procedure (SQP or RGA) the best NLDT$_{OL}$ from ten independent runs is chosen and is reoptimized using closed-loop training (Section V-B). Statistics regarding closed-loop performance of NLDT* is shown in the last two columns of Table VIII. It is to note here that the closed-loop training is done using the RGA discussed in Section V-B.

It can be observed from the results that open-loop training done with SQP in lower level is about 70 times faster than the training done using RGA at the lower level. However, the training done with RGA has a better overall open-loop performance. Thus, if the task is to closely mimic the behaviour of black-box AI or if only a high classification accuracy is desired (in case of classification problems), then RGA is the recommended algorithm for lower level optimization to obtain NLDT$_{OL}$. However, NLDT* obtained after reoptimizing NLDT$_{OL}$ corresponding to SQP and RGA have similar closed-loop completion rate (last column of Table VIII). This implies that despite low open-loop accuracy scores, the open-loop training done using SQP in lower level is successful in determining the template of split rules $f(x)$ and topology of NLDT, which upon reoptimization via closed-loop training algorithm can fetch a decent performing NLDT*. During open-loop training, the search on weights and coefficients using SQP is possibly not as perfect as compared to the one achieved through RGA; however, the reoptimization done through closed-loop training can compensate this shortcoming of SQP algorithm and produce NLDT* with a respectable closed-loop performance. In addition, in either cases, the

| Env. Name | Motor Torque ($\tau$) | Desired Height ($H$) | # State Vars. |
|-----------|----------------------|---------------------|---------------|
| 5-Link PSM | 1,000                | +2                  | 10            |
| 10-Link PSM | 2,000                | +2                  | 20            |
TABLE VIII

| Algorithm | Open-Loop NLDTO_{OL} | Closed-Loop NLDT* |
|-----------|----------------------|-------------------|
|           | Training Accuracy     | Testing Accuracy  | Training Time (s) | Cumulative Reward | Completion Rate |
| SQP       | 62.46 ± 2.01         | 69.34 ± 5.39      | 15.29 ± 4.95      | -146.81 ± 8.53    | 96.00 ± 2.77   |
| RGA       | 71.14 ± 1.77         | 69.67 ± 4.39      | 1091.96 ± 319.18  | -152.64 ± 6.62    | 99.00 ± 1.71   |
| DNN       | NA                   | NA                | NA                | -183.35 ± 12.22   | 96.00 ± 5.19   |

Table VIII: Comparing Performance of Different Lower-Level Optimization Algorithms. For Comparison, Closed-Loop Performance of the Original DNN Policy Is Also Reported.

NLDT* obtained always has a better closed-loop performance than the original black-box DNN policy. This observation suggests that it is preferable to use SQP in lower level during open-loop training to quickly arrive at a rough structure of NLDTO_{OL} and then use closed-loop training to derive a high performing NLDT*.

B. Closed-Loop Visualization

In this section, we provide a visual insight into the closed-loop performance of DNN and NLDT*, which we derived in the previous section. In our case, the frequency of the simulation is set to 240 Hz, meaning that the transition to the next state is calculated using the time step of 1/240 s. Geometrically speaking, this implies that the Euclidean distance between states from neighboring time steps would be small. An AI (DNN or NLDT*) outputs the action value of 0 (−τ torque), 1 (0 torque) or 2 (+τ torque) for a given input state. Action versus time plots corresponding to different closed-loop simulation runs obtained by using DNN, NLDT* (SQP), and NLDT* (RGA) as controllers is shown in Fig. 11 for 5-link manipulator problem (plots for 10-link PSM are provided in the supplementary material).

Certain key observations can be made by looking at the plots in Fig. 11. The control output of DNN is more erratic, with sudden jerks as compared to the control output of NLDT* (SQP) and NLDT* (RGA). The performance of NLDT* in Fig. 11(b) and (c) is smooth and regular. This behavior can be due to the involvement of relatively simpler nonlinear rules (as compared to the complicated nonlinear rule represented by DNN), which are captured inside NLDT*. This is equivalent to the observation we made for the mountain car problem in Fig. 1(a) and (b), wherein the black-box AI had a very erratic behavior for the region of state space in the lower half of the state–action plot, while the output of NLDT was more smooth. In addition, it was seen in Table VIII that the NLDT* (irrespective of how its predecessor NLDTO_{OL} was obtained, that is, either through SQP or RGA in lower level) showed better closed-loop performance than the parent DNN policy. This observation implies that simpler rules expressed in the form of a nonlinear DT have better generalizability, thereby giving more robust performance for randomly initialized control problems. A careful investigation to the plots in Fig. 11(b) and (c) reveals that only two out of three allowable actions are required to efficiently execute the given control task of lifting the end effector of a 5-link serial manipulator. This concept will be used to reengineer the NLDT*, a discussion regarding which is provided in the next section.

C. Reengineering NLDT*

It is seen in action-time plots in Fig. 11(b) and (c) that not all actions are required to perform a given control task. Also, it may be possible that while performing a closed-loop control using NLDT, not all branches and nodes of NLDT are visited while deciding an action. Thus, the portion of NLDT that is not being utilized or is getting utilized very rarely can be pruned and the overall NLDT architecture can be made simpler. To illustrate this idea, we consider the NLDT which is derived for the 5-link manipulator problem. The topology of
the best performing NLDT\textsubscript{OL} (SQP) for the 5-link manipulator problem is shown in Fig. 12(a).

As mentioned before, this NLDT\textsubscript{OL} is trained on a balanced training dataset, which is generated by collecting state–action pairs using the oracle DNN controller. In the figure, at each node, information regarding its node-id, class distribution (given in square parenthesis), and the most dominating class is provided. Other than the root node (Node 0), all nodes are colored to indicate the dominating class, however, it is to note that only the class associated to leaf-nodes carry the actual meaning while predicting the action for a given input state. This NLDT\textsubscript{OL} comprises of four split rules in total. The class distribution for each node is obtained by counting how many datapoints from the balanced training dataset visited a given node. Thus, the root node comprises of all datapoints (total 10 000), which are then scattered according to the split rules present at each conditional node.

Fig. 12(b) provides the topology of NLDT*, which is obtained after reoptimizing NLDT\textsubscript{OL} of Fig. 12(a) using closed-loop training. As discussed in Sections V-A and V-B, the topology of the tree and the structure of nonlinear rules is identical for both: NLDT\textsubscript{OL} and NLDT*. However, the weights and biases of NLDT* are updated to enhance the closed-loop control performance. Similar to NLDT\textsubscript{OL} of Fig. 12(a), the information regarding node-id and class-distribution is provided for all the nodes of NLDT* in Fig. 12(b). However, the data distribution in NLDT* is obtained by using the actual state–action data from closed-loop simulations, wherein NLDT* is used as a controller. Total 10 000 datapoints are collected in the form of sequential states–action pairs from closed-loop simulation runs, which are executed using NLDT*. As it can be seen in the root node of NLDT* [Fig. 12(b)], out of 10 000 states visited during closed-loop control, action 0 (−τ torque) was chosen by NLDT* in 7736 states and action 2 (+τ torque) was chosen for 2264 states. Action 1 was never selected by NLDT* during closed-loop control. This is consistent with what we have observed in the action versus time plot in Fig. 11(b), wherein most of the time action 0 was executed, while there was no event where action 1 was executed. The flow of these 10000 state–action pairs through NLDT* and their corresponding distribution in each node of NLDT* is provided in Fig. 12(b). It can be observed that Node 5, Node 4, and Node 8 of NLDT* are never visited during closed-loop control. This implies that splits at Node 2, Node 1, and Node 6 are redundant. Thus, the part of NLDT* shown in red box in Fig. 12(b) can be pruned and the overall topology of the tree can be simplified. The pruned NLDT* will involve only one split (occurring at Node 0) and two leaf nodes: 1) Node 1 and 2) Node 6. However, it is to note here that we need to reassign class labels to the newly formed leaf nodes (i.e., Node 1 and Node 6) based on the data-distribution from closed-loop simulations. The old class labeling for the Node 1 and Node 6 was done based on the open-loop data [Fig. 12(a)]. Using the new class distribution corresponding to NLDT* [Fig. 12(b)], Node 1 is relabeled with Class-2 and Node 6 with Class-0. The pruned version of NLDT* of Fig. 12(b) is provided in Fig. 13 [here nodes are renumbered, with Node 6 of NLDT* in Fig. 12(b) renumbered to Node 2 in the pruned NLDT* as shown in Fig. 13].

The split rule corresponding to the root node is also shown. Interestingly, out of ten total state variables, only two variables are used to decide which action to execute for closed-loop control: 1) \( x_1 \) corresponds to the angular position of the second link and 2) \( x_9 \) variable corresponds to the angular velocity of the last joint. The above rule indicates a single and interpretable relationship: action 2 must be invoked when \( x_9 \geq -0.115 / \sqrt{x_{1}^{2}} \), otherwise, action 0 must be invoked. This NLDT* corresponds to a cumulative reward of \(-139.78 \pm 7.65\) and performs at \(99.00 \pm 1.17\)% completion rate, providing a remarkably simple interpretation of the control strategy for this apparently complex problem.

VIII. CONCLUSION

In this article, we have proposed a two-step strategy to arrive at hierarchical and relatively interpretable controller using an NLDT concept. The NLDT training phases use recent advances in nonlinear optimization to focus its search on rule structure and details describing weights and biases of the rules.
micro-decision-making and a ‘right to explanation,’” Inform. Fusion, vol. 44, no. 12, pp. 2613–2625, Dec. 2014.

[7] X. Xu, Z. Huang, D. Graves, and W. Pedrycz, “A clustering-based graph Laplacian framework for value function approximation in reinforcement learning,” IEEE Trans. Cybern., vol. 44, no. 12, pp. 2613–2625, Dec. 2014.

[8] W. J. Murdoch, C. Singh, K. Kumbhar, R. Abbasai-Ali, and B. Yu, “Interpretable machine learning: Definitions, methods, and applications,” 2019, arXiv:1901.04597.

[9] Z. C. Lipton, “The myths of model interpretability,” Queue, vol. 16, no. 3, pp. 31–57, 2018.

[10] M. A. Ahmad, C. Eckert, and A. Teredesai, “Interpretable machine learning in healthcare,” in Proc. ACM Int. Conf. Bioinf. Comput. Biol. Health Inform., 2018, pp. 359–560.

[11] B. Goodman and S. Flaxman, “European union regulations on algorithmic decision-making and a ‘right to explanation,”’ Al Mag., vol. 38, no. 3, pp. 50–57, 2017.

[12] R. S. Sutton, “Generalization in reinforcement learning: Successful examples using sparse coarse coding,” in Advances in Neural Information Processing Systems, vol. 8, D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds. Cambridge, MA, USA: MIT Press, 1996, pp. 1038–1044.

[13] J. Peters, K. Mülling, and Y. Altun, “Relative entropy policy search,” in Proc. 24th AAAI Conf. Artif. Intell. (AAAI), vol. 10, 2010, pp. 1607–1612.

[14] W. D. Smart and L. P. Kaelbling, “Practical reinforcement learning in continuous spaces,” in Proc. Int. Conf. Mach. Learn. (ICML), 2000, pp. 903–910.

[15] R. Noothigattu et al., “Interpretable multi-objective reinforcement learning through policy orchestration,” 2018, arXiv:1809.08343.

[16] F. Maes, R. Fonteneau, L. Wehenkel, and D. Ernst, “Policy search in a space of simple closed-form formulas: Towards interpretability of reinforcement learning,” in Proc. Int. Conf. Disc. Sci., 2012, pp. 37–51.

[17] D. Hein, S. Udluft, and T. A. Runkler, “Interpretable policies for reinforcement learning by genetic programming,” Eng. Appl. Artif. Intell., vol. 76, pp. 158–169, Nov. 2017.

[18] G. Liu, O. Schulte, W. Zhu, and Q. Li, “Toward interpretable deep reinforcement learning with linear model U-trees,” in Proc. Joint Eur. Conf. Mach. Learn. Knowl. Disc. Databases, 2018, pp. 414–429.

[19] A. Verma, V. Murari, R. Singh, P. Kohli, and S. Chaudhuri, “Programmatically interpretable reinforcement learning,” 2018, arXiv:1804.02477.

[20] J. Kennedy and R. Eberhart, “Particle swarm optimization,” in Proc. Int. Conf. Neural Netw., vol. 4, 1995, pp. 1942–1948.

[21] D. Hein, A. Hentschel, T. Runkler, and S. Udluft, “Particle swarm optimization for generating interpretable fuzzy reinforcement learning policies,” Eng. Appl. Artif. Intell., vol. 65, pp. 87–98, Oct. 2017.

[22] L. Breiman, Classification and Regression Trees. Boca Raton, FL, USA: Routledge, 2017.

[23] G. Van der Wiele, O. Janssens, F. De Turck, and S. Van Hoeye, “Genesim: Genetic extraction of a single, interpretable model,” 2016, arXiv:1611.05722.

[24] D. Ernst, P. Geurts, and L. Wehenkel, “Tree-based batch mode reinforcement learning,” J. Mach. Learn. Res., vol. 6, no. 18, pp. 503–556, 2005.

[25] J. L. Bentley, “Multidimensional binary search trees used for associative searching,” Commun. ACM, vol. 18, no. 9, pp. 509–517, 1975.

[26] A. Silva, M. Gombolay, T. Killian, I. Jimenez, and S.-H. Son, “Optimization methods for interpretable differentiable decision trees applied to reinforcement learning,” in Proc. Int. Conf. Artif. Intell. Stat., 2020, pp. 1855–1865.

[27] D. Dhebar and K. Deb, “Interpretable rule discovery through bivel optimization of split-rules of nonlinear decision trees for classification problems,” IEEE Trans. Cybern., vol. 51, no. 11, pp. 5573–5584, Nov. 2021.

[28] A. Sinha, P. Malo, and K. Deb, “A review on bilevel optimization: From classical to evolutionary approaches and applications,” IEEE Trans. Evol. Comput., vol. 22, no. 2, pp. 276–295, Apr. 2018.

[29] S. Nageshrao, B. Costa, and D. Felipe, “Interpretable approximation of a deep reinforcement learning agent as a set of if-then rules,” in Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA), 2019, pp. 216–221.
[33] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-learning!” 2015, arXiv:1509.06461.
[34] E. Coumans and Y. Bai, “PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning,” 2020. Online. Available: http://pybullet.org
[35] P. Virtanen et al., “SciPy 1.0: Fundamental algorithms for scientific computing in Python,” Nat. Methods, vol. 17, pp. 261–272, 2020.
[36] L. Bobrowski and M. Kretowski, “Induction of multivariate decision trees by using dipolar criteria,” in Proc. Eur. Conf. Principles Data Min. Knowl. Disc., 2000, pp. 331–336.
[37] M. Kretowski, “An evolutionary algorithm for oblique decision tree induction,” in Proc. Int. Conf. Artif. Intell. Soft Comput., 2004, pp. 432–437.
[38] M. Kretowski and M. Grzes, “Global induction of oblique decision trees: An evolutionary approach,” in Intelligent Information Processing and Web Mining. Berlin, Germany: Springer, 2005, pp. 309–318.
[39] K. Deb and A. Srinivasan, “Innovization: Innovating design principles through optimization,” in Proc. 8th Annua. Conf. Genet. Evol. Comput., 2006, pp. 1629–1636.

Yashesh Dhebar received the bachelor’s and master’s degrees in mechanical engineering from the Indian Institute of Technology Kanpur, Kanpur, India, in 2015, with his thesis on modular robotics and their operation in cluttered environments, and the Ph.D. degree from the Department of Mechanical Engineering, Michigan State University, East Lansing, MI, USA, in 2020, under the guidance of Prof. K. Deb, with his thesis focused at developing Interpretable AI models using modern optimization techniques.

He was with Michigan State University. He is currently working with the HEEDS Design Space Exploration Software Development Team, Siemens DISW, East Lansing.

Kalyanmoy Deb (Fellow, IEEE) received the B.Tech. (Hons.) degree from the Indian Institute of Technology Kharagpur, Kharagpur, India, in 1985, and the Ph.D. and M.S. degrees from the University of Alabama, Tuscaloosa, AL, USA, in 1989 and 1991, respectively.

He is a University Distinguished Professor and the Koenig Endowed Chair Professor with the Department of Electrical and Computer Engineering, Michigan State University, East Lansing, MI, USA. He has written more than 580 research papers with more than 168,000 Google Scholar citations. His research interests are in evolutionary optimization algorithms and their application in optimization and machine learning.

Dr. Deb received many awards, including the IEEE EC Pioneer Award, the Infosys Prize, and the Edgeworth-Pareto Award. He is an ASME.

Subramanya Nageshrao received the Ph.D. degree from the Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands, in 2016.

He is currently a Research Engineer with Ford Green Field Labs, Palo Alto, CA, USA. Prior to that, he worked as a Postdoctoral Research Scholar with the University of Michigan Ann-Arbor, Ann Arbor, MI, USA. He is actively leveraging machine learning methods particularly reinforcement learning techniques to build next generation state-of-the-art advanced driver assistance technologies.

Ling Zhu received the master’s and Ph.D. degrees from the Computer Science and Engineering Department, Michigan State University, East Lansing, MI, USA, under the supervision of Prof. S. Kulkarni and Prof. K. Deb.

She was with the Advance Research and Engineering, Ford Motor Company, Dearborn, MI, USA. She is currently an Algorithm Engineer with Microsoft, Mountain View, CA, USA, and working on research and development in advanced 3-D sensing technology. Her research interests include 3-D machine vision, computer vision, evolutionary optimization, metaheuristic optimization, robotics, and distributed systems and computing.

Dimitar Filev (Fellow, IEEE) received the integrated bachelor’s and master’s degree in electrical engineering and the Ph.D. degree in electrical and electronics engineering from Czech Technical University, Prague, Czechia, in 1976 and 1979, respectively.

He is a Senior Henry Ford Technical Fellow with Ford Motor Company, Detroit, MI, USA. Prior to joining Ford Motor Company in 1994, he held faculty positions with the State University of New York, Albany, NY, USA, from 1991 to 1994, and Bulgarian Academy of Sciences, Sofia, Bulgaria, from 1980 to 1991. His technical contributions include more than 100 granted U.S. patents and over 200 publications with more than 16,000 peer citations.

Dr. Filev is a member of NAE.