Learning Interpretable, High-Performing Policies for Continuous Control

Rohan Paleja[1,†], Yaru Niu[2,†,*], Andrew Silva[1], Chace Ritchie[1], Sugju Choi[1], Matthew Gombolay[1]

***Abstract*— Gradient-based approaches in reinforcement learning (RL) have achieved tremendous success in learning policies for continuous control problems. While the performance of these approaches warrants real-world adoption in domains, such as in autonomous driving and robotics, these policies lack interpretability, limiting deployability in safety-critical and legally-regulated domains. Such domains require interpretable and verifiable control policies that maintain high performance. We propose Interpretable Continuous Control Trees (ICCTs), a tree-based model that can be optimized via modern, gradient-based, RL 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 interpretable policy representations that parity or outperform baselines by up to 33% in autonomous driving scenarios while achieving a 300x-600x reduction in the number of policy parameters against deep learning baselines. We release our codebase at https://github.com/CORE-Robotics-Lab/ICCT.**

I. 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 [1] and autonomous driving [2] to protein folding [3] and traffic regulation [4]. While the performance of these policies opens up the possibility of real-world adoption in domains such as autonomous driving and robotics, the conventional deep-RL policies used in prior work [1], [2], [4] lack interpretability, limiting deployability in safety-critical and legally-regulated domains [5], [6], [7], [8]. White-box approaches, as opposed to typical black-box models (e.g., deep neural nets) 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. Utilizing white-box approaches within machine learning can help to build trust, ensure safety, and enable end-users to inspect policies before deploying them to the real world [9], [10], [11]. 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 problems.

Prior work [9], [12], [10] has attempted to approximate interpretability via explainability, a practice that can have severe consequences [13]. 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 [13]. In contrast to local explanations, an *interpretable model* provides a transparent *global* representation of a policy's behavior. This model can be understood directly by its structure and parameters [14] (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 [15], [16], [17] or imitation learning via a decision tree across trajectories generated via a deep model [18], leaving much to be desired. Interpretable RL remains an open challenge [19]. 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.

II. Method

In this section, we introduce our ICCTs, a novel interpretable reinforcement learning architecture. We provide several extensions to prior DDT frameworks (detailed in Section V) 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.

A. ICCT Architecture

Our ICCTs are initialized to be a symmetric decision tree with n_l decision leaves (red nodes in Figure 1) and $n_l - 1$ decision nodes (blue nodes in Figure 1). The tree depth is given by $\log_2(n_l)$. Each decision leaf is represented by a linear sparse controller that is operated on $\vec{x}$. Decisions are routed via decision nodes towards a leaf controller, which is then used to produce the continuous control output (e.g., acceleration or steering wheel angle).

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 7. Each leaf node, l_d, where

[1]Georgia Institute of Technology
[2]Carnegie Mellon University
∗Work done at Georgia Institute of Technology
†Authors contributed equally to this work

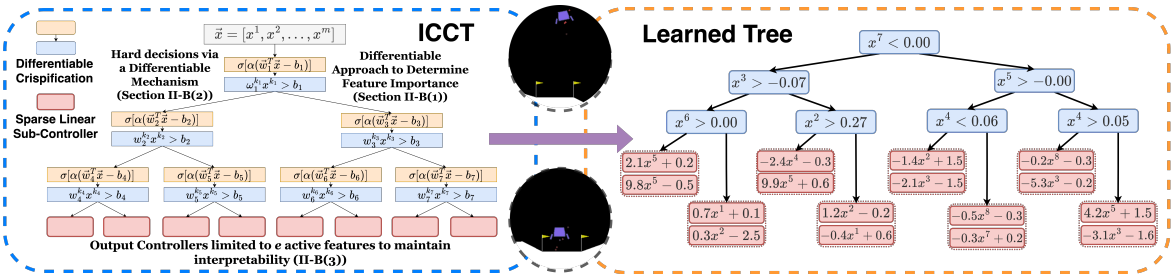


Fig. 1: The ICCT framework (left) displays decision nodes, both in their fuzzy form (orange blocks) and crisp form (blue blocks), and sparse linear leaf controllers with pointers to sections discussing our contributions. A learned representation of a high-performing ICCT policy in Lunar Lander (right) 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.

$d \in \{1, \ldots, n_l\}$, 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.

$$l_d \triangleq (\vec{u} \circ \vec{\beta}_d)^T (\vec{u} \circ \vec{x}) - \vec{u}^T \vec{\phi}_d = \vec{\beta}_d^T \vec{x} - \vec{u}^T \vec{\phi}_d \quad (1)$$

Before enforcing leaf node sparsity (Section II-B.3), $\vec{u} = [1, ..., 1]^T$ is an all-ones vector, representing a selection of all input features for the leaf node. 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 7 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, which is represented in the dense representation of $\vec{\beta}_d^T \vec{x} - \vec{u}^T \vec{\phi}_d$, as a fuzzy leaf node. It is worth noting that we parameterize the bias term as a vector $\vec{\phi}_d$ instead of a scalar to provide a corresponding bias for each feature and facilitate feature-wise optimization for the bias.

Utilizing a novel differentiable crispification procedure to convert fuzzy decision nodes into crisp decision nodes and fuzzy leaf nodes into sparse leaf nodes, 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 II-B.1 and II-B.2 and leaf controller sparsification procedure in Section II-B.3.

B. ICCT Key Elements

In this section, we discuss our ICCT's interpretable procedure for determining an action given an input feature. 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 II-B.1), and the evaluation of a decision node results in a Boolean (Section II-B.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 II-B.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, (Step 6) during training 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**

1) Decision Node Crispification: The NODE_CRISP procedure in Algorithm 1 recasts each decision node to split upon a single dimension of $\vec{x}$ to achieve sparsity while maintaining differentiable. Instead of using a non-differentiable argument max function as in [20] to determine the most impactful feature dimension, we utilize a softmax function, also known as softargmax [21], described by Equation 2. 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 $f(\cdot)$.

$$f(\vec{w}_i)_k = \frac{\exp\left(\frac{w_i^k}{\tau}\right)}{\sum_j^m \exp\left(\frac{w_i^j}{\tau}\right)} \tag{2}$$

We utilize a differentiable one_hot function, $g(\cdot)$, which produces a one-hot vector with the element associated with the highest-weighted class set to one and all other elements set to zero, as shown in in Equation 3.

$$\vec{z_i} = g(f(|\vec{w_i}|)) \tag{3}$$

Here, $|\vec{w_i}|$ represents a vector with absolute elements within $\vec{w_i}$. *We maintain differentiability in the procedure described in Equation 3 by a differentiable argument max function (Algorithm 4) which utilizes the straight-through trick [22].* We provide an algorithm detailing the NODE_CRISP procedure within the Appendix (Section VI-A).

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 set of probabilities 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 representation of the decision node output $\vec{v}_i = [\alpha(w_i^k x^k - b_i), 0]$, 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 4. 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 $\mathbb{1}(\alpha(w_i^k x^k - b_i) > 0)$.

$$(y_i, 1 - y_i) = g(f(\vec{v}_i)) \tag{4}$$

We provide an algorithm detailing the OUTCOME_CRISP procedure within Section VI-B.

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 interpretability.

Equation 5 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 differentiable k_hot function, denoted by $h(\cdot)$, takes as input a vector of class 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.

$$\vec{u}_d = h(f(|\vec{\theta}_d|)) \tag{5}$$

Here, $|\vec{\theta}_d|$ represents a vector with absolute elements within $\vec{\theta}_d$. We maintain differentiability and formulate a differentiable top-k function in Equation 5 by iteratively applying the differentiable argument max function (Algorithm 4) for k times. In Equation 6, we transform a fuzzy leaf node, for leaf, l_d (Equation 1), into a sparse linear sub-controller, l_d^*, with the sparse feature selection vector, $\vec{u_d}$.

$$l_d^* \triangleq (\vec{u}_d \circ \vec{\beta}_d)^T (\vec{u}_d \circ \vec{x}) + \vec{u}_d^T \vec{\phi}_d \tag{6}$$

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

III. EXPERIMENTS

A. Environments

We evaluated our proposed methods and baselines across six domains, including two common continuous control environments: Inverted Pendulum [23] and Lunar Lander [24], [25], and four autonomous driving scenarios: Lane-Keeping provided by [26] and Single-Lane Ring, Multi-Lane Ring, and Figure-8 all provided by the Flow [2]. We provide additional details within Section XII-B.

B. Baselines

We compare our methods against interpretable models, black-box models, and models that can be converted post-hoc into an interpretable form. We also include the number of parameters[1] for each method, shown in Table I. The details of the baselines and the calculation of the number of parameters are presented in Section XII-A in the appendix.

C. Results and Discussion

We present the results of our trained policies in Table I. We provide the performance of each method alongside the associated complexity of each benchmark in Table I 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, 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 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 accessing any superior expert policy.

Controller Leaf Distributions (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

[1]We only consider the active parameters involved during the deployment of the trained model.

TABLE I: 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. All presented results are averaged over 5 seeds.

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)

achieve the maximum performance in the simple domain of Inverted Pendulum. However, CDDT-controllers-crisp encounters an inconsistency issue from the crispification procedure of [20], [27] and achieves very low performance. We provide additional results within Section VIII-B 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 (denoted by the dark orange shade).

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 [20], [27] 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 to we can outperform MLPs by 33% on average while achieving a 300x-600x reduction in the number of policy parameters required.

IV. Conclusion

In this work, we present a novel tree-based model for continuous control problems. Our Interpretable Continuous Control Trees (ICCTs) can directly optimize over a sparse decision-tree-like representation for RL in continuous action spaces, and can maintain sparsity-adjustable liner sub-controllers on the leaf nodes. As a result, ICCTs have competitive performance to that of deep neural networks across six continuous control domains, including four difficult autonomous driving scenarios, while maintaining high interpretability. The maintenance of both high performance and interpretability within an interpretable reinforcement learning architecture provides a paradigm that would be beneficial for the real-world deployment of autonomous systems.

REFERENCES

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2016.

[2] C. Wu, A. Kreidieh, K. Parvate, E. Vinitsky, and A. M. Bayen, "Flow: Architecture and benchmarking for reinforcement learning in traffic control," *ArXiv*, vol. abs/1710.05465, 2017.

[3] J. M. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Zídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. A. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, and D. Hassabis, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, pp. 583 – 589, 2021.

[4] J. Cui, W. Macke, H. Yedidsion, A. Goyal, D. Urielli, and P. Stone, "Scalable multiagent driving policies for reducing traffic congestion," *ArXiv*, vol. abs/2103.00058, 2021.

[5] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning," *arXiv preprint arXiv:1702.08608*, 2017.

[6] B. Letham, C. Rudin, T. H. McCormick, D. Madigan *et al.*, "Interpretable classifiers using rules and bayesian analysis: Building a better stroke prediction model," *The Annals of Applied Statistics*, vol. 9, no. 3, pp. 1350–1371, 2015.

[7] U. Bhatt, A. Xiang, S. Sharma, A. Weller, A. Taly, Y. Jia, J. Ghosh, R. Puri, J. M. F. Moura, and P. Eckersley, "Explainable machine learning in deployment," 2019.

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

[9] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The building blocks of interpretability," *Distill*, vol. 3, no. 3, p. e10, 2018.

[10] L. A. Hendricks, R. Hu, T. Darrell, and Z. Akata, "Generating counterfactual explanations with natural language," *arXiv preprint arXiv:1806.09809*, 2018.

[11] L. Anne Hendricks, R. Hu, T. Darrell, and Z. Akata, "Grounding visual explanations," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 264–279.

[12] B. Kim, "Interactive and interpretable machine learning models for human machine collaboration," Ph.D. dissertation, Massachusetts Institute of Technology, 2015.

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

[14] G. Ciravegna, P. Barbiero, F. Giannini, M. Gori, P. Li'o, M. Maggini, and S. Melacci, "Logic explained networks," *ArXiv*, vol. abs/2108.05149, 2021.

[15] N. Frosst and G. E. Hinton, "Distilling a neural network into a soft decision tree," *ArXiv*, vol. abs/1711.09784, 2017.

[16] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," *Advances in neural information processing systems*, vol. 31, 2018.

[17] M. Wu, M. C. Hughes, S. Parbhoo, M. Zazzi, V. Roth, and F. 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*, ser. AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018, pp. 1670–1678.

[18] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," *Advances in neural information processing systems*, vol. 31, 2018.

[19] C. Rudin, C. Chen, Z. Chen, H. Huang, L. Semenova, and C. Zhong, "Interpretable machine learning: Fundamental principles and 10 grand challenges," *ArXiv*, vol. abs/2103.11251, 2021.

[20] A. Silva and M. Gombolay, "Encoding human domain knowledge to warm start reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 5042–5050.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[22] Y. Bengio, N. Léonard, and A. C. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *ArXiv*, vol. abs/1308.3432, 2013.

[23] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[24] I. Parberry, *Introduction to Game Physics with Box2D*. CRC Press, 2017.

[25] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[26] E. Leurent, "An environment for autonomous driving decision-making," https://github.com/eleurent/highway-env, 2018.

[27] R. Paleja, A. Silva, L. Chen, and M. Gombolay, "Interpretable and personalized apprenticeship scheduling: Learning interpretable scheduling policies from heterogeneous user demonstrations," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6417–6428, 2020.

[28] A. Suárez and J. F. Lutsko, "Globally optimal fuzzy decision trees for classification and regression," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 12, pp. 1297–1311, 1999.

[29] D. Hafner, T. P. Lillicrap, M. Norouzi, and J. Ba, "Mastering atari with discrete world models," *ArXiv*, vol. abs/2010.02193, 2021.

[30] M. Vasić, A. Petrović, K. Wang, M. Nikolić, R. Singh, and S. Khurshid, "Moët: Mixture of expert trees and its application to verifiable reinforcement learning," *Neural Networks*, vol. 151, pp. 34–47, 2022.

[31] H. Chen, H. Zhang, S. Si, Y. Li, D. Boning, and C.-J. Hsieh, "Robustness verification of tree-based models," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[32] G. Katz, C. W. Barrett, D. L. Dill, K. D. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," *ArXiv*, vol. abs/1702.01135, 2017.

[33] H. Lakkaraju, S. H. Bach, and J. 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*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1675–1684. [Online]. Available: https://doi.org/10.1145/2939672.2939874

[34] E. Jang, S. S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *ArXiv*, vol. abs/1611.01144, 2017.

[35] S. Wilson, P. Glotfelter, L. Wang, S. Mayya, G. Notomista, M. L. Mote, and M. Egerstedt, "The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems," *IEEE Control Systems*, vol. 40, pp. 26–44, 2020.

[36] G. V. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989.

[37] R. R. Selmic and F. L. Lewis, "Neural-network approximation of piecewise continuous functions: application to friction compensation," *IEEE transactions on neural networks*, vol. 13 3, pp. 745–51, 2002.

[38] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8, 2018.

[39] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.

[40] P. Linardatos, V. Papastefanopoulos, and S. Kotsiantis, "Explainable ai: A review of machine learning interpretability methods," *Entropy*, vol. 23, 2021.

[41] J. Kim and J. F. Canny, "Interpretable learning for self-driving cars by visualizing causal attention," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2961–2969, 2017.

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *ArXiv*, vol. abs/1707.06347, 2017.

[43] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *ArXiv*, vol. abs/1802.09477, 2018.

[44] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[45] W.-Y. Loh, "Classification and regression trees," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[46] J. Basak, "Online adaptive decision trees," *Neural computation*, vol. 16, no. 9, pp. 1959–1981, 2004.

[47] C. Olaru and L. Wehenkel, "A complete fuzzy decision tree technique," *Fuzzy sets and systems*, vol. 138, no. 2, pp. 221–254, 2003.
[48] E. Angelino, N. Larus-Stone, D. Alabi, M. Seltzer, and C. Rudin, "Learning certifiably optimal rule lists for categorical data," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 8753–8830, 2017.
[49] S. M. Weiss and N. Indurkhya, "Rule-based machine learning methods for functional prediction," *Journal of Artificial Intelligence Research*, vol. 3, pp. 383–403, 1995.
[50] C. Chen and C. Rudin, "An optimization approach to learning falling rule lists," *arXiv preprint arXiv:1710.02572*, 2017.
[51] D. Laptev and J. M. Buhmann, "Convolutional decision trees for feature learning and segmentation," in *German Conference on Pattern Recognition*. Springer, 2014, pp. 95–106.
[52] P. Kontschieder, M. Fiterau, A. Criminisi, and S. Rota Bulo, "Deep neural decision forests," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1467–1475.
[53] R. Tanno, K. Arulkumaran, D. C. Alexander, A. Criminisi, and A. V. Nori, "Adaptive neural trees," *arXiv preprint arXiv:1807.06699*, 2018.
[54] P. Barbiero, G. Ciravegna, D. Georgiev, and F. Giannini, "Pytorch, explain! a python library for logic explained networks," *arXiv preprint arXiv:2105.11697*, 2021.
[55] M. T. Correia Ribeiro, "Model-agnostic explanations and evaluation of machine learning," Ph.D. dissertation, University of Washington, 2018.
[56] M. Wu, S. Parbhoo, M. Hughes, R. Kindle, L. Celi, M. Zazzi, V. Roth, and F. Doshi-Velez, "Regional tree regularization for interpretability in deep neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 6413–6421.
[57] Z. C. Lipton, "The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery." *Queue*, vol. 16, no. 3, pp. 31–57, 2018.
[58] A. Adadi and M. Berrada, "Peeking inside the black-box: a survey on explainable artificial intelligence (xai)," *IEEE access*, vol. 6, pp. 52 138–52 160, 2018.
[59] R. Paleja, M. Ghuy, N. Ranawaka Arachchige, R. Jensen, and M. Gombolay, "The utility of explainable ai in ad hoc human-machine teaming," in *Advances in Neural Information Processing Systems*, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., vol. 34. Curran Associates, Inc., 2021, pp. 610–623. [Online]. Available: https://proceedings.neurips.cc/paper/2021/file/05d74c48b5b30514d8e9bd60320fc8f6-Paper.pdf
[60] S. Ross, G. Gordon, and D. 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*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

APPENDIX

V. DIFFERENTIABLE DECISION TREES (DDTS)

Prior work has proposed differentiable decision trees (DDTs) [28], [20] – 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 7.

$$y_i = \frac{1}{1 + \exp(-\alpha(\vec{w}_i^T \vec{x} - b_i))} \tag{7}$$

Here, the features vectors 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 [20], [27]. Leaf node distributions are then weighted by the probability of reaching the respective leaf and summed to produce a final action distribution over possible outputs.

A. Conversion of a DDT to a DT

DDTs with decision nodes represented in the form of Equation 7 are not interpretable. As DDTs maintain a one-to-one correspondence to DTs with respect to their structure, prior work [20], [27] proposed 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) [20], [27]. 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, leaf nodes, l, must be modified to produce a single output class, o, 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.

Drawbacks: 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 α to infinity). These inconsistencies can lead to performance degradation of the interpretable model, as we show in Section III, and results in an interpretable model that is not representative of and inconsistent with the model learned via reinforcement learning.[2]

In our work, we design 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-training crispification procedure. To the best of our knowledge, we are the first work to deploy an interpretable tree-based framework for continuous control.

VI. METHOD DETAILS

A. Node Crispification Algorithm

In this section, we provide a description of node crispification, displayed in Algorithm 2 and termed NODE_CRISP

[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., $w_i^{k_i} x^{k_i} < b$).

in the main paper. We display the transformation performed by node crispification by the green arrow in Figure 2. 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: $\vec{z}_i$ = 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., $k = \arg\max_j(|w_i^j|)$).

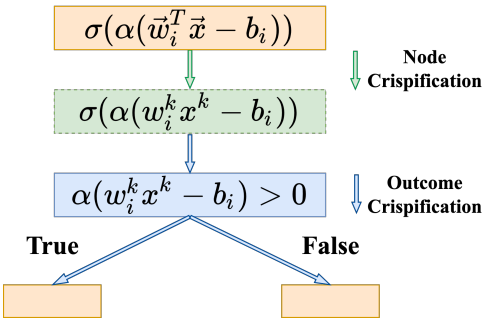


Fig. 2: This figure displays the process of our proposed differentiable crispification, including node crispification (Algorithm 2) and outcome crispification (Algorithm 3). The node crispification sparsifies the weight vector $\vec{w}_i$ and chooses the most impactful feature. The outcome crispification helps make a "hard" decision instead of a "soft" decision to choose one branch. Both operations are differentiable.

B. Outcome Crispfication Algorithm

In this section, we provide a description of outcome crispification, displayed in Algorithm 3 and termed OUTCOME_CRISP in the main paper. We display the transformation performed by outcome crispification by the blue arrows in Figure 2. 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).

C. Differentiable Argument Max Function for Differentiable Crispification

In this section, we provide a description of the differentiable argument max function which is utilized in both decision node crispification and decision outcome crispification.

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 $g(\cdot)$)
3: $\vec{h} = \vec{h}_{hard} + \vec{h}_{soft} -$ STOP_GRAD($\vec{h}_{soft}$) (step 2 for $g(\cdot)$)

Similar to [29], 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 detaches the gradient data of $\vec{h}_{soft}$. The

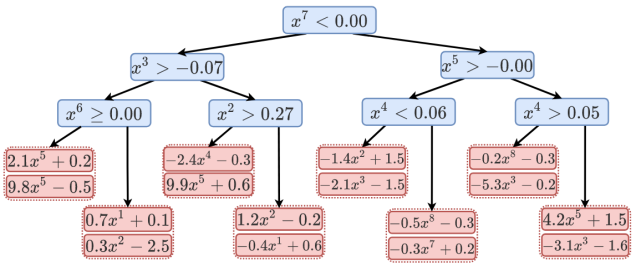


Fig. 3: A Learned ICCT in Lunar Lander

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 3 and 4.

VII. Qualitative Exposition of ICCT Interpretability

Here, we provide a display of the utility and interpretability of a learned ICCT model. In Figure 3, 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 (≤ 0.00 m), 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. Tree-based models are amerable to verification [30]. *Furthermore, tree-based models similar to ICCTs can be verified in linear time [31], while DNN verification is NP-complete [32].* 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.

VIII. Additional Results

A. Learning Curves

In Figure 4, we display the learning curves of the eleven methods across six domains shown in Table 1 of the main paper. In general, ICCT-complete has competitive or better performance with regards to running-average rollout rewards and convergence rate, compared to MLP-Max, MLP-Upper and fuzzy DDTs in Inverted Pendulum, Lunar Lander, Lane-Keeping, Sing-Lane Ring, and Multi-Lane Ring, while maintaining interpretability. We also notice as the sparsity of the linear sub-controller increases, the performance of ICCT gradually drops. However, the interpretable approaches of ICCT-3-feature and ICCT-2-feature still have comparable or better performance with respect to MLP-Lower and ICCT-L1-sparse.

B. 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. [33] 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 also 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 change 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 above 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 change 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.

C. Ablation: Differentiable Argument Max and Gumbel-Softmax

In this section, we provide an ablation study on the differentiable operator used in ICCTs. Here, we substitute the Softmax function with a Gumbel-Softmax [34] function, a widely-used differentiable approximate sampling mechanism for categorical variables, to perform decision node crispification, perform decision outcome crispification, and enforce sub-controller sparsity. Changing ICCT to utilize the Gumbel-Softmax function as opposed to DIFF_ARGMAX$(\cdot)$ in Algorithm 4 requires modifying the original Softmax

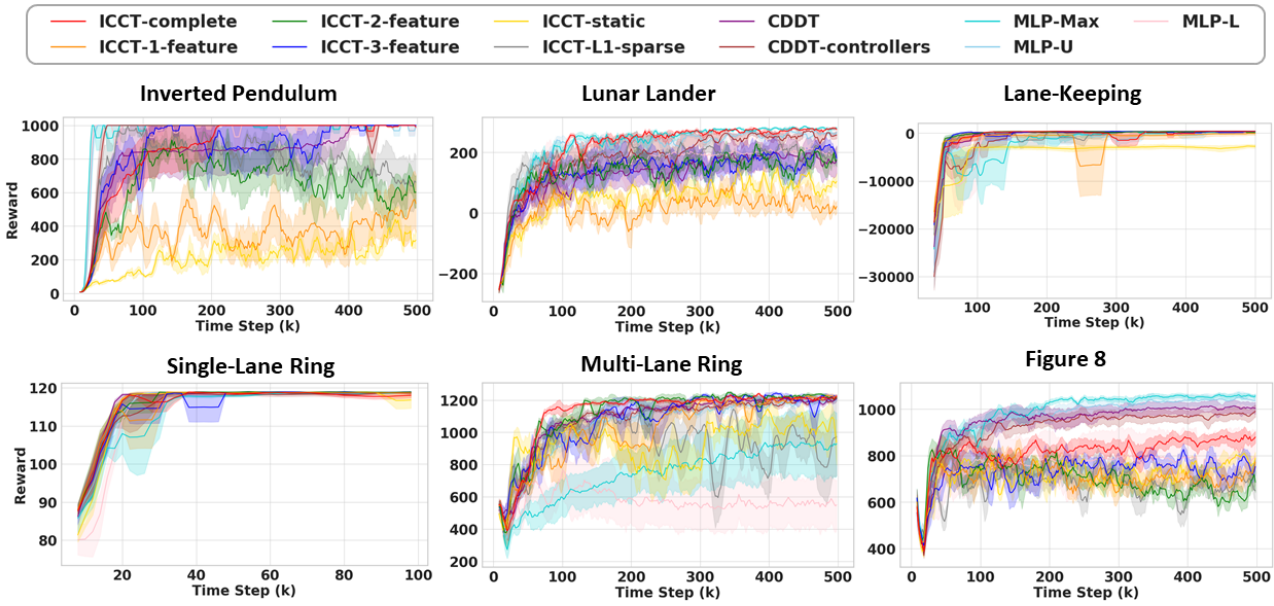


Fig. 4: In this figure, we display the learning curves of eleven methods in six domains shown in the main paper. The rewards are rollout rewards throughout the training process. The curves in 5 domains except Lane-Keeping are smoothed by a sliding window of size 5, the curves in Lane-Keeping are smoothed by a sliding window of size 20. The shadow region depicts the standard error across 5 seeds.

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

function, f, introduced by Equation 2, to f' as follows:

$$f'(\vec{w}_i)_k = \frac{\exp\left(\frac{w_i^k + g_i^k}{\tau}\right)}{\sum_j^m \exp\left(\frac{w_i^j + g_i^j}{\tau}\right)} \tag{8}$$

Here, $\vec{w}_i$ is a 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$(0, 1)$ distribution [34]. Here, 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.

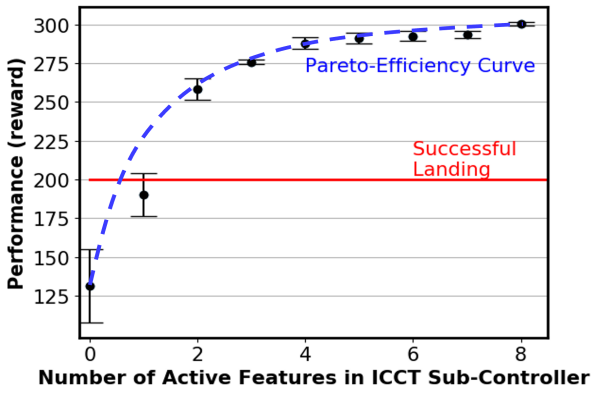
From the results shown in Figure 6 and Table II, we find that the addition of Gumbel noise reduces performance by a wide margin. Furthermore, comparing crisp ICCTs utilizing Gumbel-Softmax to ICCTs utilizing Gumbel-Softmax, we see that due to the sampling procedure within the Gumbel-Softmax, an inconsistency issue arises between non-crisp and crisp performance. Such results support our design choice of the differentiable argument max function.

D. Physical Robot Demonstration

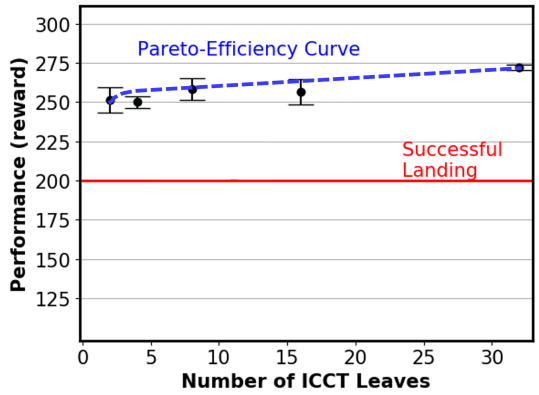
Here, we demonstrate our algorithm with physical robots in a 14-car figure-8 driving scenario and provide an online, easy-to-inspect visualization of our ICCTs, which controls the ego vehicle. We utilize the Robotarium, a remotely accessible swarm robotic research platform [35], to demonstrate the learned 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.

IX. 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



(a) Performance vs. Number of Controller Features



(b) Performance vs. Number of ICCT Leaves

Fig. 5: In this figure, we display the interpretability-performance tradeoff of our ICCTs with respect to active features within our linear sub-controllers (Figure 5a) and tree depth (Figure 5b). Within each figure, we display the Pareto-Efficiency Curve and denote the reward required for a successful lunar landing as defined by [25].

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 9, we display a computed output for a 4-leaf tree, where decision node outputs, y_i, are determined via Equation 1 of the main paper. 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 9 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.

$$ICCT(x) = p_1(y_1 * y_2) + p_2(y_1 * (1 - y_2)) + p_3((1 - y_1) * y_3) + p_4 * ((1 - y_1) * (1 - y_3)) \quad (9)$$

Equation 9 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 [36]. [36] 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 [36].

Lemma IX.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 IX.2. *An indicator function is discriminatory.*

Proof: As an indicator function is bounded and measureable, by Lemma 1 of [36], it is discriminatory.

Theorem IX.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 [36], 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 [37] 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.

X. Dynamic Depth

In this section, we present a dynamic deepening algorithm here drawing inspiration from [20]. Allowing our ICCTs to automatically deepen and increase in complexity has several advantages. In continuous control, it is typical that a deployed policy may encounter a covariate shift in a real-world setting [38]. As such, our ICCT may need to change to account for features previously thought unimportant. Dynamic deepening would allow our ICCTs to mitigate the encountered covariate shift and represent additional complexities when deployed. Furthermore, the ability to deepen eliminates the need to set the tree depth a priori.

Algorithm 5 Dynamic Deepening Procedure

Input: Pretrained ICCT P, deepened ICCT P_{deep}, controller sparsity e

```
for i epochs do
    Collect trajectory rollouts, τ, with P
    P, P_deep ← NETWORK_UPDATE(P, P_deep)
    H, H_deep ← CALCULATE_LEAF_ENTROPIES(P, P_deep, τ)
    if H(P_deep) + ε < H(P) then
        P ← P_deep
        P_deep ← DEEPEN(P_deep)
    end if
end for
```

Our proposed procedure for deepening is shown in Algorithm 5. This procedure should be conducted after a

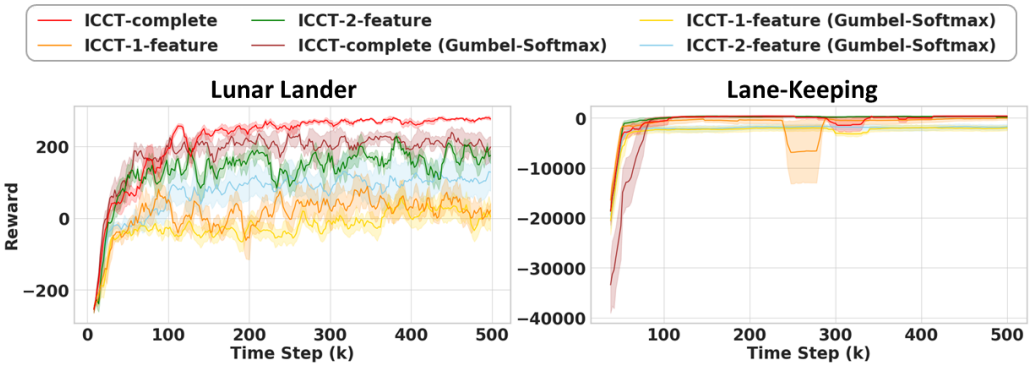


Fig. 6: This figure displays the average running rollout rewards of six methods for the ablation study during training. The results are averaged over 5 seeds, and the shadow region represents the standard error.

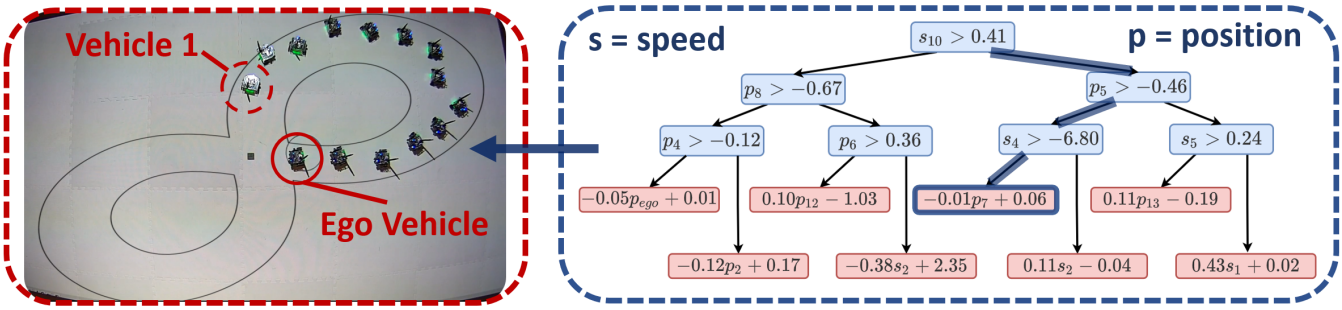


Fig. 7: In this figure, we display our ICCTs controlling a vehicle in a 14-car physical robot demonstration within a Figure-8 traffic scenario. Active nodes and edges are highlighted by the right online visualization, where s_i represents the speed of vehicle i, and p_i represents the position of vehicle i.

ICCT model, P, has been pretrained with an initial dataset (e.g., with simulated data for transfer to the real world). At initialization of the dynamic deepening procedure, two ICCT models are maintained, a shallow pre-trained version, P, and deeper-by-one-depth version P_{deep}. Both models utilize the same controller sparsity e. The deepened ICCT is initialized so that the top-level weights match that of P, and lower-level weights are randomly initialized. During deployment, the pre-trained ICCT model, P is utilized to collect trajectory rollouts, τ, as shown in Line 1 of Algorithm 5. Given these trajectory rollouts and associated rewards, both models are updated via gradient descent, as shown by the function NETWORK_UPDATE in Line 3. P is updated via Equation 2 in our main paper. As we do not have access to rollout trajectories for P_{deep}, the model update is simulated by utilizing the likelihood that P_{deep} will take similar actions to P given the states within τ. In Line 4 of Algorithm 5 (CALCULATE_LEAF_ENTROPIES), we calculate the entropy across each leaf within P and P_{deep}. As our leaf nodes are input-parameterized (based on state), we utilize the sample of trajectories collected in Line 2 to estimate the leaf entropy. Here, $\vec{H}$ and $\vec{H}_{deep}$ represent a vector of leaf entropies. The deeper ICCT has more leaves (generated via the deepening procedure) and, thus, $\vec{H}_{deep}$ is a higher-dimensional vector. In Line 5 of Algorithm 5, we compare the entropy of adjacent leaf nodes between P and P_{deep}. For example, in the case where we have P as a two-leaf tree and P_{deep} as a four-leaf tree, if the entropy of the left leaf node of P is at least ϵ greater than that of the combined entropy of the two left leaf nodes of P_{deep}, P_{deep} has learned a leaf distribution that is more precise in representing high-performance control behavior. Thus, in Line 6 and 7 of Algorithm 5, the shallow model, P, is updated the additional leaves of the deeper model, P_{deep}, and the deeper model, P_{deep}, is deepened by an additional level for each decision tree path that had lower entropy (determined in Line 5). This procedure continues for a set number of predefined epochs.

XI. Related Work

Due to recent accidents with autonomous vehicles (c.f. [39]), there has been growing interest in developing Explainable AI (xAI) approaches to understand an AV's decision-making and ensure robust and safe operation. Explainable AI (xAI) is concerned with understanding and interpreting the behavior of AI systems [40]. 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) [41], [5]. In such domains, prior work [42], [1], [43], [44] 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 [45], [46], [47] or rule list [48], [49], [6], [50]). In particular, tree-based frameworks could represent complex decision-making processes while maintaining interpretability. Decision trees [45] 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 [51], [52], [53]. Decision trees provide *global* explanations of a decision-making policy that are valid throughout the input space [54], as opposed to local explanations typically provided via "post-hoc" explainability techniques [55], [20], [27]. Several approaches have attempted to distill trained neural network models into decision trees [56], [16]. 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 [13], [57], [58], [59].

Recently, [19] presented a set of grand challenges in interpretable machine learning to guide the field towards 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.

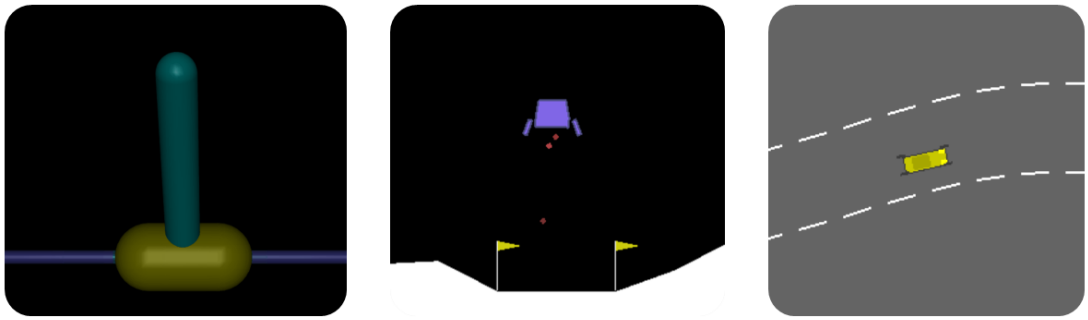
XII. Experiment Settings

A. 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. We also include the number of parameters[3] for each method, shown in Table I. We list the following notations for an easier understanding of the number of parameters. 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. The dimension of the action space d_a. The calculated number of parameters is denoted as n_p. Our approach, ICCT-e-feature, has a number of parameters of $n_p = 3(n_l - 1) + (2e + 1)d_a n_l = (2ed_a + d_a + 3)n_l - 3$.

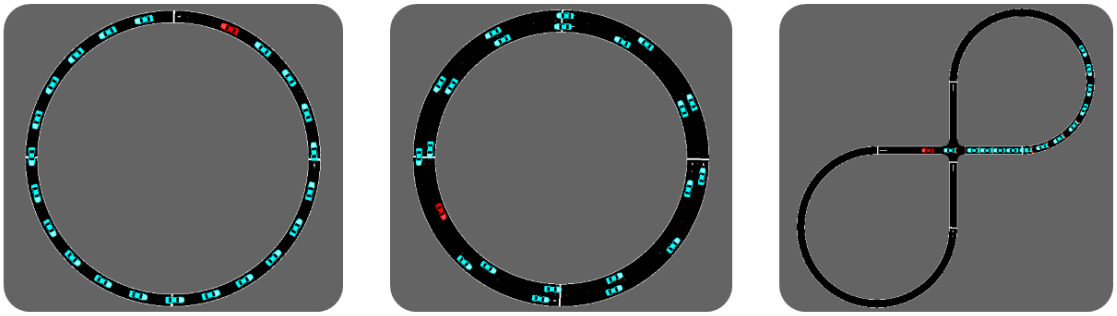
- Continuous DDTs (CDDT): We translate the framework of [20] to function with continuous action-spaces by modifying the leaf nodes to represent static probability distributions. Here, $n_p = (m+2)(n_l-1)+d_a n_l = (d_a+m+2)n_l - m - 2$. When converted into an interpretable form post-hoc, this approach 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, $n_p = 3(n_l - 1) + d_a n_l = (3 + d_a)n_l - 3$.
- 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, $n_p = 3(n_l - 1) + (m + 1)d_a n_l = (md_a + d_a + 3)n_l - 3$.
- ICCT with L1-regularized controllers (ICCT-L1-sparse): We achieve sparsity via L1-regularization applied to ICCT-complete rather than enforce 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 decision-nodes and thus, the interpretability of the resulting model is limited. Here, $n_p = 3(n_l - 1) + (m+1)d_a n_l = (md_a + d_a + 3)n_l - 3$.
- 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 after training (e.g., non-top feature weights in decision nodes). The last (MLP-Lower) maintains approximately the same number of *active* parameters as our ICCTs with sparse leaf controllers during evaluation. The number of parameters of MLP depends on the size of the network and we

[3]We only consider the active parameters involved during the deployment of the trained model.



(a) Inverted Pendulum (b) Lunar Lander (c) Lane-Keeping

Fig. 8: This figure displays the first three environments we utilized including Inverted Pendulum, Lunar Lander, and Lane-Keeping.



(a) Ring Network (b) Multi-Lane Ring Network (c) Figure-8 Network

Fig. 9: In this figure, we display the configuration of each Flow network used in our experimentation. In each image, the red vehicle is the autonomous agent that utilizes the learned control policy to stabilize traffic flow. Each cyan vehicle is a simulated human vehicle which contains a noisy acceleration behavior, the severity of which is defined by the user.

count all the weights and bias parameters but leave out all the optimizer parameters.

- Decision Tree (DT): We train a DT via CART [45] on state-action pairs generated from MLP-Max. This baseline represents the distillation approach from a high-performance black-box policy to an interpretable model. Here, $n_p = 2(n_l - 1) + d_a n_l = (2 + d_a)n_l - 2$.
- DT w\ DAgger: We utilize the DAgger imitation learning algorithm [60] to train a DT to mimic MLP-Max.

B. Environments

We provide detailed descriptions for the two common continuous control problems: Inverted Pendulum and Lunar-Lander, and autonomous driving domains: Lane-Keeping and those from Flow, including Single-Lane Ring, Multi-Lane Ring, and Figure 8.

Inverted Pendulum: Inverted Pendulum (Figure 8(a)) is provided by MujoCo [23] and OpenAI Gym [25]. The observation includes the cart position, velocity, pole angle, and pole angular velocity. The goal is to apply a force to the cart to balance a pole on it and prevent the pole from falling. A plus one reward is provided at each timestep provided if the pole keeps upright at each time step.

Lunar Lander: Lunar Lander (Figure 8(b)) is a game provided by Box2D [24] and OpenAI Gym. The goal is to land a lunar lander as close to a landing pad between the flags. 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 for controlling the main engine thruster and side thrusters. At each timestep, the lander reward is determined by a proximity to the landing pad, whether each leg is touching the landing pad, and a fuel cost. The episode ends if the lander crashes and a terminal reward of -100 is provided. If the lander successfully lands, a terminal reward of 100 is provided.

Lane-Keeping: Lane-Keeping (Figure 8(c)) is a domain with continuous actions within highway-env [26]. The observation is 12-dimensional, which consists of the vehicle's lateral position, heading, lateral speed, yaw rate, linear, lateral, and angular velocity, and the lane information. The action is the steering angle to control the vehicle. At each time step, +1 reward will be provided if the vehicle can keep in the center of the lane. The reward decreases as the vehicle drives away from the lane center. The terminal condition within this domain is a maximum timestep of 500.

Flow Domains [2]: Each of the following domains are custom continuous actions domains provided within the Flow deep reinforcement learning framework for mixed autonomy

traffic scenarios. Flow utilizes the SUMO traffic simulator, which allows for both autonomous agents and simulated human agents. The simulated human agents are constructed by adjusting the noise factor present within their acceleration and deceleration control. Stabilization of traffic is defined as the average velocity of all vehicles approaching a set value for velocity. The general observation includes the global positions and velocity of all vehicles in the network. These are normalized based upon the length and max velocity for the network, which is defined by the user, and then combined into a single 1-Dimensional array with a length that is double the number of vehicles present. The reward is measured by how closely the network's average velocity matches the user-defined average velocity. The episode is terminated if any collision between two vehicles is detected, or preset time steps are executed.

Flow Single-Lane Ring Network: The Single-Lane Ring Network (Figure 9(a)) is a ring road network, with the objective being to stabilize the flow of all traffic within the network. A control policy must apply acceleration commands to an autonomous agent in order to stabilize the flow. There are 21 human vehicles and 1 ego vehicle, with a maximum time step of 750.

Flow Multi-Lane Ring Network: The Multi-Lane Ring network (Figure 9(b)) is a complex ring road network consisting of multiple lanes of traffic, with the objective being to stabilize the flow of all traffic within the network. The observation for this network includes the general observation for Flow networks, as well as the current lane index for each vehicle. The network holds 21 human vehicle and 1 ego vehicle. A control policy must apply acceleration commands and lane changing commands (a continuous value from -1 to 1) to the ego vehicle in order to stabilize the flow of traffic between both lanes. The maximum step in this domain is set to 1500.

Flow Figure-8 Network: The Figure-8 Network (Figure 9(c)) is a complex Flow domain as it contains a road section where the vehicles will cross-over, which is in the center of the figure-8. The control policy must consider this congestion point when controlling the ego vehicle, applying acceleration commands to the autonomous agent in order to stabilize the flow. The network holds 13 human vehicles besides the ego vehicle. Here the maximum time step is 1500.

C. Hyperparameters

All methods are trained using SAC with the same structure of critic network, which consists of an MLP with 2 hidden layers of 256 units. The buffer size is always set to 1000000, and the discount factor γ is always set to 0.99. In the Single-Lane Ring network, the training steps are 100000, otherwise the training steps are 500000 for all the methods. The soft update coefficient, τ, is set to 0.005 for ICCT-static in Inverted Pendulum, and set to 0.01 in all other domains. We display the learning rates, batch sizes, and network sizes used for the methods discussed in the paper across each domain in Tables III-VIII.

TABLE III: Hyperparameters in Inverted Pendulum

Hyperparameter	Learning Rate	Batch Size	Network Size
ICCT-complete	3×10^{-4}	1024	2 leaves
ICCT-1-feature	6×10^{-4}	1024	8 leaves
ICCT-2-feature	5×10^{-4}	1024	4 leaves
ICCT-3-feature	5×10^{-4}	1024	2 leaves
ICCT-static	5×10^{-4}	1024	32 leaves
ICCT-L1-sparse	3×10^{-4}	256	4 leaves
CDDT	3×10^{-4}	1024	2 leaves
CDDT-controllers	3×10^{-4}	1024	2 leaves
MLP-Max	3×10^{-4}	1024	[256, 256]
MLP-U	3×10^{-4}	1024	[8, 8]
MLP-L	3×10^{-4}	1024	[6, 6]

TABLE IV: Hyperparameters in Lunar Lander

Hyperparameter	Actor Learning Rate	Critic Learning Rate	Batch Size	Network Size
ICCT-complete	5×10^{-4}	5×10^{-4}	256	8 leaves
ICCT-1-feature	5×10^{-4}	5×10^{-4}	256	8 leaves
ICCT-2-feature	5×10^{-4}	5×10^{-4}	256	8 leaves
ICCT-3-feature	5×10^{-4}	5×10^{-4}	256	8 leaves
ICCT-static	5×10^{-4}	3×10^{-4}	256	32 leaves
ICCT-L1-sparse	5×10^{-4}	5×10^{-4}	256	8 leaves
CDDT	5×10^{-4}	3×10^{-4}	256	8 leaves
CDDT-controllers	5×10^{-4}	3×10^{-4}	256	8 leaves
MLP-Max	3×10^{-4}	3×10^{-4}	256	[256, 256]
MLP-U	3×10^{-4}	3×10^{-4}	256	[10, 10]
MLP-L	3×10^{-4}	3×10^{-4}	256	[6, 6]

TABLE V: Hyperparameters in Lane-Keeping

Hyperparameter	Learning Rate	Batch Size	Network Size
ICCT-complete	3×10^{-4}	1024	16 leaves
ICCT-1-feature	3×10^{-4}	1024	16 leaves
ICCT-2-feature	3×10^{-4}	1024	16 leaves
ICCT-3-feature	3×10^{-4}	1024	16 leaves
ICCT-static	2×10^{-4}	1024	16 leaves
ICCT-L1-sparse	3×10^{-4}	1024	16 leaves
CDDT	3×10^{-4}	256	16 leaves
CDDT-controllers	3×10^{-4}	512	16 leaves
MLP-Max	3×10^{-4}	256	[256, 256]
MLP-U	3×10^{-4}	256	[14, 14]
MLP-L	3×10^{-4}	256	[6, 6]

TABLE VI: Hyperparameters in Sing-Lane Ring Network

Hyperparameter	Learning Rate	Batch Size	Network Size
ICCT-complete	5×10^{-4}	1024	16 leaves
ICCT-1-feature	5×10^{-4}	1024	16 leaves
ICCT-2-feature	5×10^{-4}	1024	16 leaves
ICCT-3-feature	5×10^{-4}	1024	16 leaves
ICCT-static	5×10^{-4}	1024	16 leaves
ICCT-L1-sparse	5×10^{-4}	1024	16 leaves
CDDT	5×10^{-4}	1024	16 leaves
CDDT-controllers	5×10^{-4}	1024	16 leaves
MLP-Max	3×10^{-4}	1024	[256, 256]
MLP-U	3×10^{-4}	1024	[12, 12]
MLP-L	3×10^{-4}	1024	[3, 3]

TABLE VII: Hyperparameters in Multi-Lane Ring Network

Hyperparameter	Learning Rate	Batch Size	Network Size
ICCT-complete	5×10^{-4}	1024	16 leaves
ICCT-1-feature	5×10^{-4}	1024	16 leaves
ICCT-2-feature	6×10^{-4}	1024	16 leaves
ICCT-3-feature	5×10^{-4}	1024	16 leaves
ICCT-static	5×10^{-4}	1024	16 leaves
ICCT-L1-sparse	5×10^{-4}	1024	16 leaves
CDDT	5×10^{-4}	1024	16 leaves
CDDT-controllers	5×10^{-4}	1024	16 leaves
MLP-Max	5×10^{-4}	1024	[256, 256]
MLP-U	5×10^{-4}	1024	[32, 32]
MLP-L	5×10^{-4}	1024	[3, 3]

TABLE VIII: Hyperparameters in Figure-8 Network

Hyperparameter	Learning Rate	Batch Size	Network Size
ICCT-complete	5.5×10^{-4}	1024	16 leaves
ICCT-1-feature	6×10^{-4}	1024	16 leaves
ICCT-2-feature	6×10^{-4}	1024	16 leaves
ICCT-3-feature	7×10^{-4}	1024	16 leaves
ICCT-static	5×10^{-4}	1024	16 leaves
ICCT-L1-sparse	5×10^{-4}	1024	16 leaves
CDDT	5×10^{-4}	1024	16 leaves
CDDT-controllers	5×10^{-4}	1024	16 leaves
MLP-Max	5×10^{-4}	1024	[256, 256]
MLP-U	5×10^{-4}	1024	[20, 20]
MLP-L	5×10^{-4}	1024	[3, 3]