Journal of Machine Learning Research 1 (2023) 1-38 Submitted 4/00; Published 10/00

Learning Interpretable, High-Performing Policies for Continuous Control

Rohan Paleja[*,1,†] RPALEJA3@GATECH.EDU
Letian Chen[*,1] LETIAN.CHEN@GATECH.EDU
Yaru Niu[*,2] YARUN@ANDREW.CMU.EDU
Andrew Silva[1] ASILVA9@GATECH.EDU
Zhaoxin Li[1] ZLI3088@GATECH.EDU
Songan Zhang[3] SZHAN117@FORD.COM
Chace Ritchie[1] CHACERITCHIEDEV@GMAIL.COM
Sugju Choi[1] SCHOI24@GATECH.EDU
Kimberlee Chestnut Chang[4] CHESTNUT@LL.MIT.EDU
Hongtei Eric Tseng[3] HONGTEI.TSENG@GMAIL.COM
Yan Wang[3] YWANG21@FORD.COM
Subramanya Nageshrao[3] SNAGESHR@FORD.COM
Matthew Gombolay[1] MATTHEW.GOMBOLAY@CC.GATECH.EDU

Editor:

Abstract

Interpretability in machine learning is critical for the safe deployment of learned policies across legally-regulated and safety-critical domains. While gradient-based approaches in reinforcement learning have achieved tremendous success in learning policies for continuous control problems such as robotics and autonomous driving, the lack of interpretability is a fundamental barrier to adoption. We propose Interpretable Continuous Control Trees (ICCTs), a tree-based model that can be optimized via modern, gradient-based, reinforcement learning approaches to produce high-performing, interpretable policies. The key to our approach is a procedure for allowing direct optimization in a sparse decision-tree-like representation. We validate ICCTs against baselines across six domains, showing that ICCTs are capable of learning policies that parity or outperform baselines by up to 33% in autonomous driving scenarios while achieving a 300x-600x reduction in the number of parameters against deep learning baselines. We prove that ICCTs can serve as universal function approximators and display analytically that ICCTs can be verified in linear time. Furthermore, we deploy ICCTs in two realistic driving domains, based on interstate Highway-94

1. Georgia Institute of Technology, Atlanta, GA 30332, USA
2. Carnegie Mellon University, Pittsburgh, PA 15213, USA
3. Ford Motor Company, Dearborn, MI 48120, USA
4. MIT Lincoln Laboratory, Lexington, MA 02421, USA

∗. Equal contribution
†. Corresponding author

and 280 in the US. Finally, we verify ICCT's utility with end-users and find that ICCTs are rated easier to simulate, quicker to validate, and more interpretable than neural networks.

Keywords: Differentiable Decision Tree, Interpretable Machine Learning, Autonomous Driving, Reinforcement Learning, Trustworthy Learning

1. Introduction

Reinforcement learning (RL) with deep function approximators has enabled the generation of high-performance continuous control policies across a variety of complex domains, from robotics Lillicrap et al. (2016) and autonomous driving Wu et al. (2017) to protein folding Jumper et al. (2021) and traffic regulation Cui et al. (2021). These approaches hold tremendous promise in real-world applicability and have the potential to increase traffic safety (Katrakazas et al., 2015), decrease traffic congestion, increase average traffic speed in human-driven traffic (Cui et al., 2021), reduce CO_2 emissions, and allow for more affordable transportation (Abe, 2019). However, while the performance of these controllers opens up the possibility of real-world adoption, the conventional deep-RL policies used in prior work (Lillicrap et al., 2016; Wu et al., 2017; Cui et al., 2021) lack *interpretability*, limiting deployability in safety-critical and legally-regulated domains (Doshi-Velez and Kim, 2017; Letham et al., 2015; Bhatt et al., 2019; Voigt and Von dem Bussche, 2017).

White-box approaches, as opposed to typical black-box models (e.g., deep neural networks) used in deep-RL, model decision processes in a human-readable representation. Such approaches afford interpretability, allowing users to gain insight into the model's decision-making behavior. In autonomous driving, such models would provide insurance companies, law enforcement, developers, and passengers with insight into how an autonomous vehicle (AV) reasons about state features and makes decisions. Utilizing such white-box approaches within machine learning is necessary for the deployment of autonomous vehicles and essential in building trust, ensuring safety, and enabling developers to inspect and verify policies before deploying them to the real world (Olah et al., 2018; Hendricks et al., 2018; Anne Hendricks et al., 2018). In this work, we present a novel tree-based architecture that affords gradient-based optimization with modern RL techniques to produce high-performance, interpretable policies for continuous control applications. We note that our proposed architecture can be applied to a multitude of continuous control problems ranging from robotics (Lillicrap et al., 2016), protein folding (Jumper et al., 2021), and traffic regulation (Cui et al., 2021) to high-speed autonomous driving (Wu et al., 2017) and autopilots for landing spacecraft (Banerjee and Padhi, 2018).

Prior work (Olah et al., 2018; Kim, 2015; Hendricks et al., 2018) has attempted to approximate interpretability via explainability, a practice that can have severe consequences (Rudin, 2018). While the explanations produced in prior work can help to partially explain the behavior of a control policy, the explanations are not guaranteed to be accurate or generally applicable across the state-space, leading to erroneous conclusions and a lack of accountability of predictive models (Rudin, 2018). In autonomous driving, where understanding a decision-model is critical to avoiding collisions, local explanations are insufficient. An *interpretable model*, instead, provides a transparent *global* representation of a policy's behavior. This model can be understood directly by its structure and parameters (Ciravegna et al., 2021) (e.g., linear models, decision trees, and our ICCTs), offering verifiability and guarantees that are not afforded by post-hoc explainability frameworks. Few works have attempted to learn an interpretable model directly; rather, prior work has attempted policy distillation to a decision tree (Frosst and Hinton, 2017; Bastani et al., 2018b; Wu et al., 2018)

or imitation learning via a decision tree across trajectories generated via a deep model (Bastani et al., 2018a), leaving much to be desired. Interpretable RL remains an open challenge (Rudin et al., 2021). In this work, we directly produce high-performance, interpretable policies represented by a minimalistic tree-based architecture augmented with low-fidelity linear controllers via RL, providing a novel interpretable RL architecture. Our Interpretable Continuous Control Trees are human-readable, allow for closed-form verification (associated with safety guarantees), and parity or outperform baselines by up to 33% in autonomous driving scenarios. In this work, our key contributions are:

1. We propose Interpretable Continuous Control Trees (ICCTs), a novel tree-based architecture that can be optimized via gradient descent with modern RL algorithms to produce high-performance, interpretable continuous control policies. We provide several extensions to prior differentiable decision tree (DDT) frameworks to increase expressivity and allow for direct optimization of a sparse decision-tree-like representation.

2. We show that our ICCTs are universal function approximators and can thus be utilized to learn continuous control policies in any domain, assuming that the ICCT has a reasonable depth.

3. We empirically validate ICCTs across six continuous control domains, including four autonomous driving scenarios. Further, we demonstrate ICCT's ability to learn driving policies in complex domains grounded in realistic real-world lane geometries, including the I-94 highway in Michigan, USA, and the I-280 highway in California, USA.

4. We show that our ICCTs can be verified in linear time, a vital characteristic in assessing and understanding a model's behavior under a set of inputs. Whereas black-box approaches are difficult to verify, our ICCTs can be verified quickly, providing the possibility of safety guarantees and opening the door for safe, real-world adoption.

5. We demonstrate the utility of our ICCTs with end-users through a human-subjects study (N=34) and show that the ICCT is rated by users as easier to simulate, quicker to validate, and more interpretable than neural networks.

This paper presents our work in the field of Interpretable Reinforcement Learning and Explainable AI, highlighting prior techniques in Section 2. In Section 3, we introduce the necessary preliminary work on Differentiable Decision Trees and Reinforcement Learning. Our Methodology, covered in Section 4, outlines the ICCT architecture and the Differentiable Crispification technique used for enabling policy updates via gradient-based techniques. Section 5 establishes ICCTs as universal function approximators, and Section 6 analyzes the time complexity for model verification. In Sections 7 and 8, we introduce and evaluate our ICCT across six continuous control domains. In Section 9, we offer a qualitative example of a learned policy within the Lunar Lander domain to showcase the interpretability of our model. Section 10 explores the tradeoff between performance, leaf controller sparsity, and tree depth. We compare our differential crispification method to the Gumbel-Softmax procedure in Section 11. Section 12 demonstrates the interpretability and utility of ICCTs through a 14-car physical robot demonstration. In Section 13, we introduce two realistic driving domains based on real-world lane geometries and find that trained ICCTs can produce safe, high-performance behavior that follows traffic regulations. Finally, Section 14 presents a user study evaluating the interpretability of our model.

2. Related Work

Due to recent accidents with autonomous vehicles (cf. Yurtsever et al. (2020)), there has been growing interest in developing Explainable AI (xAI) approaches to understand a control policy's decision-making and ensure robust and safe operation. xAI is concerned with understanding and interpreting the behavior of AI systems (Linardatos et al., 2021). In recent years, the necessity for human-understandable models has increased greatly for safety-critical and legally-regulated domains, many of which involve continuous control (e.g., specifying joint torques for a robot arm or the steering angle for an autonomous vehicle) (Kim and Canny, 2017; Doshi-Velez and Kim, 2017). In such domains, prior work (Schulman et al., 2017; Lillicrap et al., 2016; Fujimoto et al., 2018; Haarnoja et al., 2018) has typically used highly-parameterized deep neural networks in order to learn high-performance policies, completely lacking in model transparency.

Interpretable machine learning approaches refers to a subset of xAI techniques that produce globally transparent policies (i.e., humans can inspect the entire model, as in a decision tree (Loh, 2011; Basak, 2004; Olaru and Wehenkel, 2003) or rule list (Angelino et al., 2017; Weiss and Indurkhya, 1995; Letham et al., 2015; Chen and Rudin, 2017)). Decision trees (Loh, 2011) represent a hierarchical structure where an input decision can be traced to an output via evaluation of decision nodes (i.e., "test" on an attribute) until arrival at a leaf node. Decision nodes within the tree are able to split the problem space into meaningful subspaces, simplifying the problem as the tree gets deeper (Laptev and Buhmann, 2014; Kontschieder et al., 2015; Tanno et al., 2018). Decision trees provide *global* explanations of a decision-making policy that are valid throughout the input space (Barbiero et al., 2021), as opposed to local explanations typically provided via "post-hoc" explainability techniques (Correia Ribeiro, 2018; Silva and Gombolay, 2021; Paleja et al., 2020). Several approaches have attempted to distill trained black-box neural network models into decision trees (Wu et al., 2020; Bastani et al., 2018b). While these approaches produce interpretable models, the resulting model is an approximation of the neural network rather than a true representation of the underlying model. Our work, instead, directly learns an interpretable tree-based policy via reinforcement learning, producing a model that can be directly verified without utilizing error-prone post-hoc explainability techniques. We emphasize that explainability stands in contrast to interpretability, as explanations may fail to capture the true decision-making process of a model or may apply only to a local region of the decision-space, thereby preventing a human from building a clear or accurate mental model of the entire policy (Rudin, 2018; Lipton, 2018; Adadi and Berrada, 2018; Paleja et al., 2021).

Recently, Rudin et al. (2021) presented a set of grand challenges in interpretable machine learning to guide the field toward solving critical research problems that must be solved before machine learning can be safely deployed within the real world. In this work, we present a solution to directly assess two challenges: (1) Optimizing sparse logical models such as decision trees and (10) Interpretable reinforcement learning. We propose a novel high-performing, sparse tree-based architecture, Interpretable Continuous Control Trees (ICCTs), which allows end-users to directly inspect the decision-making policy and developers to verify the policy for safety guarantees.

3. Preliminaries

In this section, we review differentiable decision trees (DDTs) and reinforcement learning.

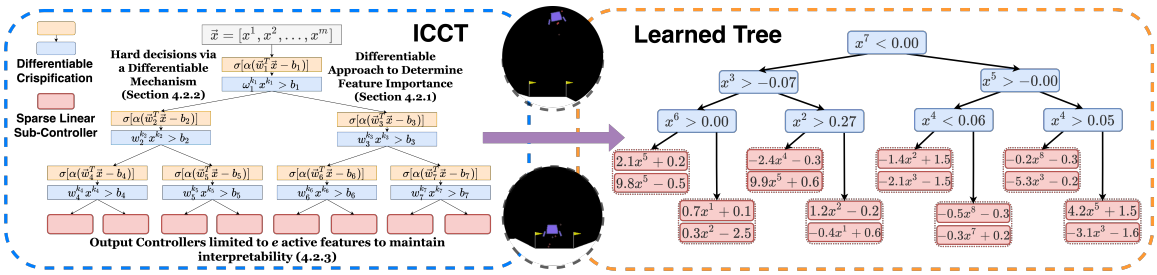


Figure 1: The ICCT framework (left) displays decision nodes, both in their fuzzy form (orange blocks) and crisp form (blue blocks[2]), and sparse linear leaf controllers with pointers to sections discussing our contributions. A learned representation of a high-performing ICCT policy in Lunar Lander (right) displays the interpretability of our ICCTs. Each decision node is conditioned upon only a single feature and the sparse linear controllers (to control the main engine throttle and left/right thrusters) are set to have only **one** active feature.

3.1 Differentiable Decision Trees (DDTs)

Prior work has proposed differentiable decision trees (DDTs) (Suárez and Lutsko, 1999; Silva and Gombolay, 2021; Paleja et al., 2020; Tambwekar et al., 2023) – a neural network architecture that takes the topology of a decision tree (DT). Similar to a decision tree, DDTs contain decision nodes and leaf nodes; however, each decision node within the DDT utilizes a sigmoid activation function (i.e., a "soft" decision) instead of a Boolean decision (i.e., a "hard" decision). Each decision node, i, is represented by a sigmoid function, displayed in Equation 1.

$$y_i = \frac{1}{1 + \exp(-\alpha(\vec{w}_i^T \vec{x} - b_i))} \tag{1}$$

Here, the features vector describing the current state, $\vec{x}$, are weighted by $\vec{w}_i$, and a splitting criterion, b_i, is subtracted to form the splitting rule. y_i is the probability of decision node i evaluating to TRUE, and α governs the steepness of the sigmoid activation, where $\alpha \to \infty$ results in a step function. Prior work with discrete-action DDTs modeled each leaf node with a probability distribution over possible output classes (Silva and Gombolay, 2021; Paleja et al., 2020). Leaf node distributions, $\vec{L}$, are then weighted by the probability of reaching the respective leaf and summed to produce a final action distribution over possible outputs. For a simple 4-leaf tree, this results in an *fuzzy* output distribution with a complex interplay of node and leaf probabilities, displayed in Equation 2.

$$P(a|x) = \vec{L}_1(y_1 * y_2) + \vec{L}_2(y_1 * (1 - y_1)) + \vec{L}_3((1 - y_1) * y_3) + \vec{L}_4((1 - y_1)(1 - y_3)) \tag{2}$$

3.1.1 CONVERSION OF A DDT TO A DT

DDTs with decision nodes represented in the form of Equation 1 are not interpretable. As DDTs maintain a one-to-one correspondence to DTs with respect to their structure, prior work (Silva and

2. For figure simplicity, when displaying the crisp node (blue block), we assume $\alpha > 0$ in the fuzzy node (orange block). If $\alpha < 0$, the sign of the inequality would be flipped (i.e., $w_i^{k_i} x^{k_i} < b$).

Gombolay, 2021; Paleja et al., 2020) proposed a methodology to convert a DDT into an interpretable decision tree (a process termed "crispification"). To create an interpretable, "crisp" tree from a differentiable form of the tree, prior work adopted a simplistic procedure. Starting with the differentiable form, prior work first converts each decision node from a linear combination of all variables into a single feature check (i.e., a 2-arity predicate with a variable and a threshold). The feature reduction is accomplished by considering the feature dimension corresponding to the weight with the largest magnitude (i.e., most impactful), $k = \arg\max_j |w_i^j|$, where j represents the feature dimension, resulting in the decision node representation $y_i = \sigma(\alpha(w_i^k x^k - b_i))$. The sigmoid steepness, α, is also set to infinity, resulting in a "hard" decision (branch left OR right) (Silva and Gombolay, 2021; Paleja et al., 2020). After applying this procedure to each decision node, decision nodes are represented by $y_i = \mathbb{1}(w_i^k x^k - b_i > 0)$. As each leaf node is represented as a probability mass function over output classes in prior work, each leaf node, l_d, indexed by d, must be modified to produce a single output class, o_d, during crispification. As such, we can apply an argument max, $o_d = \arg\max_a l_d^a$, where a denotes the action dimension, to find the maximum valued class within the d-th leaf distribution.

Deficiency of direct conversion from DDT to DT: This simplistic crispification procedure results in an interpretable crisp tree that is inconsistent with the original DDT (model differences arise from each *argmax* operation and setting the signmoid steepness, α, to infinity). These inconsistencies can lead to performance degradation of the interpretable model, as we show in Section 8, and results in an interpretable model that is not representative of and inconsistent with the model learned via reinforcement learning.

In our work, we address these limitations by designing a novel architecture that updates its parameters via gradient descent while maintaining an interpretable decision-tree-like representation, thereby avoiding any inconsistencies generated through a post-hoc crispification procedure. To the best of our knowledge, we are the first work to deploy an interpretable tree-based framework for continuous control.

3.2 Reinforcement Learning (RL)

A Markov Decision Process (MDP) M is defined as a 6-tuple $\langle S, A, R, T, \gamma, \rho_0 \rangle$. S is the state-space, A is the action-space, $R(s, a)$ is the reward received by an agent for executing action, a, in state, s, $T(s'|s, a)$ is the probability of transitioning from state, s, to state, s', when applying action, a, $\gamma \in [0, 1]$ is the discount factor, and $\rho_0(s)$ is the initial state distribution. A policy, $\pi(a|s)$, gives the probability of an agent taking action, a, in state, s. The goal of RL is to find the optimal policy, $\pi^* = \arg\max_\pi \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right]$ to maximize cumulative discounted reward, where $\tau = \langle s_0, a_0, \cdots, s_T, a_T \rangle$ is an agent's trajectory.

In this work, while ICCTs are framework-agnotistic (i.e., ICCTs will work with any RL update rule), we proceed with Soft Actor-Critic (SAC) (Haarnoja et al., 2018) as our RL algorithm due to its learning stability and sample efficiency. The actor objective within SAC is given in Equation 3, where $Q_w(s_t, a_t)$ is expected, future discounted reward parameterized by ω and α_τ is a temperature parameter that determines the relative importance of the stochastic policy entropy versus the reward.

$$J_\pi(\theta) = \mathbb{E}_{s_t\sim\mathcal{D}}[\mathbb{E}_{a_t\sim\pi_\theta}[\alpha_\tau \log(\pi_\theta(a_t|s_t)) - Q_\omega(s_t, a_t)]] \tag{3}$$

4. Method

In this section, we introduce our ICCTs, a novel interpretable reinforcement learning architecture. ICCTs are able to maintain interpretability while representing high-performance continuous control policies, making them suitable for applications that require trust and accountability, such as robotic manipulation and autonomous vehicle control. We provide several extensions to prior DDT frameworks within our proposed architecture, including 1) a differentiable crispification procedure allowing for optimization in a sparse decision-tree-like representation and 2) the addition of sparse linear leaf controllers to increase expressivity while maintaining legibility.

4.1 ICCT Architecture

Our ICCTs are initialized to be a symmetric decision tree with N_l decision leaves and $N_l - 1$ decision nodes. A depiction of our ICCT can be seen in Figure 1, with decision leaves shown in red and decision nodes shown in blue. The tree depth can be determined by $\log_2(N_l)$. Each decision leaf is represented by a sparse linear controller that operates on $\vec{x}$. Decisions are routed via decision nodes toward a leaf controller, which is then used to produce the continuous control output (e.g., acceleration or steering wheel angle). Our ICCT is similar to hierarchical models, which encompass a high-level controller that governs and coordinates multiple low-level controllers. Prior work has shown this to be a successful paradigm in continuous control (Nachum et al., 2018).

Each decision node, i, has an activation steepness weight, α, associated weights, $\vec{w}_i$, with cardinality, m, matching that of the input feature vector, $\vec{x}$, and a scalar bias term, b_i, similar to that of Equation 1. Each leaf node, l_d, where $d \in \{1, \ldots, N_l\}$, contains per-leaf weights, $\vec{\beta}_d \in \mathbb{R}^m$, per-leaf selector weights that learn the relative importance of candidate features, $\vec{\theta}_d \in \mathbb{R}^m$, per-leaf bias terms, $\vec{\phi}_d \in \mathbb{R}^m$, and per-leaf scalar standard deviations, γ_d. We note that if the action space is multi-dimensional, then *only* the leaf controllers (and associated weights) are expanded across $|A|$ dimensions, where $|A|$ is the cardinality of the action space. For each action dimension, the mean of the output action distribution is represented by the linear controller, l_d.

$$l_d \triangleq (\vec{u} \circ \vec{\beta}_d)^T (\vec{u} \circ \vec{x}) - \vec{u}^T \vec{\phi}_d \tag{4}$$

Before enforcing leaf controller sparsity (i.e., by forcing the controller to condition upon only a subset of features, Section 4.2.3), $\vec{u} = [1, ..., N_l]^T$ is an all-ones vector, representing the set of active features within the leaf node, in which case Equation 4 can be simplified as $l_d = \vec{\beta}_d^T \vec{x} - \vec{u}^T \vec{\phi}_d$. The output action can be determined via sampling ($a \sim \mathcal{N}(\vec{\beta}_d^T \vec{x} - \vec{u}^T \vec{\phi}_d, \gamma_d)$) during training and directly via the mean during runtime. We term decision nodes that are represented as Equation 1 as fuzzy decision nodes, displayed by the orange rectangles within the left-hand side of Figure 1. Similarly, we term the leaf node, l_d, when it is represented in the dense representation of $\vec{\beta}_d^T \vec{x} - \vec{u}^T \vec{\phi}_d$, as a fuzzy leaf node. Here, we parameterize the bias term as a vector, $\vec{\phi}_d$, instead of a scalar as in our decision nodes to provide a corresponding bias for each feature and facilitate feature-wise optimization across different dimensions of the bias term.

Utilizing a novel differentiable crispification procedure to convert fuzzy decision nodes into crisp decision nodes (i.e., 2-arity predicate with a variable and a threshold) and fuzzy leaf nodes into sparse leaf nodes (i.e., linear controller conditioned upon a small subset of features), our model representation follows that of a decision tree with sparse linear controllers at the leafs (shown on the right-hand side of Figure 1). We further discuss our differentiable crispification procedure in

Sections 4.2.1-4.2.2 (i.e., the mechanism that translates orange blocks to blue within Figure 1) and leaf controller sparsification procedure in Section 4.2.3.

While decision trees (DT) are generally considered interpretable (Letham et al., 2015), trees of arbitrarily large depths can be difficult to understand (Ghose and Ravindran, 2020) and simulate (Lipton, 2018). A sufficiently sparse DT is desirable and considered interpretable (Lakkaraju et al., 2016). Furthermore, the utilization of linear controllers at the leaves also allows us to maintain interpretability, as linear controllers are widely used and generally considered interpretable for humans (Hein et al., 2020). However, for large feature spaces typically encountered in real-world problems, such a controller would not be interpretable. As such, in our work, we utilize *sparse* linear controllers at the leaves to balance the trade-off between sparsity/complexity in logic, model depth, and performance.

4.2 ICCT Key Elements

In this section, we discuss our ICCT's interpretable procedure for determining an action given an input feature. As our ICCT configuration maintains interpretability both during training via RL and deployment, the inference of an action must allow gradient flow. We present a novel approach that allows for direct optimization of sparse logical models via an online differentiable crispification procedure to determine feature importance (Section 4.2.1) and allows for bifurcate decisions (Section 4.2.2). In Algorithm 1, we provide general pseudocode representing our ICCT's decision-making process.

At each timestep, the ICCT model, $I(\cdot)$, receives a state feature, $\vec{x}$. To determine an action in an interpretable form, in Steps 1 and 2 of Algorithm 1, we start by applying the differentiable crispification approaches of NODE_CRISP and OUTCOME_CRISP to decision nodes so that each decision node is only conditioned upon a single variable (Section 4.2.1), and the evaluation of a decision node results in a Boolean (Section 4.2.2). Once the operations are completed, in Step 3, we can utilize the input feature, $\vec{x}$, and logically evaluate each decision node until arrival at a linear leaf controller (INTERPRETABLE_NODE_ROUTING). The linear leaf controller is then modified, in Step 4, to only condition upon e features, where e is a sparsity parameter specified a priori (Section 4.2.3). Finally, an action can be determined via sampling from a Gaussian distribution conditioned upon the mean generated via the input-parameterized sparse leaf controller, l_d^*, and scalar variance maintained within the leaf, γ_d, during training (Step 6) or directly through the outputted mean (Step 8) during runtime.

Algorithm 1 ICCT Action Determination

Input: ICCT $I(\cdot)$, state feature $\vec{x} \in S$, controller sparsity e, training flag $t \in \{\text{TRUE}, \text{FALSE}\}$
Output: action $a \in \mathbb{R}$

1: NODE_CRISP($\sigma(\alpha(\vec{w}_i^T \vec{x} - b_i))) \rightarrow \sigma(\alpha(w_i^k x^k - b_i))$
2: OUTCOME_CRISP($\sigma(\alpha(w_i^k x^k - b_i))) \rightarrow \mathbb{1}(\alpha(w_i^k x^k - b_i) > 0)$
3: $l_d \leftarrow$ INTERPRETABLE_NODE_ROUTING($\vec{x}$)
4: $l_d^* \leftarrow$ ENFORCE_CONTROLLER_SPARSITY(e, l_d)
5: **if** t **then**
6: $\quad a \sim \mathcal{N}(l_d^*(\vec{x}), \gamma_d)$
7: **else**
8: $\quad a \leftarrow l_d^*(\vec{x})$
9: **end if**

4.2.1 DECISION NODE CRISPIFICATION

The NODE_CRISP procedure in Algorithm 1 recasts each decision node to split upon a single dimension of $\vec{x}$. Instead of using a non-differentiable argument max function as in Silva and Gombolay (2021) to determine the most impactful feature dimension, we utilize a softmax function, also known as softargmax (Goodfellow et al., 2016), described by Equation 5. In this equation, we denote the softmax function as $f(\cdot)$, which takes as input a set of class weights and produces class probabilities. Here, $\vec{w}_i$ represents a categorical distribution with class weights, individually denoted by w_i^j, and τ is the temperature, determining the steepness of $f(\cdot)$.

$$f(\vec{w}_i)_k = \frac{\exp\left(\frac{w_i^k}{\tau}\right)}{\sum_{j=1}^{m} \exp\left(\frac{w_i^j}{\tau}\right)} \tag{5}$$

While setting the temperature near-zero would satisfy our objective of producing a one-hot vector, where the outputted class probability of the index of the most impactful feature would be one, this operation can lead to a large variance within the gradients and unstable training. We therefore set the softmax temperature, τ equal to 1, which we find effective empirically, and utilize a differentiable one_hot function, $g(\cdot)$, to produce a one-hot vector with the element associated with the highest-weighted class set to one and all other elements set to zero. We display the procedure for determining the one-hot vector associated with the largest magnitude in Equation 6.

$$\vec{z_i} = g(f(|\vec{w}_i|)) \tag{6}$$

Here, $|\vec{w}_i|$ represents a vector with absolute elements within $\vec{w}_i$. *We maintain differentiability in the procedure described in Equation 6 by utilizing the straight-through trick (Bengio et al., 2013).* This allows us to obtain the desired output, $\vec{z}_i$, a one-hot vector over weights required for the purpose of matching the decision node representation of a decision tree, while maintaining gradients for all weight parameters $\{w_i^j\}_{j=1}^m$ (by treating the gradient with respect to $\vec{z_i}$ as the gradient with respect to $f(|\vec{w}_i|)$). This procedure is further elaborated in Section 4.2.4 and Algorithm 4.

The one-hot encoding $\vec{z_i}$ can then be element-wise multiplied by the original weights to produce a new set of weights with only one active weight, $\vec{z}_i \circ \vec{w}_i \rightarrow \vec{w}'_i$. Accordingly, the decision node representation is transferred from $\sigma(\alpha(\vec{w}_i^T\vec{x} - b_i)) \rightarrow \sigma(\alpha(\vec{w}_i'^T\vec{x} - b_i)) = \sigma(\alpha(w_i^k x^k - b_i))$, where k is the index of the most impactful feature. We provide an algorithm detailing the NODE_CRISP procedure in Algorithm 2, where node crispification recasts each decision node to split upon a single dimension of the input.

Algorithm 2 Node Crispification: NODE_CRISP$(\cdot)$

Input: The original fuzzy decision node $\sigma(\alpha(\vec{w}_i^T\vec{x} - b_i))$, where i is the decision node index, $\vec{w}_i = [w_i^1, w_i^2, ..., w_i^j, w_i^{j+1}, ..., w_i^m]^T$, and m is the number of input features

Output: The intermediate decision node representation $\sigma(\alpha(w_i^k x^k - b_i))$ (see the green box in Figure 2)

1: $\vec{z}_i = \text{DIFF\_ARGMAX}(|\vec{w}_i|)$ (DIFF_ARGMAX$(\cdot)$ displayed in Algorithm 4)
2: $\vec{w}'_i = \vec{z}_i \circ \vec{w}_i$
3: $\sigma(\alpha(w_i^k x^k - b_i)) = \sigma(\alpha(\vec{w}_i'^T\vec{x} - b_i))$

Node crispification takes as input the original fuzzy decision node, $\sigma(\alpha(\vec{w}_i^T\vec{x} - b_i))$, where *all* input features are used in determining the output of decision node i. The output of this function is an

intermediate decision node, $\sigma(\alpha(w_i^k x^k - b_i))$, where the output of decision node i is only determined by *a single* feature, x^k. To perform this transformation, in Line 1, we use the differentiable argument max function (in Algorithm 4) to produce a one-hot vector, $\vec{z}_i$, with the element associated with the most impactful feature set to one and all other elements set to zero. In Line 2, we element-wise multiply the one-hot encoding, $\vec{z}_i$, by the original weights, $\vec{w}_i$, to produce a new set of weights with only one active weight, $\vec{w}'_i$. In Line 3, we show that by multiplying $\vec{x}$ by $\vec{w}'_i$, we can obtain the intermediate decision node $\sigma(\alpha(w_i^k x^k - b_i))$, where k is the index of the most impactful feature (i.e., $k = \arg\max_j(|w_i^j|)$). The transformation to each decision node performed by node crispification is further displayed by the green arrow in Figure 2.

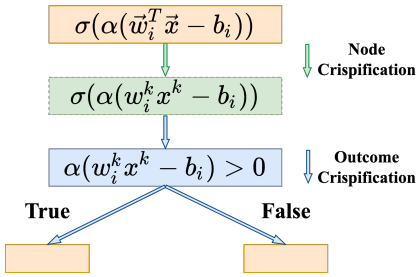


Figure 2: This figure displays the process of differentiable crispification, including node crispification (Algorithm 2) and outcome crispification (Algorithm 3). The node crispification sparsifies the weight vector, $\vec{w}_i$, and chooses the most impactful feature. The outcome crispification enforces a "hard" decision at the node rather than a "soft" decision, so the computation proceeds along one branch. Both operations are differentiable through the use of the straight-through trick.

Below, we conduct a short example detailing our procedure.

Example: Consider we have a two-leaf decision tree (one decision node) with an input feature, $\vec{x} = [2, 3]^T$ with a cardinality of 2 (i.e., $m = 2$), associated weights of $\vec{w_1} = [2, 1]^T$, and a bias term b_1 of 1. The sigmoid steepness, α, is also set equal to 1 for simplicity. It is easily seen that the most impactful weight within the decision node is $w_1^1 = 2$. Utilizing Equation 6, we can compute $\vec{z_1} = [1, 0]^T$. Multiplying $\vec{z_1}$ to the original weights, $\vec{w_1}$, and input feature, $\vec{x}$, subtracting b_1, and scaling by α, we have an crisp decision node $\sigma(2x^1 - 1)$ or $\sigma(3) = 0.95$. Here, 0.95 is the probability that the decision node evaluates to TRUE. We display a depiction of this example in the left-hand side of Figure 3.

4.2.2 DECISION OUTCOME CRISPIFICATION

Here, we describe the second piece of our online differentiable crispification procedure, noted as OUTCOME_CRISP in Algorithm 1. OUTCOME_CRISP translates the outcome of a decision node so that the outcome is a Boolean decision rather than a probability generated via a sigmoid function (i.e., $p = y_i$ for True/Left Branch and $q = 1 - y_i$ for False/Right Branch). We start by creating a soft vector, $\vec{v}_i = [\alpha(w_i^k x^k - b_i), 0]$, for the i^{th} decision node. Placing $\vec{v}_i$ through a softmax operation, we can produce the probability of tracing down the left branch or right. We can then apply the differentiable one-hot function, $g(\cdot)$, to produce a hard decision of whether to branch left or right,

denoted by y_i and described by Equation 7.

$$[y_i, 1 - y_i] = g(f(\vec{v}_i)) \tag{7}$$

Essentially, the decision node will evaluate to TRUE if $\alpha(w_i^k x^k - b_i) > 0$ and right otherwise. This process can be expressed as an indicator function $\mathbb{1}(\alpha(w_i^k x^k - b_i) > 0)$.

We note the procedure of $g(f(\vec{v}_i))$ is highly similar to that in Equation 6, both outputting a one-hot vector, with the former input being the decision node weights, $|\vec{w}_i|$, and the latter input being the soft vector representation of the decision node outcome, $\vec{v}_i$. We provide an algorithm detailing the OUTCOME_CRISP procedure in Algorithm 3, where outcome crispification translates the outcome of a soft decision node to a hard decision node, resulting in a Boolean output from the decision node rather than a set of probabilities.

Algorithm 3 Outcome Crispfication: OUTCOME_CRISP$(\cdot)$

Input: The intermediate decision node $\sigma(\alpha(w_i^k x^k - b_i))$, where i is the decision node index, $k = \arg\max_j(|w_i^j|)$, and w_i^j is the jth element in $\vec{w}_i$

Output: Crisp decision node $\mathbb{1}(\alpha(w_i^k x^k - b_i) > 0)$ (see the blue box in Figure 2)

1: $\vec{v}_i = [\alpha(w_i^k x^k - b_i), 0]$
2: $\vec{z}_i' =$ DIFF_ARGMAX$(\vec{v}_i)$ (DIFF_ARGMAX$(\cdot)$ displayed in Algorithm 4)
3: $\mathbb{1}(\alpha(w_i^k x^k - b_i) > 0) = \vec{z}_i'[0]$

Outcome crispification takes in the intermediate decision node $\sigma(\alpha(w_i^k x^k - b_i))$, which outputs the probability of branching left. The output of OUTCOME_CRISP is the crisp decision node, $\mathbb{1}(\alpha(w_i^k x^k - b_i) > 0)$, a Boolean decision to trace down to the left branch OR right. In Line 1, we construct a soft vector representation of the decision node i's output, $\vec{v}_i$, by concatenating $\alpha(w_i^k x^k - b_i)$ with a 0. In Line 2, we use the differentiable argument max function (in Algorithm 4) to produce a one-hot vector, $\vec{z}_i'$, where the first element represents the Boolean outcome of the decision node. In Line 3, we show that the output of the crisp decision node, $\mathbb{1}(\alpha(w_i^k x^k - b_i) > 0)$, can be obtained by choosing the first element of vector $\vec{z}_i'$ (we use bracket indexing notation here, starting with zero). We further display the transformation performed by outcome crispification by the blue arrows in Figure 2.

Example: Continuing the example in Section 4.2.1, we can take the outputted crisp decision node and generate a vector $\vec{v}_1 = [2x^1 - 1, 0]^T$, or by substituting the input feature, $\vec{v}_1 = [3, 0]^T$. Performing the operations specified in Equation 7, we receive the intermediate output from the softmax $[0.95, 0.05]^T$ (rounded to two decimal places), the one-hot vector $[1, 0]^T$ after performing the one_hot operation, and finally $y_1 = 1$, denoting that the decision-tree should follow the left branch. We display a depiction of this example on the right-hand side of Figure 3.

Conversion to a Simple Form: The above crispification processes produce decision-tracing equal to that of a DT. The node representation can thus be simplified to that of Figure 1 by algebraically reducing each crisp decision node to $x^k > \frac{b_i}{w_i^k}$ (given $\alpha w_i^k > 0$) or $x^k < \frac{b_i}{w_i^k}$ (given $\alpha w_i^k < 0$).

4.2.3 SPARSE LINEAR LEAF CONTROLLERS

After applying the decision node and outcome crispification to all decision nodes and outcomes, the decision can be routed to leaf node (Step 3). This section describes the procedure to translate a linear leaf controller to condition upon e features (ENFORCE_CONTROLLER_SPARSITY procedure in Algorithm 1), enforcing sparsity within the leaf controller and thereby, enhancing ICCT

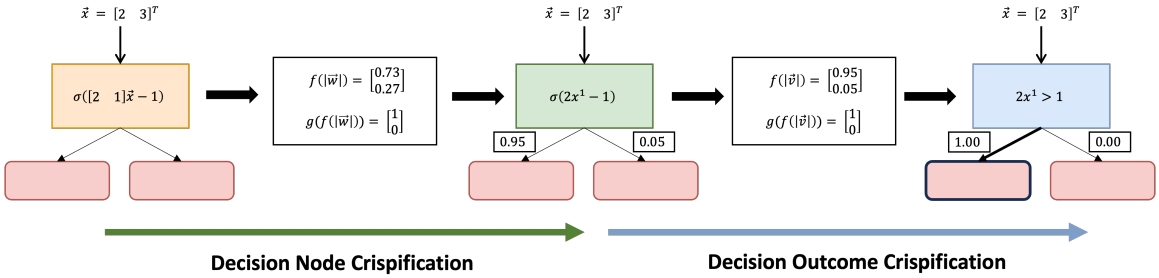


Figure 3: This figure displays the process of decision node crispification and decision outcome crispification across the Examples within Section 4.2.1 and Section 4.2.2.

interpretability. As noted in Section 4.1, our ability to utilize sparse sub-controllers allows us to balance between interpretability and performance. The sparsity of the linear sub-controllers ranges from setting $e = 0$ and maintaining static leaf distributions, where each leaf node contains scalar value representing the mean (i.e., ICCT-static in Section 8), to $e = m$, containing a linear controller parameterized by the entire feature space of $\vec{x}$ (i.e., ICCT-complete in Section 8).

Equation 8 displays the procedure for determining a k-hot encoding, $\vec{u}_d$, that represents the k (or in our case, e) most impactful selection weights within a leaf's linear controller. The k-hot function, denoted by $h(\cdot)$, takes as input a vector of weights and returns an equal-dimensional vector with k elements set to one. The indexes associated with the elements set to one match the k highest-weighted elements within the input feature.

$$\vec{u}_d = h(f(|\vec{\theta}_d|)) \tag{8}$$

Here, $|\vec{\theta}_d|$ represents a vector with absolute elements within the per-leaf selector weights, $\vec{\theta}_d$. As before, we maintain differentiability and formulate a differentiable top-k function in Equation 8 by utilizing the straight-through trick (Bengio et al., 2013) and iteratively applying DIFF_ARGMAX$(\cdot)$ (Algorithm 4) k times. In Equation 9, we transform a fuzzy leaf node, for leaf, l_d (represented in Equation 4), into a sparse linear sub-controller, l_d^*, with the sparse feature selection vector, $\vec{u_d}$, given by Equation 8.

$$l_d^* \triangleq (\vec{u}_d \circ \vec{\beta}_d)^T(\vec{u}_d \circ \vec{x}) + \vec{u}_d^T\vec{\phi}_d \tag{9}$$

A depiction of the sparse sub-models can be seen at the bottom right-hand side of Figure 1, where the sparsity of the sub-controllers, e, is set to 1 and the dimension of the action space is 2.

4.2.4 DIFFERENTIABLE ARGUMENT MAX FUNCTION FOR DIFFERENTIABLE CRISPIFICATION

In this section, we provide a description of the differentiable argument max function.

Algorithm 4 Differentiable Argument Max Function for Crispification: DIFF_ARGMAX$(\cdot)$

Input: Logits $\vec{q}$
Output: One-Hot Vector $\vec{h}$

1: $\vec{h}_{soft} \leftarrow f(\vec{q})$
2: $\vec{h}_{hard} \leftarrow$ ONE_HOT(ARGMAX$(f(\vec{q}))$) # step 1 for function $g(\cdot)$
3: $\vec{h} = \vec{h}_{hard} + \vec{h}_{soft} -$STOP_GRAD$(\vec{h}_{soft})$ # step 2 for function $g(\cdot)$

Similar to (Hafner et al., 2021), we present a function call (in Algorithm 4) that can be utilized to maintain gradients over a non-differentiable argument max operation. The function takes in a set of logits, $\vec{q}$, and applies a softmax operation, denoted by $f(\cdot)$, to output $\vec{h}_{soft}$, as shown in Line 1. In Line 2, the logits are transformed using an argument max followed by a one-hot procedure, causing the removal of gradient information, producing $\vec{h}_{hard}$. In Line 3, we combine $\vec{h}_{soft}$, $\vec{h}_{hard}$, and STOP_GRAD($\vec{h}_{soft}$) to output $\vec{h}$, where STOP_GRAD$(\cdot)$ keeps the values and removes the gradient data of $\vec{h}_{soft}$. The outputted value of $\vec{h}$ is equal to that of $\vec{h}_{hard}$. However, the gradient maintained within $\vec{h}$ is associated with $\vec{h}_{soft}$. Automatic differentiation frameworks can then utilize the outputted term to perform backpropagation. Here, the operations in Line 2 and Line 3 compose function $g(\cdot)$ in Equation 6 and 7.

Summary: In this section, we discuss our novel interpretable reinforcement learning architecture, ICCTs. We present a description of components of ICCTs, including decision nodes and linear leaf controllers, and provide a differentiable crispification procedure allowing for optimization similar to a differentiable decision tree (DDT) while maintaining a forward-propagation process identical to a sparse decision tree. To the best of our knowledge, we present the first truly interpretable tree-based framework for continuous control.

5. Universal Function Approximation

In this section, we provide a proof to show our ICCTs are universal function approximators, that is, can represent any decision surface given enough parameters. Our ICCT architecture consists of successive indicator functions, whose decision point lies among a single dimension of the feature space, followed by a linear controller to determine a continuous control output. For simplicity, we assume below that the leaf nodes contain static distributions. However, maintaining a linear controller at the leaves is more expressive and thus, the result below generalizes directly to ICCTs.

The decision-making of our ICCTs can be decomposed as a sum of products. In Equation 10, we display a computed output for a 4-leaf tree, where decision node outputs, y_i, are given by Equation 1. Here, the sigmoid steepness, α is set to infinity (transforming the sigmoid function into an indicator function) resulting in hard decision points ($y_i \in \{0, 1\}$). Equation 10 shows that the chosen action is determined by computation of probability of reaching a leaf, y, multiplied by static tree weights maintained at the distribution, p.

$$\begin{aligned} ICCT(x) = p_1(y_1 * y_2) + p_2(y_1 * (1 - y_2)) \\ + p_3((1 - y_1) * y_3) + p_4 * ((1 - y_1) * (1 - y_3)) \end{aligned} \tag{10}$$

Equation 10 can be directly simplified into the form of $G(x) = \sum_{j=1}^{N} p_j \sigma(w_j^T x + b_j)$, similar to Equation 1 in (Cybenko, 1989). (Cybenko, 1989) demonstrates that finite combination of fixed, univariate functions can approximate any continuous function. The key difference between our architecture is that our univariate function is an indiator function rather than the commonly used sigmoid function. Below, we provide two lemmas to show that indicator functions fall within the space of univariate functions (Cybenko, 1989).

Lemma 1 *An indicator function is sigmoidal.*

Proof: This follows from the definition of sigmoidal: $\sigma(t) \to 1$ as $t \to \infty$ and $\sigma(t) \to 0$ as $t \to -\infty$.

Lemma 2 *An indicator function is discriminatory.*

Proof: As an indicator function is bounded and measureable, by Lemma 1 of (Cybenko, 1989), it is discriminatory.

Theorem 3 *Let σ be any continuous discriminatory function. ICCTs are universal function approximators, that is, dense in the space of $C(I_n)$. In other words, there is a representation of ICCTs, $I(x)$, for which $|I(x) - f(x)| < \epsilon$ for all $x \in I_n$, for any function, f ($f \in C(I_n)$), where $C(I_n)$ denotes the codomain of an n-dimensional unit cube, I_n.*

Proof As the propositional conditions hold for Theorem 1 in (Cybenko, 1989), the result that ICCTs are dense in $C(I_n)$ directly follows. We note that as the indicator function jump-continuous, we refer readers to (Selmic and Lewis, 2002) whom extend UFA for $G(x) = \sum_{j=1}^{N} p_j \sigma(w_j^T x + b_j)$ to the case when σ is jump-continuous. As ICCTs are dense in $C(I_n)$, ICCTs are universal function approximators. ■

6. Model Robustness Verification

A desirable property for a controller is to be able to verify the policy, ensuring the controller outputs desirable values for a set of inputs. This often translates to answering the following question:

- *For a range of input features, what is the range of output values that can be expected?*

By answering this question, engineers and end-users can attain key insights into a policy's decision-making behavior and make guarantees about its behavior. Utilizing autonomous driving as an example, an engineer may want to verify that if a human is detected within 5 meters, the acceleration of the vehicle is never above $5m/s^{-2}$. Verification of policies is vital in creating models that are safe and can help ensure that models accurately perform the purpose they are designed for.

In this section, we analyze the time complexity of verifying an ICCT. Following Chen et al. (2019), we formalize the problem of *robustness verification* as follows: Consider a regression model: $f : \mathbb{R}^d \rightarrow \mathbb{R}$, where d is the dimension of the input features and the output is a real-valued scalar. In Chen et al. (2019), for input, x, the minimal adversarial perturbation is defined by Equation 11, where $y = f(x)$ is the expected controller output value. The solution to this equation determines the minimum input perturbation required to have the controller output a value different from the expected value, y.

$$r^* = \min_{\lambda} ||\lambda||_\infty \text{ s.t. } f(x + \lambda) \neq y \tag{11}$$

As we are concerned with *continuous* control policies, where a slight perturbation on the input may cause a slight perturbation on the controller output, we instead generalize y to an expected output range, $y = \{a, b\}$, and search for an input perturbation that causes y to fall out of the specified range, i.e., $f(x + \lambda) \notin \{a, b\}$. Following our autonomous driving example, finding the minimum adversarial perturbation allows an engineer to understand what input deviations may result in unsafe or undesirable controller outputs. Thus below, we present a discussion of the time complexity associated with solving Equation 11 of neural network models and our ICCTs. *Positively, for our ICCTs, determining the minimal input perturbation can be done in linear time.*

Due to complex nonlinearities within neural network architectures, small input perturbations can lead to large changes in predicted values (Szegedy et al., 2013; Carlini and Wagner, 2016), making it intractable to perform verification. Often, verification is only possible if the neural network architecture follows certain desiderata (Katz et al., 2017) or by utilizing convex relaxations of nonlinear activation functions (Salman et al., 2019). Even so, such verification, in the best case, is only NP-Complete.

ICCTs, on the other hand, can be verified in linear time. Here, we first present an analysis showing ICCT-static (ICCT maintaining static Gaussian distributions at each leaf) can be verified in linear time. We then extend this proof to ICCT-complete, where linear controllers parameterized by the input features are maintained at each leaf. For simplicity, we assume that the output value of our controller is solely determined by the mean value within the leaf distribution of our ICCTs, similar to how ICCTs are deployed during runtime.

Assume we have a decision tree that has n decision nodes and $n + 1$ leaf nodes. For each decision node i, we define a variable, t_i, representing which feature is activated within the decision node, and a variable, η_i, representing the threshold maintained with the decision node (which is equal to $\frac{b_i}{w_i}$). Depending on the outcome of the decision node, the computation will further proceed to either the left or right child until arrival at a leaf node, where the leaf node contains a scalar value representing the Gaussian mean.

Following Chen et al. (2019), the key of the proof is to split the d-dimensional input space to $n + 1$ hyperspaces corresponding to leaf nodes, such that any input will result in falling into one and only one leaf node's hyperspace. This can be done by traversing the entire tree and computing bounding boxes via a depth-first search. By definition, all input variables will reach the root node, resulting in a root node box represented by the Cartesian product $B = [-\infty, \infty] \times \cdots [-\infty, \infty]$, of cardinality d. Each child's box can be obtained by splitting one interval from the parent box based on the split condition (the variable selected, t_i, and threshold, η_i). This process can be completed until the entire tree is traversed (via a depth-first search fashion), resulting in a time complexity of $O(nd)$, where n is the number of nodes and d is the cardinality of the input space.

The distance from an input x to a leaf node's region can be written as a vector, $\epsilon(x, B^i) \in \mathbb{R}^d$, defined by the Equation 12, where l_t^i, r_t^i, represent the upper and lower bound of a node's Cartesian product on dimension t for leaf i, respectively.

$$\epsilon(x, B^i)_t = \begin{cases} 0 & \text{if } i_t \in (l_t^i, r_t^i) \\ x_t - r_t^i & \text{if } x_t > r_t^i \\ l_t^i - x_t & \text{if } x_t \leq l_t^i \end{cases} \tag{12}$$

Thus, the minimal distortion required to result in an incorrect output value can be obtained by Equation 13, where $y = \{a, b\}$ is the output range desired, and v_i is the value for leaf node i. Intuitively, Equation 13 finds the minimum perturbation on a feature, t, that leads to a leaf node with associated output outside of the desired output range.

$$r^* = \min_{i:v_i \notin y, t \in [d]} \epsilon(x, B^i)_t \tag{13}$$

The time complexity of the verification algorithm for ICCT robustness is $O(nd)$ due to the traversal of the tree, the combination of bounding boxes, and the minimal perturbation finding in Equation 13 by iterating over all leaf node and all feature dimensions. As stated above, this results in addressing the *decision problem of robustness verification* in linear time.

When extending to ICCTs with linear controllers at the leaves, we can utilize the same formalism to obtain the bounding boxes represented by Cartesian products at each leaf. However, as each leaf controller depends on input variables, the range of outputs would require extra computation based on the bounding boxes and linear controller parameters. Because of the monotony of the linear controllers with respect to each input feature, the computation is still $O(d)$ for each leaf node, and therefore we can still achieve the overall $O(nd)$ time complexity for the *robustness verification*.

7. Environments

Here, we provide short descriptions across six domains used in our extensive evaluation. We start with two common continuous control problems, Inverted Pendulum, and Lunar Lander, provided by OpenAI Gym (Brockman et al., 2016). We then test across four autonomous driving scenarios: Lane-Keeping provided by Leurent (2018) and Single-Lane Ring Network, Multi-Lane Ring Network, and Figure-8 Network all provided by the Flow deep reinforcement learning framework for mixed autonomy traffic scenarios (Wu et al., 2017).
Inverted Pendulum: In Inverted Pendulum (Todorov et al., 2012), a control policy must apply throttle (ranging from +3 to move left to -3 to move right) to balance a pole. The observation includes the cart position, velocity, pole angle, and pole angular velocity.

- *Lunar Lander:* In Lunar Lander (Parberry, 2017; Brockman et al., 2016), a policy must throttle the main engine and side engine thrusters for a lander to land on a specified landing pad. The observation is 8-dimensional, including the lander's current position, linear velocity, tilt, angular velocity, and information about ground contact. The continuous action space is two-dimensional, with the first dimension controlling the main engine thruster and the second controlling the side engine thrusters.
- *Lane-Keeping (Leurent, 2018):* A control policy must control a vehicle's steering angle to stay within a curving lane. The observation is 12-dimensional, which consists of lane information and the vehicle's lateral position, heading, lateral speed, yaw rate, and linear, lateral, and angular velocity. The action is the steering angle to control the vehicle.
- *Flow Single-Lane Ring Network (Wu et al., 2017):* A control policy must apply acceleration commands to a vehicle agent to stabilize traffic flow consisting of 21 other human-driven (synthetic) vehicles. The observation includes the world position and velocity of all vehicles.
- *Flow Multi-Lane Ring Network (Wu et al., 2017):* A control policy must apply acceleration and lane-changing commands to an ego vehicle to stabilize the flow of noisy traffic flow across multiple lanes. The observation includes the world position and velocity of all vehicles.
- *Flow Figure-8 Network (Wu et al., 2017):* A control policy must apply acceleration to a vehicle to stabilize the flow in a Figure-8 network (contains a section where the vehicles must cross paths at the center of the 8), requiring the policy to adapt its control input to create a stable flow through this section. The observation is the world position and velocity of all vehicles.

8. Results

In this section, we present the set of baselines we test our model, ICCTs, against. Then, we report the results of our approach versus these baselines across the six continuous control domains, as shown

Worst to Best:

Method	Common Continuous Control Problems		Autonomous Driving Problems			
	Inverted Pendulum	Lunar Lander	Lane Keeping	Single-Lane Ring	Multi-Lane Ring	Figure-8
DT	155.0 ± 0.9 256 leaves (766 params)	-285.5 ± 15.6 256 leaves (1022 params)	-359.0 ± 11.0 256 leaves (766 params)	**123.2 ± 0.03** 32 leaves (94 params)	503.2 ± 24.8 256 leaves (1022 params)	831.1 ± 1.1 256 leaves (766 params)
DT w\ DAgger	776.6 ± 54.2 32 leaves (94 params)	184.7 ± 17.3 32 leaves (126 params)	**395.2 ± 13.8** 16 leaves (46 params)	121.5 ± 0.01 16 leaves (46 params)	1249.4 ± 3.4 31 leaves (122 params)	**1113.8 ± 9.5** 16 leaves (46 params)
CDDT-Crisp	5.0 ± 0.0 2 leaves (5 params)	-451.6 ± 97.3 8 leaves (37 params)	-43526.0 ± 15905.0 16 leaves (61 params)	68.1 ± 18.7 16 leaves (61 params)	664.5 ± 192.6 16 leaves (77 params)	322.9 ± 47.1 16 leaves (61 params)
ICCT-static	**984.0 ± 10.4** 32 leaves (125 params)	**192.4 ± 10.7** 32 leaves (157 params)	374.2 ± 55.8 16 leaves (61 params)	120.5 ± 0.5 16 leaves (61 params)	**1271.7 ± 4.1** 16 leaves (77 params)	1003.8 ± 27.2 16 leaves (61 params)
ICCT-1-feature	1000.0 ± 0.0 8 leaves (45 params)	190.1 ± 13.7 8 leaves (69 params)	437.6 ± 7.0 16 leaves (93 params)	121.6 ± 0.5 16 leaves (93 params)	1269.6 ± 10.7 16 leaves (141 params)	1072.4 ± 37.1 16 leaves (93 params)
ICCT-2-feature	1000.0 ± 0.0 4 leaves (29 params)	258.4 ± 7.0 8 leaves (101 params)	458.5 ± 6.3 16 leaves (125 params)	**121.9 ± 0.5** 16 leaves (125 params)	1280.4 ± 7.3 16 leaves (205 params)	**1088.6 ± 21.6** 16 leaves (125 params)
ICCT-3-feature	**1000.0 ± 0.0** 2 leaves (17 params)	275.8 ± 1.5 8 leaves (133 params)	448.8 ± 3.0 16 leaves (157 params)	120.8 ± 0.5 16 leaves (157 params)	**1280.8 ± 7.7** 16 leaves (269 params)	1048.7 ± 46.7 16 leaves (157 params)
ICCT-L1-sparse	1000.0 ± 0.0 4 leaves (29 params)	265.2 ± 4.3 8 leaves (165 params)	465.5 ± 4.3 16 leaves (253 params)	121.5 ± 0.3 16 leaves (765 params)	1275.3 ± 6.7 16 leaves (2189 params)	993.2 ± 14.6 16 leaves (509 params)
ICCT-complete	1000.0 ± 0.0 2 leaves (13 params)	**300.5 ± 1.2** 8 leaves (165 params)	**476.6 ± 3.1** 16 leaves (253 params)	120.7 ± 0.5 16 leaves (765 params)	1248.6 ± 3.6 16 leaves (2189 params)	994.1 ± 29.1 16 leaves (509 params)
CDDT-controllers Crisp	84.0 ± 10.4 2 leaves (13 params)	-126.6 ± 53.5 8 leaves (165 params)	-39826.4 ± 21230.0 16 leaves (253 params)	97.9 ± 12.0 16 leaves (765 params)	639.62 ± 160.4 16 leaves (2189 params)	245.5 ± 48.5 16 leaves (509 params)
MLP-Lower	1000.0 ± 0.0 79 params	231.6 ± 49.8 110 params	474.7 ± 5.8 127 params	**121.8 ± 0.6** 151 params	646.4 ± 151.2 221 params	868.4 ± 100.9 103 params
MLP-Upper	1000.0 ± 0.0 121 params	288.7 ± 2.8 222 params	467.9 ± 8.5 407 params	121.8 ± 0.3 709 params	1239.5 ± 4.2 3266 params	1077.7 ± 31.1 1021 params
MLP-Max	1000.0 ± 0.0 67329 params	**298.5 ± 0.7** 68610 params	**478.2 ± 6.7** 69377 params	121.7 ± 0.4 77569 params	1011.9 ± 141.3 83458 params	**1104.3 ± 9.4** 73473 params
CDDT	**1000.0 ± 0.0** 2 leaves (8 params)	226.4 ± 44.5 8 leaves (86 params)	464.7 ± 5.4 16 leaves (226 params)	120.9 ± 0.5 16 leaves (706 params)	**1248.0 ± 6.4** 16 leaves (1036 params)	1033.2 ± 24.1 16 leaves (466 params)
CDDT-controllers	1000.0 ± 0.0 2 leaves (16 params)	289.0 ± 2.4 8 leaves (214 params)	469.7 ± 11.1 16 leaves (418 params)	120.1 ± 0.3 16 leaves (1410 params)	1243.8 ± 3.6 16 leaves (2092 params)	1010.9 ± 25.7 16 leaves (914 params)

Table 1: In this table, we display the results of our evaluation. For each evaluation, we report the mean (± standard error) and the complexity of the model required to generate such a result. Our table is broken into three segments, the first containing equally interpretable approaches that utilize static distributions at their leaves. The second segment contains interpretable approaches that maintain linear controllers at their leaves. The ordering of methods denotes the relative interpretability. The third segments displays black-box approaches. We bold the highest-performing method in each segment, and break ties in performance by model complexity. We color table elements in association with the number of parameters and performance. Reddish colors relate to a larger number of policy parameters and lower average reward.

in Table 1. All presented results are across five random seeds, and all differentiable frameworks are trained via SAC (Haarnoja et al., 2018). Each tree-based framework is trained while maximizing performance and minimizing the complexity required to represent such a policy, thereby emphasizing interpretability. We release our codebase at https://github.com/CORE-Robotics-Lab/ICCT.

8.1 Baselines

We provide a list of baselines alongside abbreviations used for reference and brief definitions below. We compare against interpretable models, black-box models, and models that can be converted post-hoc into an interpretable form. For each method, we also include details regarding the number of active parameters utilized in each model (a surrogate measure for model complexity). We briefly list the following notations for an easier understanding of the computation of the number of parameters for each model discussed below. The number of leaf nodes is N_l (the number of decision nodes is $N_l - 1$), the dimension of the observation space is m, the number of active features within the leaf controllers is e, and the dimension of the action space is d_a. The calculated number of parameters is denoted as N_p for each model. Our approach, ICCT-e-feature, has a number of parameters of $N_p = 3(N_l - 1) + (2e + 1)d_a N_l = (2ed_a + d_a + 3)N_l - 3$.

- Continuous DDTs (CDDT): We translate the framework of Silva and Gombolay (2021) to function with continuous action-spaces by modifying the leaf nodes to represent static probability distributions. Here, $N_p = (m+2)(N_l-1)+d_a N_l = (d_a+m+2)N_l - m - 2$. When converted into an interpretable form post-hoc, this approach is reported as CDDT-crisp which has a number of parameters, $N_p = 3(N_l - 1) + d_a N_l = (3 + d_a)N_l - 3$.

- Continuous DDTs with controllers (CDDT-controllers): We modify CDDT leaf nodes to utilize linear controllers rather than static distributions. Here, $N_p = (m+2)(N_l-1)+(m+1)d_a N_l = (md_a + d_a + m + 2)N_l - m - 2$. When converted into an interpretable form post-hoc, this approach is reported as CDDT-controllers Crisp that has $N_p = 3(N_l - 1) + (m + 1)d_a N_l = (md_a + d_a + 3)N_l - 3$.

- ICCTs with static leaf distributions (ICCT-static): We modify the leaf architecture of our ICCTs to utilize static distributions for each leaf (i.e., set $e = 0$). Comparing ICCT and ICCT-static displays the effectiveness of the addition of sparse linear sub-controllers. Here, the number of parameters can be computed as $N_p = 3(N_l - 1) + d_a N_l = (3 + d_a)N_l - 3$.

- ICCT with complete linear sub-controllers (ICCT-complete): We allow the leaf controllers to maintain weights over all features (no sparsity enforced, i.e., $e = m$). Comparing ICCT-complete and CDDT-controllers displays the effectiveness of the proposed differentiable crispification procedure. Here, the number of parameters can be computed as $N_p = 3(N_l - 1) + (m + 1)d_a N_l = (md_a + d_a + 3)N_l - 3$.

- ICCT with L1-regularized controllers (ICCT-L1-sparse): We achieve sparsity via L1-regularization applied to ICCT-complete rather than enforcing sparsity directly via the Enforce_Controller_Sparsity procedure. While this baseline produces sparse sub-controllers, there are drawbacks limiting its interpretability. L1-regularization enforces weights to be near zero rather than exactly zero. These small weights must be represented within leaf nodes, and thus, the interpretability of the resulting model is limited. Here, the number of parameters can be computed as $N_p = 3(N_l - 1) + (m + 1)d_a N_l = (md_a + d_a + 3)N_l - 3$.

- Multi-layer Perceptron (MLP): We maintain three variants of an MLP. The first (MLP-Max) contains a very large number of parameters, typically utilized in continuous control domains. The second (MLP-Upper) maintains approximately the same number of parameters of our ICCTs with sparse leaf controllers during training, including all inactive parameters. The last (MLP-Lower) maintains approximately the same number of *active* parameters as our ICCTs with sparse leaf controllers. The number of parameters of MLP depends on the size of the network, including all weights and bias parameters.

- Decision Tree (DT): We train a DT via CART (Loh, 2011) on state-action pairs generated from MLP-Max. This baseline represents policy distillation from a high-performance black-box policy to an interpretable model. Here, the number of parameters can be computed as $N_p = 2(N_l - 1) + d_a N_l = (2 + d_a)N_l - 2$.

- DT w\ DAgger: We utilize the DAgger imitation learning algorithm (Ross et al., 2011) to train a DT to mimic the MLP-Max policy. The number of parameters can be computed as in the DT baseline above.

8.2 Discussion

We present the results of our trained policies in Table 1. We provide the performance of each method alongside the associated complexity of each benchmark in Table 1 across three sections, with the top section representing interpretable approaches that maintain static distributions at their leaves, the middle section containing interpretable approaches that maintain linear controllers at their leaves, and the bottom section containing black-box methods.

Static Leaf Distributions (Top): The frameworks of DT, DT w\ DAgger, CDDT-Crisp, and ICCT-static maintain similar representations and are equal in terms of interpretability given that the approaches have the same depth. We see that across three of the six control domains, ICCT-static is able to widely outperform both the DT and CDDT-Crisp models. In the remaining three domains, ICCT-static outperforms CDDT-Crisp by a large margin and achieves competitive performance compared to DTs, even without access to a superior expert policy.

Linear Controller Leaf (Middle): Here, we rank frameworks (top-down) by their relative interpretability. As the sparsity of the sub-controller decreases, the interpretability diminishes. We see that most approaches are able to achieve the maximum performance in the simple domain of Inverted Pendulum. However, CDDT-controllers-crisp encounters an inconsistency issue from the crispification procedure of (Silva and Gombolay, 2021; Paleja et al., 2020) and achieves very low performance. In regards to interpretability-performance tradeoff, in Inverted Pendulum, we see that as sparsity increases within the sub-controller, a lower-depth ICCT can be used to achieve a equally high-performing policy. We note that across all domains, we do not find such a linear relationship. We provide additional results within Section 10 that provide deeper insight into the interpretability-performance tradeoff.

Black-Box Approaches (Bottom): MLP-based approaches and fuzzy DDTs are not interpretable. While the associated approaches perform well across many of the six domains, the lack of interpretability limits the utility of such frameworks in real-world applications such as autonomous driving. We see that in half the domains, highly-parameterized architectures with over 65,000 parameters are required to learn effective policies.

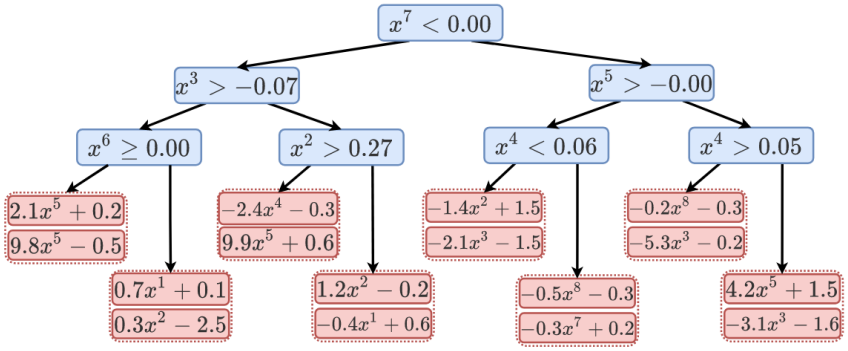


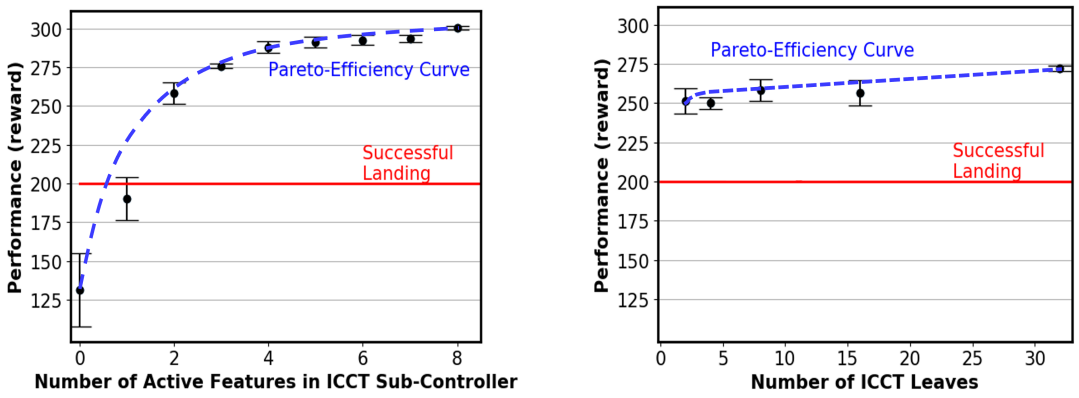
Figure 4: A Learned ICCT in Lunar Lander

Comparison Across All Approaches: We see that across all continuous control domains, CDDT-Crisp and CDDT-controllers Crisp typically are the lowest-performing models. This displays the drawbacks of the crispification procedure of (Silva and Gombolay, 2021; Paleja et al., 2020) and the resultant performance inconsistency. Comparing our ICCTs to black-box models, we see that in all domains, we parity or outperform deep highly-parameterized models in performance while reducing the number of parameters required by orders of magnitude. In the difficult Multi-Lane Ring scenario, we see that we can outperform MLPs by 33% on average while achieving a 300x-600x reduction in the number of policy parameters required.

Overall, we find strong evidence for our Interpretable Continuous Control Trees, displaying and validating the ability to at least parity black-box approaches while maintaining high interpretability. Our novel architecture and training procedure provide a strong step towards providing solutions for two grand challenges in interpretableML: (1) Optimizing sparse logical models such as DTs and (10) Interpretable RL.

9. Qualitative Exposition of ICCT Interpretability

Here, we provide a display of the utility and interpretability of a learned ICCT model. In Figure 4, we present our learned ICCT model in Lunar Lander, rounding each element to two decimal places for brevity. The displayed figure is an ICCT-1-feature model (i.e., only one active feature within the sparse sub-controller). The 8-dimensional input in Lunar Lander is composed of position (x^1,x^2), velocity (x^3,x^4), angle (x^5), angular velocity (x^6), left (x^7) and right (x^8) lander leg-to-ground contact. The action space is two-dimensional: the first (dictated by the top of each pair of the red-colored leaves) controls the main engine thrust, and the second (bottom) controls the net thrust for the side-facing engines. The tree can be interpreted as follows: taking the leftmost path as an example, if the left leg is not touching the ground, the horizontal velocity is greater than -0.07 m/s, and the angular velocity is greater than 0.00 rad/s, then the main engine action is $2.1 * (\text{the lander angle}) + 0.2$, and the side engine action is $9.8 * (\text{the lander angle}) - 0.5$. Such a tree has several use cases: 1) An engineer/developer may pick certain edge cases and verify the behavior of the lander. Performing robustness verification on our ICCTs can be done in linear time (see Section 6), while DNN verification is NP-complete (Katz et al., 2017). 2) An engineer can evaluate the decision-making in the tree and detect anomalies. Furthermore, there are hands-on use-cases of such a model, such as threshold editing (directly modifying nodes to increase affordances), etc.



(a) Performance vs. Number of Controller Features

(b) Performance vs. Number of ICCT Leaves

Figure 5: In this figure, we display the interpretability-performance tradeoff of our ICCTs with respect to the number of active features within our linear sub-controllers (Figure 5a) and the number of tree leaves (Figure 5b) in Lunar Lander. Within each figure, we display the approximate Pareto-Efficiency Curve and denote the reward required for a successful lunar landing as defined by (Brockman et al., 2016).

10. Ablation: Interpretability-Performance Tradeoff

Here, we provide an ablation study over how ICCT performance changes with respect to the number of active features within our linear sub-controllers and depth of the learned policies. Lakkaraju et al. (2016) states that decision trees are interpretable because of their simplicity and that there is a cognitive limit on how complex a model can be while still being understandable. Accordingly, for our ICCTs to maximize interpretability, we emphasize the sparsity of our sub-controllers and attempt to minimize the depth of our ICCTs. Here, we present a deeper analysis by displaying the performance of our ICCTs while varying the number of active features, e, from ICCT-static to ICCT-complete (Figure 5a), and varying the number of leaves maintained within the ICCT from $N_l = 2$ to $N_l = 32$. We conduct our ablation study within Lunar Lander.

In Figure 5a, we show how the performance of our ICCTs changes as a function of active features in the Sub-Controller. Here, we fix the number of ICCT leaves to 8. We see that as the number of active features increase, the performance also increases. However, there is a tradeoff in interpretability. As greater than 200 reward is considered successful in this domain, a domain expert may determine a point on the Pareto-Efficiency curve that maximizes the interpretability-performance tradeoff. In Figure 5b, we show how the performance of our ICCTs changes as a function of tree depth while fixing the number of active features in the ICCT sub-controller to two. We see a similar, albeit weaker, relationship between performance and interpretability. As model complexity increases, there is a slight gain in performance and a large decrease in interpretability. The Pareto-Efficiency curve provides insight into the interpretability-performance tradeoff for ICCT tree depth.

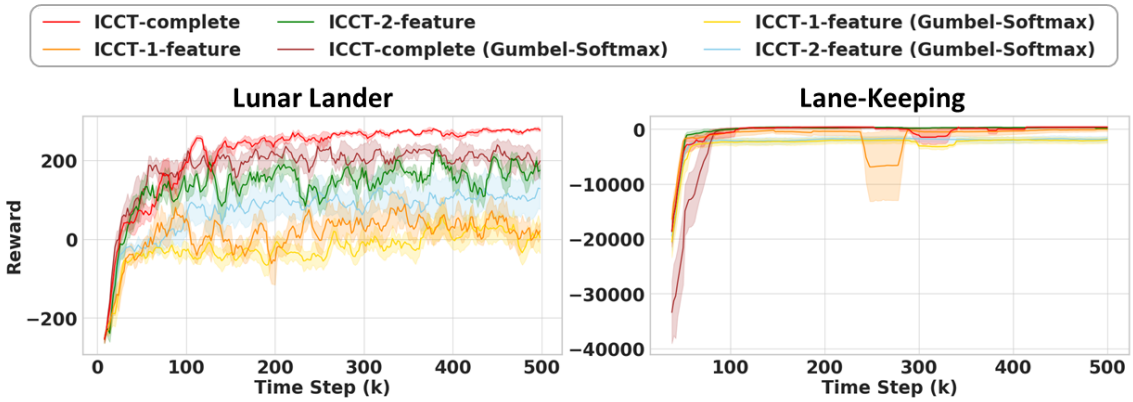


Figure 6: This figure displays the average running rollout rewards of six methods for the ablation study during training. The results are averaged over 5 seeds, and the shadow region represents the standard error.

Method	Lunar Lander	Lane-Keeping
ICCT-complete	300.5 ± 1.2	476.6 ± 3.1
ICCT-complete (Gumbel-Softmax)	276.7 ± 7.0	412.6 ± 31.3
ICCT-complete (Gumbel-Softmax, Crisp)	239.0 ± 18.9	309.1 ± 94.6
ICCT-1-feature	190.1 ± 13.7	437.6 ± 7.0
ICCT-1-feature (Gumbel-Softmax)	113.2 ± 43.1	-853.4 ± 333.2
ICCT-1-feature (Gumbel-Softmax, Crisp)	-20.1 ± 50.0	-658.114 ± 345.3
ICCT-2-feature	258.4 ± 7.0	458.5 ± 6.3
ICCT-2-feature (Gumbel-Softmax)	161.7 ± 54.8	-560.6 ± 251.6
ICCT-2-feature (Gumbel-Softmax, Crisp)	62.3 ± 82.2	-945.0 ± 331.0

Table 2: This table shows a performance comparison between ICCTs utilizing our proposed differentiable argument max function (DIFF_ARGMAX$(\cdot)$ in Algorithm 4), and a variant of ICCTs utilizing the Gumbel-Softmax function (fuzzy and crisp). Across each approach, we present our findings across Lunar Lander and Lane-Keeping and include ICCTs with fully parameterized sub-controllers (ICCT-complete) and sparse sub-controllers.

11. Ablation: Differentiable Argument Max and Gumbel-Softmax

In this section, we provide an ablation study on the differentiable operator used in ICCTs to perform decision node crispification, perform decision outcome crispification, and enforce sub-controller sparsity. Here, we substitute the Softmax function with a Gumbel-Softmax (Jang et al., 2017) function, a widely-used differentiable approximate sampling mechanism for categorical variables. To allow ICCTs to utilize the Gumbel-Softmax function, as opposed to DIFF_ARGMAX$(\cdot)$, we modify the original Softmax function, f, introduced by Equation 5, to f' defined in Equation 14. Here, $\vec{w_i}$ is an m-dimensional vector, $[w_i^1, \cdots, w_i^m]^T$, and $\{g_i^j\}_{j=1}^m$ are i.i.d samples from a Gumbel$(0, 1)$

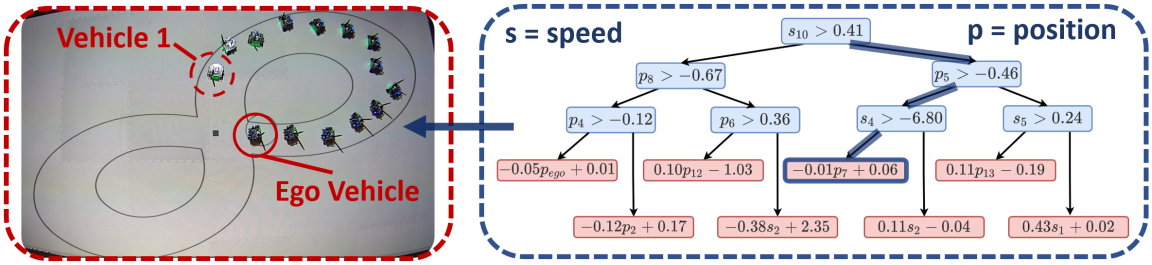


Figure 7: In this figure, we display our ICCTs controlling a vehicle in a 14-car physical robot demonstration within a Figure-8 traffic scenario. Active nodes and edges are highlighted by the right online visualization, where s_i represents the speed of vehicle i, and p_i represents the position of vehicle i. We include a full video, including an enlarged display of our ICCT at https://sites.google.com/view/icctree.

distribution (Jang et al., 2017).

$$f'(\vec{w}_i)_k = \frac{\exp\left(\frac{w_i^k + g_i^k}{\tau}\right)}{\sum_j^m \exp\left(\frac{w_i^j + g_i^j}{\tau}\right)} \tag{14}$$

In Table 2, we compare the performance of ICCT-complete, ICCT-1-feature, and ICCT-2-feature to their variants using Gumbel-Softmax in Lunar Lander and Lane-Keeping. All the methods and their corresponding variants are trained using the same hyperparameters. From the results shown in Figure 6 and Table 2, we find that the addition of Gumbel noise reduces performance by a wide margin. Furthermore, comparing crisp ICCTs utilizing Gumbel-Softmax to ICCTs utilizing Gumbel-Softmax, we see that due to the sampling procedure within the Gumbel-Softmax, an inconsistency issue arises between fuzzy and crisp performance. Such results support our design choice of the differentiable argument max function over the Gumbel-Softmax.

12. Physical Robot Demonstration

Here, we demonstrate our algorithm with physical robots in a 14-car figure-8 driving scenario and provide an online, easy-to-inspect visualization of our ICCTs, which controls the ego vehicle. We utilize the Robotarium, a remotely accessible swarm robotic research platform (Wilson et al., 2020), to demonstrate the learned ICCT policy. The demonstration displays the feasibility of ego vehicle behavior produced by our ICCT policy and provides an online visualization of our ICCTs. A frame taken from the demonstrated behavior is displayed in Figure 7. We provide a complete video of the demonstrated behavior and the online visualization of the control policy at https://sites.google.com/view/icctree.

Figure 8: This figure illustrates the I-94 domain. The red arrows denote the traffic flow directions. There are four traffic inflows: one highway inflow (leftmost) and three ramp inflows. There are also four traffic outflows: highway outflow (rightmost) and three ramp outflows.

Metrics	ICCT					MLP		
	2 leaves 1-feature	2 leaves 2-feature	4 leaves 1-feature	4 leaves 2-feature	8 leaves 1-feature	Lower	Upper	Max
Environment Returns	878.72	899.65	858.62	**910.93**	871.80	907.11	901.29	897.13
# number of parameters	15	23	33	49	69	46	862	71426
# collisions	0	0	0	0	0	0	0	1
# hard-brakes	0	0	0	0	0	0	0	2
# unsafe lane changes	14	7	30	**0**	9	13	**0**	3

Table 3: This table shows our findings within the I-94 domain. Environment returns are the average of ten evaluation episodes after training has been completed. The remaining metrics are computed through a summation over occurrences of the respective phenomena across the ten evaluation episodes.

13. Case Studies on Complex Driving Domain Grounded in Realistic Lane Geometries

We extend our analysis of the ICCT with experiments on two realistic driving domains modeled after 1) the I-94 highway in Michigan and 2) the I-280 highway in California, and verify ICCT's ability to drive safely on crowded highways.

13.1 The I-94 Domain

The I-94 domain is built with the SUMO traffic simulator (Lopez et al., 2018) and modeled off the Interstate Highway 94 from Exit 190 to Exit 192. As illustrated in Figure 8, the I-94 domain consists of three ramp entries and three ramp exits. The task of the ego vehicle is to enter from the first ramp entry and exit from the last ramp exit and do so as quickly as possible while being safe.

The state space of I-94 is 19-dimensional and consists of the ego longitudinal location (i.e., progress of the driving), the ego speed, the ego latitudinal location (i.e., lane information), and the surrounding vehicles' status. We define "surrounding" vehicles as the leading cars and following cars on each lane with respect to the ego car, and for each surrounding vehicle, we provide the distance to the ego car as well as its velocity. As there are four lanes on I-94, the surrounding vehicles' information is 16-dimensional. The action space of the RL agent is two-dimensional, which controls the ego vehicle's acceleration and steering angle.

As we would like to ensure both the performance and safety of the RL agent, we design a 6-component reward function, $R_{\text{I-94}} = \sum_{i=1}^{6} R_i$. Here, R_1 represents the positive reward for ego speed, R_2 represents the negative constant time penalty, R_3 denotes the negative penalty for too-small headways, R_4 provides the negative penalty for wrong routing (i.e., the ego vehicle does not exit properly via the last ramp), R_5 encodes the negative penalty for emergency braking behaviors, and R_6 is the negative penalty for unsafe lane changes. Intuitively, R_1 and R_2 motivate the ego car to move faster, R_4 helps the ego car to follow the desired route, and R_3, R_5, and R_6 encourage the RL vehicle to be safe by keeping headways and avoid dangerous behaviors.

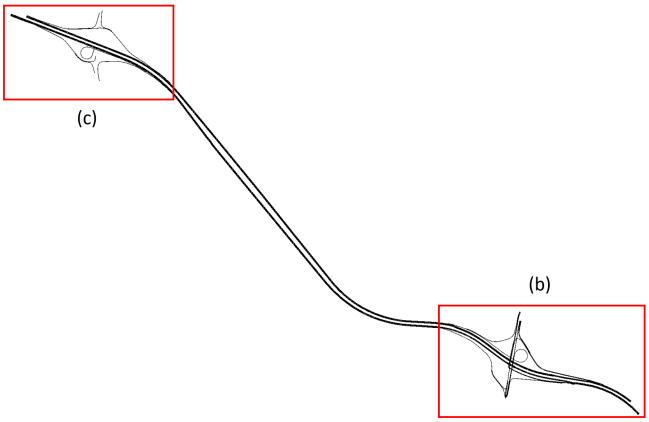
13.2 I-94 Results

We summarize the results of the I-94 domain in Table 3. We compare five sizes of ICCTs and three sizes of MLPs (as introduced in Section 8.1) across four metrics. Firstly, we find that the ICCT and MLP can achieve similar performance on the environment returns metric, with ICCT (4 leaves, 2-feature) variant achieving the highest returns. The second and third metrics we consider are the number of collisions and the number of hard brakes in the evaluation of 10 episodes after training the policies. As our driving simulator, SUMO, always prevents an actual collision by applying a high deceleration, we regard a deceleration that is higher than $10m \cdot s^{-2}$ as a collision, and a deceleration that is between $5m \cdot s^{-2}$ and $10m \cdot s^{-2}$ as a hard brake. We observe that most ICCT and MLP models achieve zero collisions and hard brakes, with the exception of MLP-Max. We hypothesize that MLP-Max may overfit to the training data and generate undesired large-model artifacts, showing the brittleness of large MLPs in general. The last metric we consider is the number of unsafe lane changes, which counts the number of instances that the ego car tries to merge onto a lane with too small of a headway or tailway, or tries to change to a non-existing lane (e.g., attempts to change to the left lane on the left-most lane). We observe that the best-performing ICCT model with 4 leaves of 2 features achieves zero unsafe lane changes and higher rewards than the best MLP model. Comparing ICCT-2-feature with 4 leaves and MLP-Upper, although the two models have similar performance on all metrics, the ICCT model maintains 94% fewer parameters compared with the MLP-Upper model. Thus, we conclude that in the case study of the I-94 domain, ICCT models were able to learn to successfully accomplish the task, meeting both performance and safety objectives while being highly parameter-efficient.

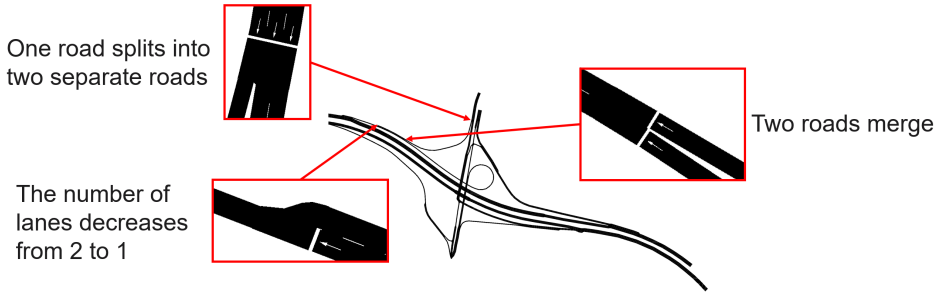
13.3 The I-280 Domain

The I-280 domain is a complex environment modeled off the Interstate Highway 280 around the Palo Alto area in California[3]. An overview of the I-280 Domain is shown in Figure 9a. The ego vehicle is tasked to join the highway from the ramp and then exit the environment at the end of the highway. As shown in Figure 9b and Figure 9c, the I-280 domain is more challenging than the I-94 domain due to the multiple road geometries encountered across a trajectory, including road splits and the introduction and disappearance of lanes. The state space for I-280 is 19-dimensional and consists of the ego lane index, the ego speed, the speed limit within the current segment, the target lane (i.e., the lanes that lead the ego to the next road segment on its route), ego progress within the current segment, and the surrounding vehicles' information. In I-280, the surrounding vehicles include the leading cars and following cars on the lanes adjacent to the ego car (i.e., the left lane, the current lane

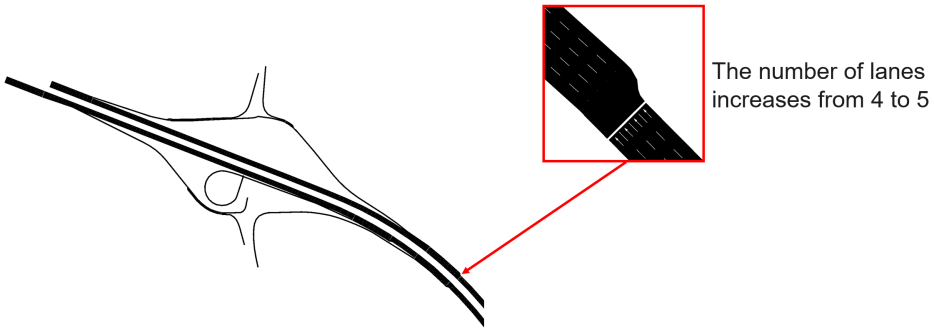
3. Our I-280 domain can be found here: https://github.com/songanz/flow_evaluation.



(a) I280-overview



(b) I280-begin



(c) I280-end

Figure 9: Figure 9a presents the overview of the I-280 domain. The ego vehicle is tasked to join the highway from the ramp and then exit the environment at the end of the highway. The ego vehicle's entrance ramp and exit is zoomed in and presented in Figure 9b and Figure 9c, respectively.

of the ego car, and the right lane). For each surrounding vehicle, the domain provides its distance to the ego car and velocity information. The action space of the ego agent is two-dimensional, which controls the ego vehicle's acceleration and steering angle. *There are 119,347 passenger vehicles*

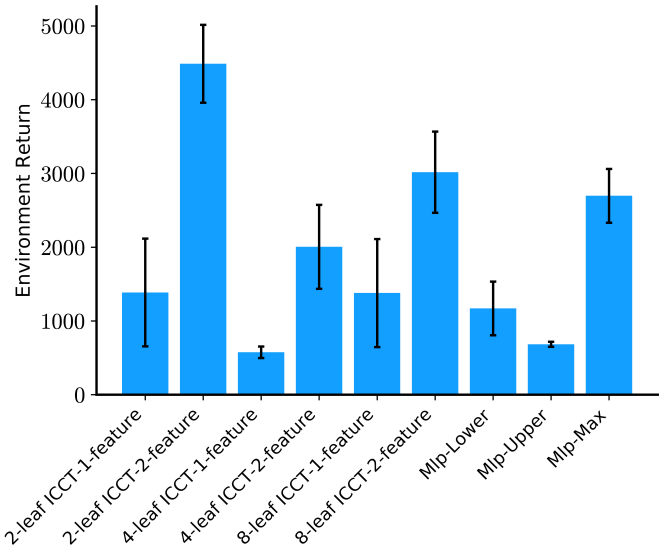


Figure 10: This figure compares the performance of ICCT agents and MLP agents in the I-280 domain. The environment returns for each model are displayed through a mean and standard deviation across ten evaluation episodes.

in total involved within this domain, creating an extremely large-scale simulation with complex, stochastic vehicle interactions. However, as dense traffic makes the simulation significantly slow, we reduce the traffic density in the original I-280 Domain to involve vehicles that are near to the ego vehicle controlled by our model. The modification makes the environment renders faster and boosts the training process.

As I-280 introduces the notion of the speed limit and complex road routing, we design an 8-component reward function, $R_{\text{I-280}} = \sum_{i=1}^{8} R_i$. Here, R_1 represents the positive reward for ego speed, R_2 represents the negative penalty if the ego vehicle is not in the target lane, R_3 encodes the negative penalty for too-small headways, R_4 provides the negative penalty for too-large headways, R_5 encodes a negative penalty for emergency braking behaviors, R_6 represents a negative penalty for unsafe lane changes, R_7 is a progress reward once the ego vehicle finishes 40% and 60% of its route, and R_8 is a reward if the ego vehicle arrives at its destination (i.e., accomplishing the task). Intuitively, R_1 and R_4 motivate the ego vehicle to move faster and follow traffic flow, R_2 helps the ego car to follow the correct route, R_3, R_5, and R_6 encourage the RL vehicle to be safe by keeping headways and avoiding dangerous operation, and R_7 and R_8 encourage the RL vehicle to move further in its trajectory.

13.4 I-280 Results

We have summarized the results of the I-280 case study in Figure 10. We compare six sizes of the ICCT and three sizes of an MLP. The "environment returns" metric represents the average episode reward in 10 rollouts. The environment returns for the 2-feature ICCT with 2 leaves and 2-feature ICCT with 8 leaves are significantly higher than the MLP models, demonstrating the ICCT's

capability on more complex and realistic domains. We also observe that ICCT-2-Feature performs better than ICCT-1-feature.

Summary: Across both realistic autonomous driving domains, we find that ICCTs can serve as safe continuous control models that follow traffic regulations and maintain high performance with respect to the domain objectives.

14. Interpretability User Study

In the previous sections, we have validated ICCT's efficacy in learning high-performance, safe policies. In this section, we seek to verify ICCT's interpretability with end-users through a human-subject experiment. To get a comprehensive understanding of ICCT's interpretability compared with neural networks, we design a $3 \times 3 \times 2 \times 2$ experiment. We describe the four independent variables as follows:

Models (3-levels): 1) *Tree*, 2) *MLP*, and 3) *Paragraph*. We compare the interpretability across the three models. The condition, *Tree*, refers to an ICCT in its original tree form (e.g., Figure 4). The condition, *Paragraph*, refers to a text description encompassing a set of rules extracted from an ICCT. We obtain this form by first training an ICCT and then transforming it into a text description after training has been completed. During the transformation, each leaf node corresponds to a sub-paragraph, which is formed by two components, namely the conditions and the consequence. The conditions of a sub-paragraph describe the junction of all decision nodes on the path from the root node to the leaf, and the consequence is the leaf node itself. For example, a four-leaf tree corresponds to four sub-paragraphs. We hypothesize such a *Paragraph* form may positively contribute to the interpretability of the ICCT as it does not assume prior familiarity with tree-based models. The condition, *MLP*, displays a multi-layer perceptron.

Repeats of evaluation (3-level): 1st, 2nd, and 3rd. To test the interpretability of the three models, we task participants to calculate the output of the model given the model parameters and its inputs. For each model, we ask participants to make predictions with the same model three times, each with varying inputs. This allows us to attain a lower-variance estimation of both the participant's accuracy and time spent in model computation. This condition also allows us to compare the learning effect of interpreting each of the models (i.e., improvements from repeated evaluations).

Contexts (2-levels): 1) With Context and 2) Without Context. We investigate whether providing context (i.e., a visualization of the traffic scenario) contributes to the user's rating of model interpretability. This condition helps us understand whether each model's interpretability is its inherent property from its model structure or is related to specific example contexts.

Domains (2-levels): 1) Multi-Lane Ring and 2) I-94. We examine whether models' interpretability is impacted by the environment's complexity.

Out of the four factors, we design *Models* and *Repeats of evaluation* to be within-subject factors as we seek to test for the learning effects and choose *Contexts* and *Domains* to be between-subject factors. We test for the following six hypotheses from the user study.

- H1: *Tree* and *Paragraph* are easier to simulate than neural network, i.e., the simulation of *Tree* and *Paragraph* has higher accuracy than *MLP*.
- H2: *Tree* and *Paragraph* are quicker to validate than MLP.
- H3: *Tree* and *Paragraph* are more interpretable than *MLP*, measured by a 13-item Likert questionnaire introduced by (Paleja et al., 2020).

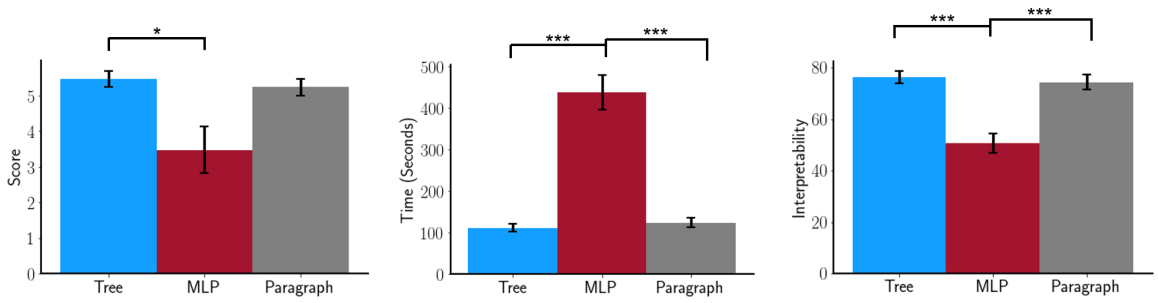


Figure 11: This figure shows the comparisons of accuracy score (left), time spent (middle), and subjective interpretability rated (right) across the three models in the I-94 user study. * denotes a significant difference of $p < .05$. *** denotes a significant difference of $p < .001$.

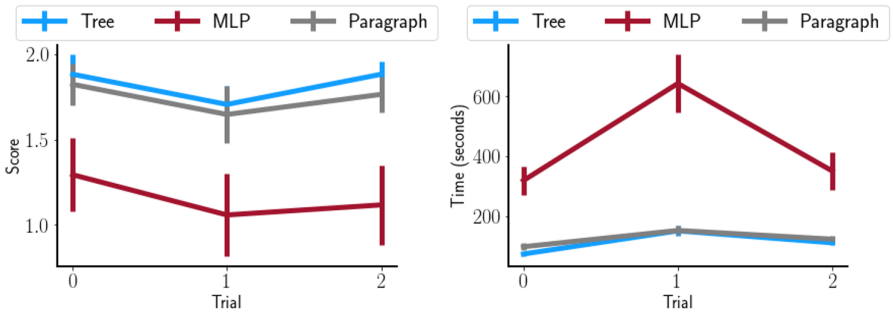


Figure 12: This figure shows the accuracy score (left) and time spent (right) changes in three repeats trials across the three models in the user study.

- H4: Performance improvement by repeated evaluations of *Tree* is larger than of *MLP*.
- H5: Environment context of the decision-making increases interpretability.
- H6: The advantage of *Tree* and *Paragraph*'s subjective interpretability over *MLP* is domain-independent.

To test for the six hypotheses, we build an online survey with Qualtrics. In each question of the survey, we show a model (*Tree*, *MLP*, or *Paragraph*) and an input feature vector. The subject is asked to make predictions (i.e., compute the output of the model) given the respective input. We collect $N = 34$ responses. The average age of participants is 25.15 with a standard deviation of 4.28. Out of the 34 participants, 15 are male, and 19 are female.

14.1 User Study Results

For H1-H5, we illustrate results on the I-94 domain, and for H6, we show the results on both the Multi-lane Ring and I-94 domains to verify the conclusions hold for both environments. For

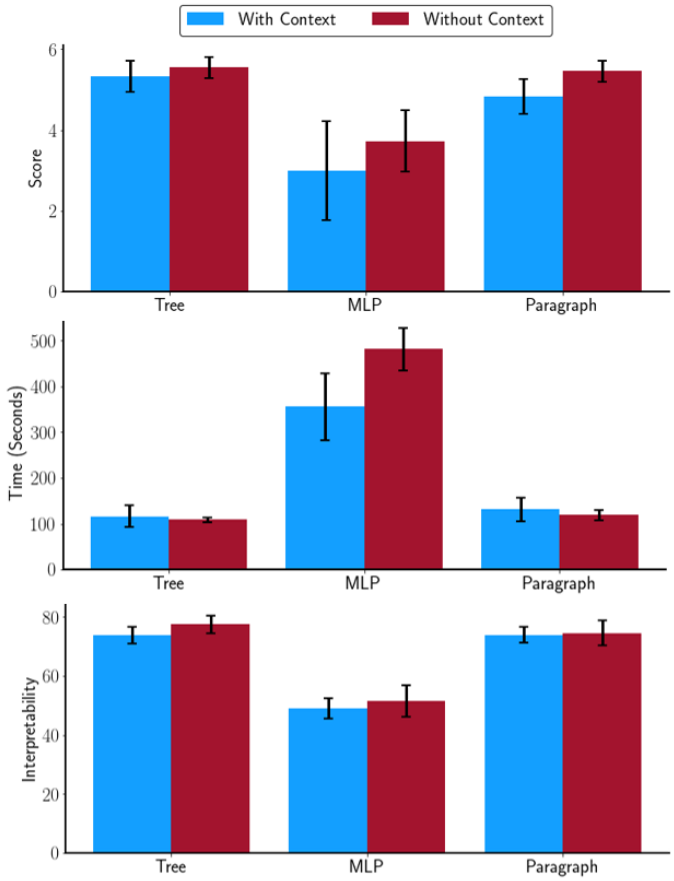


Figure 13: This figure shows the comparisons of accuracy score (left), time spent (middle), and interpretability rated (right) with or without context across the three models in the user study.

all statistical tests, the assumptions for the ANOVA test are not satisfied. Thus, we instead perform a non-parametric Friedman test followed by a posthoc Nemenyi–Damico–Wolfe (Nemenyi) test (Damico and Wolfe, 1987).

H1-H3: We summarize the results for H1-H3 in Figure 11. As we test the user's simulation of outputs on each model three times and the action output is two-dimensional (acceleration and lane-changing), each user is validated across six action outputs for each model. We denote the number of accurate answers out of the six as the "score" in Figure 11 left. We observe that participants are able to simulate the outputs of the *ICCT* more accurately than an *MLP* ($p < .05$), supporting H1. Furthermore, the time spent on the *ICCT* and *Paragraph* to evaluate the output is significantly less than the time spent on MLP ($p < .001$), shown in Figure 11 (middle). *Tree* and *Paragraph* are also rated by users to have significantly higher interpretability ($p < .001$) (Figure 11 right). As such, the results from the user study support H1-H3, showing *Tree* and *Paragraph* are easier to simulate, quicker to validate, and more interpretable than neural networks. We find there is no significant difference across all three metrics between the tree form and the paragraph form of the ICCT.

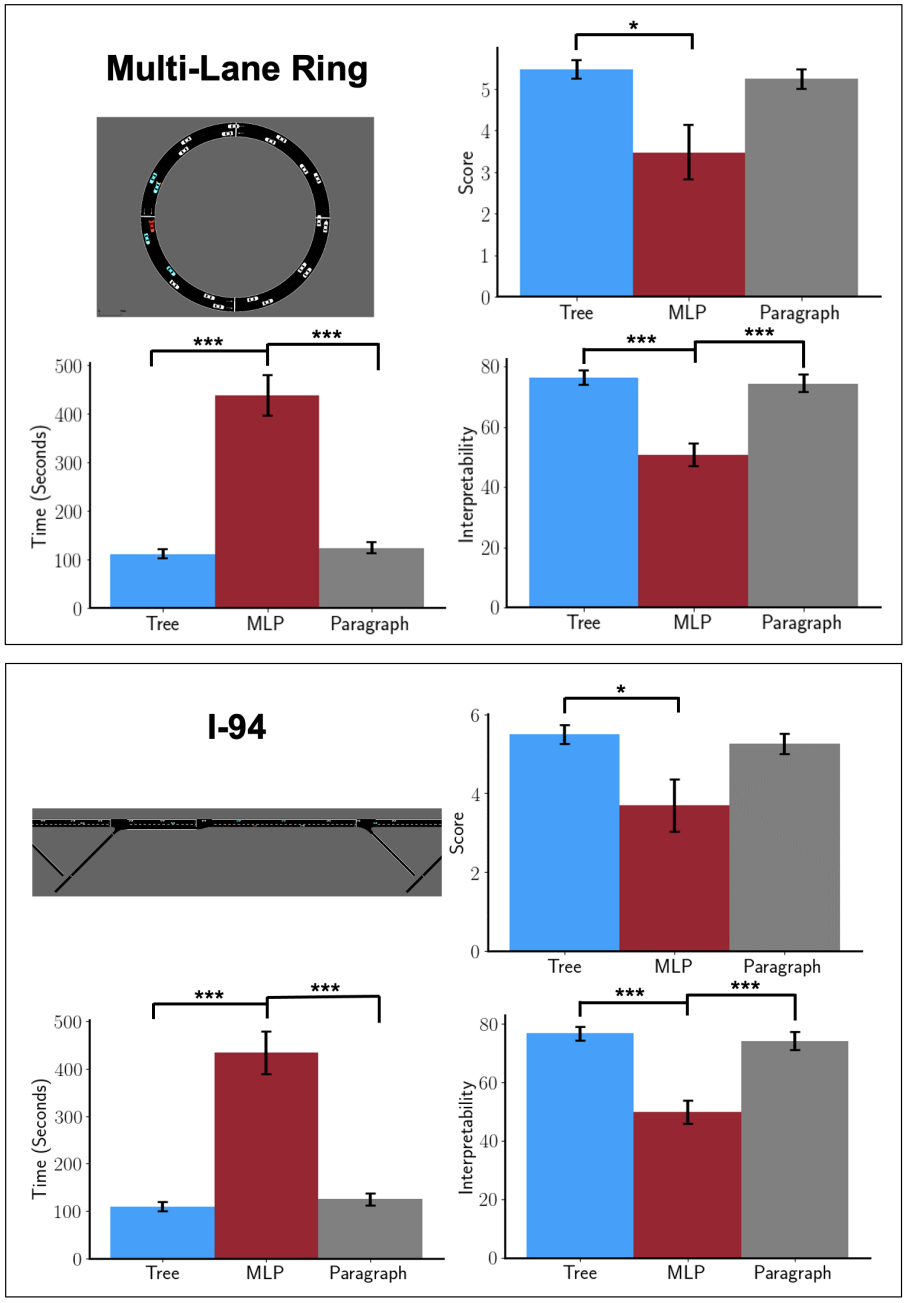


Figure 14: This figure shows the comparison of results between the Multi-Lane Ring domain and the I-94 domain across the three models in the user study. * denotes a significant difference of $p < .05$. *** denotes a significant difference of $p < .001$.

H4: We show the results to test for H4 in Figure 12. We hypothesize that with practice, the ability of the users to evaluate the models may increase. However, Figure 12 shows that the accuracy

on the first trial is the highest, and the time spent on the first trial is the lowest. Instead of the learning effect, the finding may be explained by a fatiguing effect, which causes the participants to have lower performance in later trials. Another hypothesis is that the first trial questions are relatively easy as they correspond to the early stages of the execution, where the environment has fewer cars. The changes in accuracy score and time across three trials do not have a significant interaction effect with models, and therefore H4 is not supported. However, across repeated iterations, we observe that the score and the prediction time of the ICCT tree form and paragraph form are close while being much easier to simulate than an MLP.

H5: We illustrate the result for H5 in Figure 13. We observe that generally, for all three models, the condition with context results in slightly lower accuracy scores and interpretability scores. For *Tree* and *Paragraph*, the time spent with context is slightly higher than the condition without context. However, the time spent on MLP without context is higher than with context. One possible reason could be that tree depictions and paragraph descriptions are interpretable and quick to evaluate, and therefore context does not provide more benefit but introduces some workload overhead. For MLP, the context helps the user to understand the situation and therefore makes the evaluation faster. Overall, H5 is not supported as context does not provide a significant boost to subjective interpretability.

H6: The comparison of results between the two domains, Multi-Lane Ring and I-94, can be viewed in Figure 14. We observe that the results for H1-H3 are similar for both domains and therefore, H6 is supported by displaying the advantage of different representations of the ICCT's (both the *Tree* and *Paragraph* form) interpretability over an *MLP* is regardless of the two domains ($p < .05$ on accuracy score and $p < .001$ on both time and subjective interpretability).

15. Conclusion

In this work, we present a novel tree-based model for continuous control, applicable to a wide variety of domains including robotic manipulation, autonomous driving, etc. Our Interpretable Continuous Control Trees (ICCTs) have competitive performance to that of deep neural networks across several continuous control domains, including six difficult autonomous driving scenarios and two driving domains grounded in realistic lane geometries, while maintaining high interpretability. The maintenance of high performance within an interpretable and verifiable reinforcement learning architecture provides a paradigm that would be beneficial for the safe real-world deployment of autonomous systems.

16. Limitations and Future Work:

Our framework has several limitations. Continuous control outputs (e.g., predicting a steering angle) may not be interpretable to end-users and may require post-processing to enhance a user's understanding. Also, the relationship between controller sparsity, tree depth, and interpretability is not clear, making controller sparsity and tree depth difficult-to-define hyperparameters. We also note that in more challenging environments, larger ICCTs may be required to increase their representative power. In these instances, although the size of Interpretable Continuous Control Trees (ICCTs) may pose challenges for end-users in terms of interpretation, it is important to note that they can still be verified by experts and interpreted within specific tree sub-spaces.In future work, we will extend ICCTs to incorporate constraints from end-users, provide safety guarantees on our ICCTs,

and reason about how the complexity of an ICCT may change as we move to higher-abstraction state spaces.

Acknowledgments

This work was supported by a gift award from the Ford Motor Company, National Science Foundation (NSF) 1757401 (SURE Robotics), NSF CNS 2219755, and a research grant from MIT Lincoln Laboratory.

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.
Delivered to the U.S. Government with Unlimited Rights, as defined in DFARS Part 252.227-7013 or 7014 (Feb 2014). Notwithstanding any copyright notice, U.S. Government rights in this work are defined by DFARS 252.227-7013 or DFARS 252.227-7014 as detailed above. Use of this work other than as specifically authorized by the U.S. Government may violate any copyrights that exist in this work.

References

Ryosuke Abe. Introducing autonomous buses and taxis: Quantifying the potential benefits in japanese transportation systems. *Transportation Research Part A: Policy and Practice*, 2019.

Amina Adadi and Mohammed Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160, 2018.

Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. Learning certifiably optimal rule lists for categorical data. *The Journal of Machine Learning Research*, 18 (1):8753–8830, 2017.

Lisa Anne Hendricks, Ronghang Hu, Trevor Darrell, and Zeynep Akata. Grounding visual explanations. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 264–279, 2018.

Avijit Banerjee and Radhakant Padhi. Nonlinear guidance and autopilot design for lunar soft landing. In *2018 AIAA Guidance, Navigation, and Control Conference*, page 1872, 2018.

Pietro Barbiero, Gabriele Ciravegna, Dobrik Georgiev, and Franscesco Giannini. Pytorch, explain! a python library for logic explained networks. *arXiv preprint arXiv:2105.11697*, 2021.

J Basak. Online adaptive decision trees. *Neural computation*, 16(9):1959–1981, 2004.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31, 2018a.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31, 2018b.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *ArXiv*, abs/1308.3432, 2013.

Umang Bhatt, Alice Xiang, Shubham Sharma, Adrian Weller, Ankur Taly, Yunhan Jia, Joydeep Ghosh, Ruchir Puri, José M. F. Moura, and Peter Eckersley. Explainable machine learning in deployment, 2019.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, 2016.

Chaofan Chen and Cynthia Rudin. An optimization approach to learning falling rule lists. *arXiv preprint arXiv:1710.02572*, 2017.

Hongge Chen, Huan Zhang, Si Si, Yang Li, Duane Boning, and Cho-Jui Hsieh. Robustness verification of tree-based models. *Advances in Neural Information Processing Systems*, 32, 2019.

Gabriele Ciravegna, Pietro Barbiero, Francesco Giannini, M. Gori, Pietro Li'o, Marco Maggini, and S. Melacci. Logic explained networks. *ArXiv*, abs/2108.05149, 2021.

Marco Tulio Correia Ribeiro. *Model-Agnostic Explanations and Evaluation of Machine Learning*. PhD thesis, University of Washington, 2018.

Jiaxun Cui, William Macke, Harel Yedidsion, Aastha Goyal, Daniel Urielli, and Peter Stone. Scalable multiagent driving policies for reducing traffic congestion. *ArXiv*, abs/2103.00058, 2021.

George V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

Joseph A Damico and Douglas A Wolfe. Extended tables of the exact distribution of a rank statistic for all treatments multiple comparisons in one-way layout designs. *Communications in Statistics-Theory and Methods*, 16(8):2343–2360, 1987.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Nicholas Frosst and Geoffrey E. Hinton. Distilling a neural network into a soft decision tree. *ArXiv*, abs/1711.09784, 2017.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *ArXiv*, abs/1802.09477, 2018.

Abhishek Ghose and Balaraman Ravindran. Interpretability with accurate small models. *Frontiers in Artificial Intelligence*, 3, 2020.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL `http://www.deeplearningbook.org`.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Danijar Hafner, Timothy P. Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *ArXiv*, abs/2010.02193, 2021.

Daniel Hein, Steffen Limmer, and Thomas A. Runkler. Interpretable control by reinforcement learning. *ArXiv*, abs/2007.09964, 2020.

Lisa Anne Hendricks, Ronghang Hu, Trevor Darrell, and Zeynep Akata. Generating counterfactual explanations with natural language. *arXiv preprint arXiv:1806.09809*, 2018.

Eric Jang, Shixiang Shane Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *ArXiv*, abs/1611.01144, 2017.

John M Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Zídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A A Kohl, Andy Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David A. Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with alphafold. *Nature*, 596:583 – 589, 2021.

Christos Katrakazas, Mohammed A. Quddus, Wen-Hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C-emerging Technologies*, 60:416–442, 2015.

Guy Katz, Clark W. Barrett, David L. Dill, Kyle D. Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. *ArXiv*, abs/1702.01135, 2017.

Been Kim. *Interactive and interpretable machine learning models for human machine collaboration*. PhD thesis, Massachusetts Institute of Technology, 2015.

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

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*, pages 1467–1475, 2015.

Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1675–1684, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939874. URL https://doi.org/10.1145/2939672.2939874.

Dmitry Laptev and Joachim M Buhmann. Convolutional decision trees for feature learning and segmentation. In *German Conference on Pattern Recognition*, pages 95–106. Springer, 2014.

Benjamin Letham, Cynthia Rudin, Tyler H McCormick, David Madigan, et al. Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, 9(3):1350–1371, 2015.

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

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2016.

Pantelis Linardatos, Vasilis Papastefanopoulos, and S. Kotsiantis. Explainable ai: A review of machine learning interpretability methods. *Entropy*, 23, 2021.

Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

Wei-Yin Loh. Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 1(1):14–23, 2011.

Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *2018 21st international conference on intelligent transportation systems (ITSC)*, pages 2575–2582. IEEE, 2018.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 3(3):e10, 2018.

C Olaru and L Wehenkel. A complete fuzzy decision tree technique. *Fuzzy sets and systems*, 138 (2):221–254, 2003.

Rohan Paleja, Andrew Silva, Letian Chen, and Matthew Gombolay. Interpretable and personalized apprenticeship scheduling: Learning interpretable scheduling policies from heterogeneous user demonstrations. *Advances in Neural Information Processing Systems*, 33:6417–6428, 2020.

Rohan Paleja, Muyleng Ghuy, Nadun Ranawaka Arachchige, Reed Jensen, and Matthew Gombolay. The utility of explainable ai in ad hoc human-machine teaming. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 610–623. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/file/05d74c48b5b30514d8e9bd60320fc8f6-Paper.pdf`.

Ian Parberry. *Introduction to Game Physics with Box2D*. CRC Press, 2017.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

C. Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1:206–215, 2018.

Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *ArXiv*, abs/2103.11251, 2021.

Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. A convex relaxation barrier to tight robustness verification of neural networks. In *Neural Information Processing Systems*, 2019.

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

Rastko R. Selmic and Frank L. Lewis. Neural-network approximation of piecewise continuous functions: application to friction compensation. *IEEE transactions on neural networks*, 13 3: 745–51, 2002.

Andrew Silva and Matthew Gombolay. Encoding human domain knowledge to warm start reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5042–5050, 2021.

Alberto Suárez and James F Lutsko. Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1297–1311, 1999.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, D. Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

Pradyumna Tambwekar, Andrew Silva, Nakul Gopalan, and Matthew C. Gombolay. Natural language specification of reinforcement learning policies through differentiable decision trees. *IEEE Robotics Autom. Lett.*, 8(6):3621–3628, 2023. doi: 10.1109/LRA.2023.3268593. URL `https://doi.org/10.1109/LRA.2023.3268593`.

Ryutaro Tanno, Kai Arulkumaran, Daniel C. Alexander, Antonio Criminisi, and Aditya V. Nori. Adaptive neural trees. *arXiv preprint arXiv:1807.06699*, 2018.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

Paul Voigt and Axel Von dem Bussche. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

Sholom M. Weiss and Nitin Indurkhya. Rule-based machine learning methods for functional prediction. *Journal of Artificial Intelligence Research*, 3:383–403, 1995.

Sean Wilson, Paul Glotfelter, Li Wang, Siddharth Mayya, Gennaro Notomista, Mark L. Mote, and Magnus Egerstedt. The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems. *IEEE Control Systems*, 40: 26–44, 2020.

Cathy Wu, Aboudy Kreidieh, Kanaad Parvate, Eugene Vinitsky, and Alexandre M. Bayen. Flow: Architecture and benchmarking for reinforcement learning in traffic control. *ArXiv*, abs/1710.05465, 2017.

Mike Wu, Michael C. Hughes, Sonali Parbhoo, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. Beyond sparsity: Tree regularization of deep models for interpretability. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18, pages 1670–1678. AAAI Press, 2018. ISBN 978-1-57735-800-8.

Mike Wu, Sonali Parbhoo, Michael Hughes, Ryan Kindle, Leo Celi, Maurizio Zazzi, Volker Roth, and Finale Doshi-Velez. Regional tree regularization for interpretability in deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6413–6421, 2020.

Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and K. Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020.