SPLIT-PO: Sparse Piecewise-Linear Interpretable Tree Policy Optimization

An Interpretable and Differentiable Framework for Sparse-Tree Policy Optimization

Ernesto Hellouin de Menibus[1]

Supervisor(s): Anna Lukina[1]**, Daniël Vos** [1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Ernesto Hellouin de Menibus
Final project course: CSE3000 Research Project
Thesis committee: Anna Lukina, Daniël Vos, Luciano Cavalcante Siebert

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

Deep reinforcement learning has shown strong performance in continuous control tasks, but its reliance on deep neural networks (DNNs) hinders interpretability, limiting deployment in safety-critical domains. While recent approaches using differentiable decision trees improve transparency, they often rely on fixed structures that limit flexibility and lead to unnecessarily complex policies.

We propose **SPLIT-PO** (Sparse Piecewise-Linear Interpretable Tree Policy Optimization), a novel framework that learns sparse, interpretable decision trees with linear leaf controllers and dynamically adaptive structure. SPLIT-PO introduces learnable gating and regularization to prune uninformative branches during training, enabling compact tree policies to emerge automatically. It maintains end-to-end differentiability and integrates crispification within the training loop, building on prior interpretable methods like ICCT.

Experiments on standard continuous control benchmarks show that SPLIT-PO matches neural network performance (e.g., 285 vs. 287 average reward on Lunar Lander) while producing trees with 100–1000× fewer parameters and as few as 1–3 leaf nodes. Additionally, we prove SPLIT-PO is a universal function approximator, offering neural-level expressivity in an interpretable form. Although it requires more samples to converge, SPLIT-PO provides a promising foundation for transparent and verifiable reinforcement learning.

1 Introduction

Reinforcement learning (RL) enables agents to learn through trial-and-error interactions with their environment, guided by rewards. In continuous-action settings, policies are often represented using deep neural networks (DNNs), trained via algorithms like Proximal Policy Optimization (PPO) [9] and Deep Deterministic Policy Gradient (DDPG) [6].

However, DNNs function as black boxes, posing challenges in domains where interpretability and verifiability are essential, such as healthcare, robotics, and autonomous systems.

To address this, recent work has explored decision tree-based models as more transparent policy representations. Some approaches distill trained policies into trees post hoc, while others directly train trees within the RL loop [13].

Classical decision trees are non-differentiable due to discrete splits, preventing gradient-based optimization. Differentiable decision trees (DDTs) [12] address this by replacing hard decisions with soft, differentiable splits, allowing end-to-end training.

However, converting soft trees to hard trees after training, known as crispification, can degrade performance due to misalignment between training and inference.

Interpretable Continuous Control Trees (ICCTs) [7] address this limitation by incorporating crispification directly into the training loop. During the forward pass, ICCTs evaluate actions using a crispified (hard-split) decision tree, ensuring that the agent's behavior remains interpretable throughout training. For the backward pass, gradients are computed using the soft (differentiable) version of the tree, enabling effective gradient-based optimization. This approach preserves interpretability during training and inference while maintaining end-to-end differentiability.

Nonetheless, ICCT still requires the tree structure to be fixed in advance, which may limit expressiveness and introduce structural bias.

This work introduces a method for learning **piecewise-linear differentiable decision trees with sparse structure** in **continuous-action** RL environments. Unlike classical trees with constant leaf outputs, our model uses linear functions at the leaves, allowing more expressive policies.

Instead of fixing the structure, we introduce a learnable gating mechanism that prunes uninformative branches during training, enabling compact, sparse trees without manual tuning, while maintaining end-to-end differentiability and interpretability.

We propose **Sparse Piecewise-Linear Interpretable Tree Policy Optimization (SPLIT-PO)**, a method that learns interpretable, sparse trees for continuous-action tasks using a refined crispification approach inspired by ICCT.

Experiments show that SPLIT-PO matches neural network performance on simple tasks like Inverted Pendulum (return of 1000), and approaches baseline performance on harder tasks like Lunar Lander Continuous (returns 130–285 vs. 287 for a neural net), while offering compact, human-readable policies.

These results demonstrate SPLIT-PO's ability to maintain strong performance while significantly improving interpretability.

The remainder of this paper is organized as follows: Section 2 reviews DDTs, ICCTs, and formalizes the research problem. Section 3 introduces our method, **SPLIT-PO**, detailing the architecture and training procedure. Section 4 presents our experimental setup and results. Section 5 discusses key findings and limitations. Section 6 addresses responsible research practices. Section 7 concludes with future directions.

2 Problem Description

2.1 Reinforcement Learning

Reinforcement learning (RL) is a framework in which an agent learns to make decisions by interacting with an environment to maximize cumulative rewards. At each time step t, the agent observes a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$ according to a policy $\pi_\theta(s_t)$, receives a reward r_t, and transitions to the next state s_{t+1}.

The objective is to learn a policy π_θ that maximizes the expected return, defined as the discounted sum of future rewards:

$$J(\pi_\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{1}$$

where $\gamma \in [0, 1)$ is the discount factor.

In this work, we employ an actor-critic architecture where the policy (actor) π_θ is trained alongside a value function (critic) $Q_\phi(s, a)$, parameterized by ϕ. The critic estimates the action-value function and is updated by minimizing the Bellman error:

$$y_t = r_t + \gamma Q_\phi(s_{t+1}, \pi_\theta(s_{t+1})), \tag{2}$$

$$\mathcal{L}_{\text{critic}} = \left(y_t - Q_\phi(s_t, a_t)\right)^2. \tag{3}$$

The actor is trained to maximize the expected Q-value of its actions. In the basic deterministic setting used by Deep Deterministic Policy Gradient (DDPG) [6], this is done by minimizing:

$$\mathcal{L}_{\text{actor}} = -Q_\phi(s, \pi_\theta(s)). \tag{4}$$

In addition to DDPG, we also evaluate our model using Soft Actor-Critic (SAC) [5], which augments the objective with an entropy term to encourage exploration and policy robustness. In SAC, the actor loss becomes:

$$\mathcal{L}_{\text{actor}} = \mathbb{E}_{s_t \sim \mathcal{D}}\left[\alpha \mathcal{H}(\pi_\theta(\cdot|s_t)) - Q_\phi(s_t, a_t)\right], \tag{5}$$

where α is the temperature parameter controlling the trade-off between exploration (entropy) and exploitation (Q-value maximization).

In both settings, the actor and critic are implemented using differentiable function approximators—in our case, sparse interpretable decision trees trained end-to-end using gradient-based optimization.

2.2 Decision Trees in Reinforcement Learning

Decision trees are widely regarded as interpretable function approximators due to their hierarchical and rule-based structure [2]. In RL they have been used to approximate and learn optimal policies. These approaches can be split into two types, distillation and direct policy learning.

Distillation techniques usually try to fit a tree to a well performing black box policy, whereas direct policy learning try to build the tree as the agent interacts with the environment.

Previous works such as VIPER [1] perform post-hoc imitation learning, extracting a decision tree policy from a deep Q-network by imitating the network's behavior. Nonetheless, this depends on having an already trained teacher (DQN) model and does not directly optimize the tree.

Policy Tree [4] introduces a direct policy learning algorithm which trains decision trees with linear models in leaves using policy gradients. That said, Policy Tree relies on a greedy and irreversible structure growth process, which can lead to suboptimal splits and limited flexibility during learning.

Similarly, Conservative Q-Improvement [8] greedily expands a decision tree to improve the Q-values, but suffers of the same fixed structure issue.

Silva et al. [11] introduced Differentiable Decision Trees (DDTs), which use soft splits to enable gradient-based training within the RL loop. This makes them compatible with policy optimization methods while preserving a tree-based structure. However, when crispification is applied after training, converting soft splits to hard decisions, performance often degrades significantly.

To overcome this Interpretable Continuous Control Trees (ICCTs) [7] were introduced which significantly improve the performance of the resulting interpretable tree. A key limitation of this approach is that the tree structure must be predefined, which restricts flexibility, may introduce bias if the structure does not align well with the task, and can lead to unnecessarily large trees. This constraint can result in suboptimal representations, where equally performant but sparser trees could exist. These differentiable approaches, particularly DDTs and ICCTs, form the foundation for our work, and in the following sections we provide a detailed overview of their mechanisms and limitations as a basis for our proposed extension.

2.3 Differentiable Decision Trees (DDTs)

Differentiable Decision Trees (DDTs) are a soft, continuous generalization of traditional decision trees that can be optimized using gradient-based methods like backpropagation. The key innovation in DDTs is replacing **hard threshold splits** at internal nodes with **soft decisions**, typically modeled by sigmoid functions. This enables full differentiability, making DDTs well-suited for integration into reinforcement learning pipelines.

Visual Comparison: Hard vs. Soft Trees

Figure 1 illustrates the difference between a traditional decision tree with hard splits (left) and a differentiable decision tree using soft splits (right). In the **hard tree**, each internal node makes a binary decision (e.g., $x < 0.5$), routing the input deterministically down a single path to a leaf. In contrast, the **soft tree** uses sigmoid-based gating functions (e.g., $\sigma(x-0.5)$), which probabilistically route the input down multiple paths. This allows all leaves to contribute to the final output, weighted by their respective path probabilities.

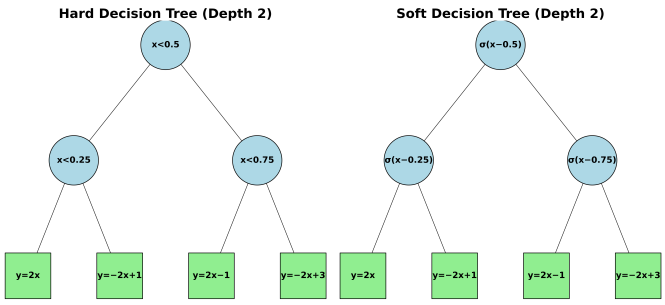

Figure 1: Comparison of hard and soft decision trees of depth 2. Left: traditional tree with hard threshold splits. Right: soft tree using sigmoid-based gating functions.

The mathematical formulation for a soft split at node i is:

$$s_i(x) = \sigma(\alpha_i(\mathbf{w}_i^\top \mathbf{x} - t_i)), \tag{6}$$

where $\mathbf{w}_i$ is the weight vector, t_i is the threshold, α_i controls the sharpness of the transition, and σ is the sigmoid function. As $\alpha_i \to \infty$, the soft split approximates a hard threshold.

The probability of reaching leaf ℓ for input x is computed as:[1]

$$P_\ell(x) = \prod_{(i,d_i)\in \text{path}(\ell)} \begin{cases} s_i(x) & \text{if } d_i = 0 \\ 1 - s_i(x) & \text{if } d_i = 1 \end{cases}, \tag{7}$$

and the overall output is a weighted combination of all leaf predictions:

$$\pi_\theta(x) = \sum_{\ell \in \mathcal{L}} P_\ell(x) \cdot y_\ell(x), \tag{8}$$

where $y_\ell(x)$ is the output of leaf ℓ, typically a constant or a linear model.

Output Behavior: Sharp vs. Smooth

The effect of soft versus hard decisions is also reflected in the model's output. Figure 2 compares the output curves of a hard decision tree and a soft decision tree on a 1D input. The hard tree (dashed yellow) exhibits sharp discontinuities at split boundaries, while the soft tree (solid orange) produces a smoother function due to the continuous nature of its gating functions.

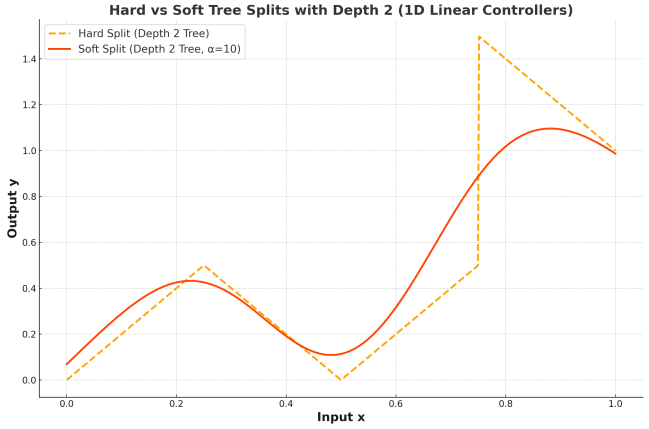

Figure 2: Output comparison of a hard vs. soft decision tree of depth 2. The soft tree provides a smoother approximation, which is beneficial for gradient-based optimization but often less interpretable.

Trade-offs and Crispification

While soft trees are fully differentiable and thus trainable via backpropagation or policy gradients, they sacrifice interpretability. To recover interpretability, a post-training *crispification* step can convert soft decisions into hard thresholds. However, as seen in Figure 2, this often leads to reduced performance due to the mismatch between the optimized soft structure and the non-differentiable hard splits.

[1]This expression computes a soft probability of reaching a leaf node by taking the product of the sigmoid gate outputs $s_i(x)$ (or their complements) along the path. It enables backpropagation through all possible paths during training.

2.4 Interpretable Continuous Control Trees (ICCTs)

Interpretable Continuous Control Trees (ICCTs) extend differentiable decision trees by introducing an online crispification procedure that enables consistent use of a hard decision tree structure during both training and inference. The key innovation of ICCTs is that the forward pass uses crisp decisions to preserve interpretability, while the backward pass computes gradients through a soft, differentiable surrogate, allowing effective end-to-end optimization using gradient-based methods.

To enforce single-feature decision splits, ICCTs use a softmax-based selection mechanism over the input dimensions. For each internal node i, the feature importance is computed using a temperature-scaled softmax:

$$\mathbf{z}_i = \text{softmax}\left(\frac{|\mathbf{w}_i|}{\tau}\right), \tag{9}$$

where $\mathbf{w}_i$ is the feature selector weight vector and τ is the temperature controlling the sharpness of the selection.

The most important feature is selected using a differentiable straight-through estimator, which enables gradient flow through the non-differentiable $\arg\max$ operation. Specifically, a hard one-hot vector is used in the forward pass to enforce discrete feature selection, while the backward pass uses the gradient of the softmax output. This is implemented as:

$$\hat{\mathbf{z}}_i = \text{onehot}(\arg\max_j z_{i,j}) + (\mathbf{z}_i - \text{detach}(\mathbf{z}_i)). \tag{10}$$

Here, the `detach` operation prevents gradients from flowing through $\mathbf{z}_i$ during backpropagation, ensuring that only the soft softmax component contributes to the gradient update, while the forward computation relies on the discrete one-hot vector.

The selected feature is then extracted from the input vector:

$$x_i = \hat{\mathbf{z}}_i^\top \mathbf{x}, \tag{11}$$

resulting in either a soft or hard selection of a single feature, depending on whether $\hat{\mathbf{z}}_i$ is relaxed or one-hot.

Each internal node computes a soft decision value based on the selected feature and a learned threshold t_i:

$$s_i(x) = \sigma(\alpha_i(x_i - t_i)), \tag{12}$$

where α_i is a sharpness parameter, and σ is the sigmoid function. To preserve interpretability during training, ICCTs use a crispified version of this split function in the forward pass:

$$\hat{s}_i(x) = \sigma(x_i - t_i) + (s_i(x) - \text{detach}(s_i(x))), \tag{13}$$

This formulation uses a straight-through estimator, where the forward pass behaves like a hard threshold function (using the unscaled sigmoid $\sigma(x_i - t_i)$), while the backward pass allows gradients to flow through the soft surrogate $s_i(x)$. The `detach` operation blocks gradients through the crispified output, ensuring that only the soft path contributes to optimization.

Each leaf node d of the tree contains a sparse linear controller, designed to remain interpretable by limiting the number of active input features. This sparsity is enforced via a

differentiable top-k feature selector, implemented by repeatedly applying a straight-through one-hot approximation to a learned importance vector $\boldsymbol{\theta}_d$:

$$\mathbf{u}_d = \sum_{j=1}^{k} \text{DIFFARGMAX}_j(|\boldsymbol{\theta}_d|), \tag{14}$$

where each DIFFARGMAX_j selects the j-th most important feature using a temperature-scaled softmax followed by an $\arg\max$ operation, with masking to avoid selecting the same index multiple times. This approximates a discrete top-k selection in a fully differentiable manner. A straight-through estimator is used to enable gradient flow through the one-hot selections, producing a k-hot binary mask vector $\mathbf{u}_d \in \{0,1\}^m$ that determines which features are active in the linear controller.

The output of the controller is given by applying the selected mask to both the learned weights $\boldsymbol{\beta}_d$ and the per-feature biases $\boldsymbol{\phi}_d$:

$$\ell_d^*(\mathbf{x}) = (\mathbf{u}_d \circ \boldsymbol{\beta}_d)^\top (\mathbf{u}_d \circ \mathbf{x}) + \mathbf{u}_d^\top \boldsymbol{\phi}_d, \tag{15}$$

where $\circ$ denotes the element-wise (Hadamard) product. This operation zeroes out all components except those selected by the sparse mask $\mathbf{u}_d \in \{0,1\}^m$.

Equivalently, the output can be written in expanded form as:

$$\ell_d^*(\mathbf{x}) = \sum_{j=1}^{m} u_{d,j} \cdot \beta_{d,j} \cdot x_j + \sum_{j=1}^{m} u_{d,j} \cdot \phi_{d,j}, \tag{16}$$

where $u_{d,j}$ denotes the j-th entry in the sparse top-k mask vector $\mathbf{u}_d$, and m is the dimensionality of the input. This formulation highlights that only the top-k selected features contribute to the output, preserving both sparsity and interpretability.

Unlike DDTs, which compute a weighted average over all leaves, ICCTs use the crispified split values $\hat{s}_i(x)$ to deterministically follow a single path from the root to a leaf during the forward pass. Thus, the final policy output is given by:

$$\pi_\theta(x) = \ell_{d(x)}^*(\mathbf{x}), \tag{17}$$

where $d(x)$ is the index of the leaf reached by the hard decision path, and $\ell_{d(x)}^*$ is the sparse linear controller at that leaf. This crisp path selection makes the policy's behavior interpretable at every step. During the backward pass, however, gradients are propagated through the corresponding soft decisions $s_i(x)$ and soft selectors $\mathbf{z}_i$, allowing the model to be trained via standard gradient descent.

By combining interpretability in the forward pass with differentiability in the backward pass, ICCTs achieve strong performance while maintaining transparency. A major limitation, however, is that the tree structure must be fixed before training. This constraint may limit representational capacity, introduce bias if the chosen structure does not align well with the task and create unnecessarily large trees.

2.5 Research Problem

Recent work in interpretable reinforcement learning has shown that differentiable decision trees like ICCTs can combine strong performance with transparency by aligning training and inference policies through gradient-based optimization.

However, these models rely on fixed tree structures (e.g., depth, branching), which introduces structural bias. This rigidity can lead to underfitting or overfitting and limits adaptability, particularly in continuous control settings where task complexity varies across states.

This raises the central research question:

> *How can we design a reinforcement learning framework that allows differentiable piecewise-linear decision trees to adapt their structure dynamically during training while encouraging sparsity?*

Solving this requires a method that is end-to-end differentiable, interpretable at inference, and capable of learning sparse, adaptable trees within the RL loop.

3 Sparse Piecewise-Linear Interpretable Tree Policy Optimization (SPLIT-PO)

To overcome the structural rigidity of existing differentiable tree policies, we propose Sparse Piecewise-Linear Interpretable Tree Policy Optimization (SPLIT-PO). SPLIT-PO augments the standard differentiable decision tree, specifically ICCT [7] with a dynamic gating mechanism that enables the model to learn not only the parameters of the splits and leaf outputs, but also the active structure of the tree during training. This is achieved through a bypass strategy that softly disables (or enables) branches, allowing the tree to prune and become sparse in a data-driven and interpretable way.

3.1 Dynamic Node Gating and Tree Structure

To enable a decision tree to dynamically adjust its structure during training, we introduce a learnable *gating mechanism* at each internal node. This mechanism determines whether a node actively participates in decision-making or is bypassed. Unlike standard decision trees, where the structure is fixed a priori, our approach allows the agent to prune or grow the tree during training in a fully differentiable manner.

Each internal node i is assigned a gating parameter $g_i \in \mathbb{R}$, which is transformed into a soft gate using a sigmoid function. To maintain a hard, interpretable structure during the forward pass while preserving gradient flow during backpropagation, we use the straight-through estimator:

$$\hat{g}_i = \text{step}(\sigma(g_i)) + (\sigma(g_i) - \text{detach}(\sigma(g_i))), \tag{18}$$

where $\sigma(\cdot)$ is the sigmoid function, and $\text{step}(\cdot)$ is a binary threshold function (e.g., $\text{step}(x) = 1$ if $x \geq 0.5$, otherwise 0). The $\text{detach}(\cdot)$ operator ensures that gradients only propagate through the soft sigmoid term. As a result: - When $\hat{g}_i \approx 1$, the node is *active*, and its decision is used. - When $\hat{g}_i \approx 0$, the node is *inactive*, and its left/right decision is bypassed, always routing to the right child.

The key idea is that **inactive nodes are skipped**, effectively allowing the model to prune unnecessary subtrees or defer splitting until useful in order to promote sparsity.

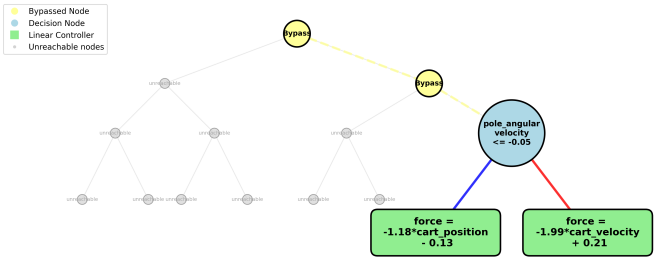

Figure 3: Learned decision tree for the Inverted Pendulum (SPLIT-PO, k=1). Yellow nodes are bypassed, blue nodes represent decision splits, and green leaves are linear controllers with their formulas. Gray nodes indicate unreachable states during training.

Figure 3 shows an example of a learned decision tree]for the Inverted Pendulum task (SPLIT-PO, k=1). The tree highlights how certain state nodes are bypassed and some become unreachable. The two yellow nodes have a $\hat{g} \approx 0$ so their decision is bypassed.

This gating modifies the path probability computation used to determine which leaf is active for a given input x. Let path(ℓ) denote the set of internal nodes (i, d_i) along the path to leaf ℓ, where $d_i \in \{0, 1\}$ indicates a left (0) or right (1) branch. The probability of reaching leaf ℓ is defined as:

$$P_\ell(x) = \prod_{(i,d_i)\in \text{path}(\ell)} \begin{cases} \hat{g}_i \cdot \hat{s}_i(x) & \text{if } d_i = 0 \\ (1 - \hat{g}_i) + \hat{g}_i \cdot (1 - \hat{s}_i(x)) & \text{if } d_i = 1 \end{cases}, \tag{19}$$

where $\hat{s}_i(x)$ is the crispified soft decision function (as defined in ICCT). This formulation ensures: - If a node is *inactive* ($\hat{g}_i \approx 0$), the left decision is never taken; the path always proceeds to the right. - If a node is *active* ($\hat{g}_i \approx 1$), the split operates as in ICCT, and either left or right is selected based on $\hat{s}_i(x)$.

The final policy output is computed as a weighted sum over all leaves, where each leaf contributes according to its path probability $P_\ell(x)$. To ensure bounded continuous actions (e.g., in $[-1, 1]$), the output is passed through a $\tanh$ activation:

$$\pi_\theta(x) = \tanh\left(\sum_{\ell\in\mathcal{L}} P_\ell(x) \cdot y_\ell(x)\right), \tag{20}$$

where $y_\ell(x)$ is the output of leaf ℓ, typically a learned scalar or vector. The output is then scaled by the range of the action space [2]. Despite the use of a soft sum formulation, the gating and crispified decision functions produce path probabilities of zero for all but one leaf, rendering their contributions to the final output null as they are multiplied by zero.

This design introduces a natural way for the model to learn a sparse structure: early in training, many nodes may remain active, effectively behaving like a full binary tree. As learning progresses, nodes can "turn off" when they don't help improve policy performance allowing the tree to become more sparse.

By embedding this gating mechanism into the path computation and ensuring full differentiability through straight-through estimators, SPLIT-PO enables interpretable tree-based policies that adapt their depth and complexity in response to the task, something fixed-structure methods like ICCT cannot achieve.

3.2 End-to-End Training with Structural Adaptation

We train SPLIT-PO using standard actor-critic reinforcement learning methods, either Soft Actor-Critic (SAC) [5] or Deep Deterministic Policy Gradient (DDPG) [6], depending on the environment. In both cases, the actor is represented by our differentiable decision tree policy, and the critic is a separate value network that guides learning.

Our main contribution lies in augmenting this framework with a structural regularization term that encourages sparsity in the tree. Each internal node has a learnable gating parameter that determines whether it participates in decision-making. During training, we penalize active gates using a sigmoid-based relaxation:

$$\mathcal{L}_{\text{reg}} = \lambda \sum_{i\in\mathcal{N}} \sigma(g_i),$$

where $\mathcal{N}$ is the set of internal nodes and λ controls the strength of the regularization. This encourages the model to deactivate unnecessary nodes, allowing it to learn both an effective policy and a compact, interpretable tree structure simultaneously.

We compute gradients end-to-end using straight-through estimators, enabling optimization over both the policy and its structure. This joint training leads to sparse policies tailored to the task, balancing performance and interpretability.

3.3 Theoretical Analysis: Expressivity and Approximation Power

We now show that our proposed model, SPLIT-PO, is a *universal function approximator* (UFA). This means that, given sufficient capacity, SPLIT-PO can approximate any continuous function defined on a compact domain to arbitrary precision. Our proof strategy is similar to the ICCT [7] strategy which uses the classical approach of Cybenko [3], adapted to the context of decision tree structures with gating and crispification mechanisms.

Theorem (Cybenko, 1989) [3]: Let $\sigma : \mathbb{R} \to \mathbb{R}$ be a bounded, measurable, and discriminatory function. Then any function $f \in C([0, 1]^n)$ can be approximated arbitrarily well in the L^∞ norm by a finite sum of the form:

$$G(x) = \sum_{j=1}^{N} \alpha_j \cdot \sigma(w_j^T x + b_j) \tag{21}$$

[2] If the environment's action space is bounded in $[a_{\min}, a_{\max}]$, then the scaled action is computed as $\frac{1}{2}(a_{\max} - a_{\min}) \cdot \tanh(\cdot) + \frac{1}{2}(a_{\max} + a_{\min})$. This ensures the action remains within the valid range while allowing the model to output unbounded values before the $\tanh$ squashing.

This result is known as the Universal Approximation Theorem. It states that a single-hidden-layer neural network with a suitable activation function σ can approximate any continuous function on a compact domain, given enough units.

We briefly recall two key conditions from the theorem:

- **Sigmoidal:** A function σ is sigmoidal if it is bounded, continuous, and non-constant, typically resembling a smooth step (e.g., the logistic sigmoid).
- **Discriminatory:** A function σ is discriminatory if the set $\{\sigma(w^T x+b) \mid w \in \mathbb{R}^n, b \in \mathbb{R}\}$ is dense in $C([0,1]^n)$.

We show below that the components of SPLIT-PO satisfy these conditions and therefore inherit the universal approximation property.

Lemma 1: The crispified split function $\hat{s}_i(x)$ used in SPLIT-PO is sigmoidal.

Proof: Each internal node in SPLIT-PO selects a single input feature x_i and computes a decision using a sigmoid function $\sigma(\alpha_i(x_i - t_i))$, where α_i is a steepness parameter and t_i is a learned threshold. The crispified version used in the forward pass is:

$$\hat{s}_i(x) = \text{step}(\sigma(\alpha_i(x_i - t_i))) + (\sigma(\alpha_i(x_i - t_i)) - \text{detach}(\sigma(\alpha_i(x_i - t_i)))) \quad (22)$$

This function behaves like an indicator (step) in the forward pass, but retains gradient information in the backward pass. Since the sigmoid function is bounded and continuous, and its step approximation approaches a jump from 0 to 1, this function meets the sigmoidal condition required by Cybenko's theorem.

Lemma 2: The gating function $\hat{g}_i$ in SPLIT-PO is discriminatory.

Proof: Each internal node has an associated gating parameter $g_i \in \mathbb{R}$, transformed into a binary decision via:

$$\hat{g}_i = \text{step}(\sigma(g_i)) + (\sigma(g_i) - \text{detach}(\sigma(g_i)))$$

The forward behavior is equivalent to an indicator function, which outputs 0 or 1 depending on whether $\sigma(g_i)$ crosses 0.5. In the backward pass, the gradient flows through the sigmoid term. The full gating function is thus a jump-discontinuous function that is bounded, measurable, and nonconstant. By Lemma 1 in [3], any such function is discriminatory. Furthermore, Selmic and Lewis [10] extended the results of [3] to allow jump-discontinuous activation functions. Hence, $\hat{g}_i$ satisfies the discriminatory condition required for universal approximation.

Theorem: SPLIT-PO is a universal function approximator.

Proof: The output of SPLIT-PO is a sum over leaf outputs $y_\ell(x)$, each weighted by a path probability $P_\ell(x)$:

$$\pi_\theta(x) = \tanh\left(\sum_{\ell \in L} P_\ell(x) \cdot y_\ell(x)\right)$$

The path probabilities $P_\ell(x)$ are computed as a product of crispified split functions $\hat{s}_i(x)$ and gating terms $\hat{g}_i$. These can be viewed as indicator-like components that partition the input space into axis-aligned regions. Since both $\hat{g}_i$ and $\hat{s}_i(x)$ are discriminatory, and the leaf outputs $y_\ell(x)$ can be linear or constant functions, the overall model approximates a piecewise-linear surface. This class of functions is dense in $C([0,1]^n)$, as shown by [3] and extended by [10] for jump-continuous activations.

Therefore, SPLIT-PO is dense in $C([0,1]^n)$, and is a universal function approximator.

This shows that given **sufficient nodes**, SPLIT-PO not only maintains the theoretical expressivity of neural networks but does so within an interpretable, tree-based structure.

4 Experimental Setup and Results

4.1 Experimental Setup

To evaluate our method, we focus on continuous control environments from OpenAI Gym: **Inverted Pendulum** and **Lunar Lander Continuous**. The implementation was done in `PyTorch`. All experiments were run on a MacBook Pro M4 with 36GB of ram. GPU acceleration was not used. For fairness, we train all models using the same actor-critic framework.

Hyperparameters such as learning rates, discount factor (γ), and network sizes are kept consistent across methods unless noted and can be found in the appendix. We report mean episodic rewards over 50 evaluation seeds after training.

4.2 ICCT and NN

We were unable to run the original ICCT implementation [7], so we reimplemented it by setting all gating variables $g = 1$ and freezing their gradients. We refer to this version as ICCT*, which matches the original ICCT algorithm except for minor differences in the training loop.

For our neural network (NN) baseline, we used a standard fully connected architecture common in continuous-control RL. Full architectural and training details are provided in Appendix C. This NN serves as a strong, high-capacity but non-interpretable baseline.

4.3 Results

We compare the performance of various interpretable and non-interpretable models in terms of average reward and the number of leaf controller nodes (for tree-based models). The SPLIT-PO and ICCT models are evaluated under different feature selection settings, using 1, 2, 3, or k features per leaf to explore varying levels of interpretability. ICCT, and MLP baselines are included for comparison. Tables 1 and 2 show the performance of our model compared to other similar models in the Inverted Pendulum and Lunar Lander environments.

Table 1: Comparison of Episode Reward, Tree Complexity, and Parameter Count on Inverted Pendulum (Max Reward 1000)

Model	Features	Reward	Leaves	Parameters
SPLIT-PO	1	$\mathbf{958.22 \pm 40.32}$	2	65
SPLIT-PO	2	$\mathbf{1000 \pm 0}$	2	73
SPLIT-PO	3	$\mathbf{1000 \pm 0}$	2	81
SPLIT-PO	k	$\mathbf{1000 \pm 0}$	2	41
ICCT*	1	$\mathbf{1000 \pm 0}$	8	58
ICCT*	2	$\mathbf{1000 \pm 0}$	4	30
ICCT*	3	$\mathbf{1000 \pm 0}$	4	34
ICCT*	k	$\mathbf{1000 \pm 0}$	2	16
MLP	full	$\mathbf{1000 \pm 0}$	n/a	67,586

Table 2: Comparison of Episode Reward, Tree Size, and Parameter Count on Lunar Lander (200+ is considered solved, max reward is just over 300)

Model	Features	Reward	Leaves	Parameters
SPLIT-PO	1	$\mathbf{130.40 \pm 59.23}$	3	109
SPLIT-PO	2	$\mathbf{245.00 \pm 71.07}$	1	125
SPLIT-PO	3	$\mathbf{248.35 \pm 57.41}$	3	141
SPLIT-PO	k	$\mathbf{285.20 \pm 21.03}$	1	221
ICCT*	1	$\mathbf{107.37 \pm 120.32}$	8	102
ICCT*	2	$\mathbf{77.56 \pm 142.36}$	8	118
ICCT*	3	$\mathbf{257.22 \pm 33.00}$	8	134
ICCT*	k	$\mathbf{279.00 \pm 18.44}$	8	214
MLP	full	$\mathbf{287.43 \pm 14.23}$	n/a	69,124

The results in Tables 1 and 2 clearly illustrate the strengths of SPLIT-PO in balancing performance, interpretability, and parameter efficiency. On the Inverted Pendulum task, SPLIT-PO matches or exceeds ICCT* performance across all feature configurations, while maintaining significantly smaller trees (as few as 2 leaves) and comparable or fewer learned parameters. In particular, SPLIT-PO achieves perfect performance (1000 ± 0) even with very compact trees and modest feature counts. For $k = 1$ while SPLIT-PO performs worse than ICCT* it uses 4 times less leaves making it much more interpretable and verifiable while having near perfect performance.

The Lunar Lander task highlights a similar trend but under more challenging conditions. SPLIT-PO consistently outperforms ICCT* when using fewer than 3 features, and at k features, it achieves a reward of 285.20, approaching the neural network baseline. Notably, SPLIT-PO does so with dramatically fewer parameters—just 221 compared to over 69,000 in the MLP—while also using trees with as few as 1–3 leaves. In contrast, ICCT* requires a fixed 8-leaf tree in all cases, leading to higher complexity and parameter count without a corresponding performance gain.

These results support the central claim of this work: SPLIT-PO provides interpretable and sparse tree policies that remain competitive with both fixed-tree methods like ICCT* and black-box baselines like MLPs, while using vastly fewer parameters and simpler structures.

5 Discussion

5.1 Interpretability vs Performance

Our results highlight the expected trade-off between interpretability and performance. Trees using fewer features are easier to understand but perform worse due to limited expressiveness in the linear controllers.

SPLIT-PO enforces interpretability via top-k feature selection at each leaf. Small k improves transparency by restricting each controller to a few inputs, but this limits the model's ability to capture complex state dependencies, especially in high-dimensional settings like image-based environments, where performance significantly degrades.

Larger k improves performance by allowing richer feature use, but reduces interpretability, as decisions depend on more intricate input combinations.

However, SPLIT-PO's structural regularization and gating often produce compact trees, even with expressive controllers. This pruning keeps the number of decision points low, preserving interpretability despite increased feature complexity.

Thus, SPLIT-PO can balance interpretability and performance by adjusting both feature-level (via k) and structural complexity, yielding concise, effective, and auditable policies.

5.2 Parameter Count

We analyze the number of learned parameters in SPLIT-PO compared to ICCT and neural network baselines.

Comparison to ICCT. Despite incorporating dynamic structure and gating, SPLIT-PO shares the same asymptotic complexity as ICCT*: both scale as $\mathcal{O}(2^d(m + o \cdot k))$, where d is tree depth, m input dimensionality, o output dimensionality, and k features per leaf. While SPLIT-PO introduces a small constant overhead per node, its ability to prune leads to smaller trees and fewer total parameters in practice—while maintaining equal or better performance.

Comparison to Neural Networks. Neural network baselines require dramatically more parameters. A standard MLP with two hidden layers of 256 units scales roughly as $\mathcal{O}(256 \cdot (m + o + 256))$, often exceeding 60,000 parameters. In contrast, SPLIT-PO models typically use under 300 parameters—achieving near-parity in reward (e.g., 285 vs. 295 on Lunar Lander) with 100–1000× fewer parameters and far greater interpretability.

Full parameter formulas are included in Appendix B.

5.3 Tree Size Tradeoff

Another important dimension of interpretability is the size of the decision tree itself, which directly impacts how easy the model is to inspect and understand. Our results show that SPLIT-PO consistently produces smaller trees than ICCT*, often with fewer than half the number of leaf nodes, while achieving similar or even better performance.

This efficiency is largely due to SPLIT-PO's ability to prune branches that contribute little to overall policy performance. Through its gating mechanism and regularization, the model learns to disable parts of the tree that do not improve decision quality, resulting in compact, task-specific structures. In contrast, ICCT* uses a fixed tree structure, which may include unnecessary branches that add complexity without providing additional value.

Moreover, SPLIT-PO's structural sparsity appears to support better generalization. By avoiding overgrowth and focusing on essential decision paths, the model is less prone to overfitting compared to rigid baselines. This enables SPLIT-PO to learn more generalizable policies that maintain performance across a wider range of states, while still preserving interpretability through a minimal number of high-impact decision rules.

5.4 Sample Efficiency

SPLIT-PO requires more environment interactions than neural network (NN) baselines to reach comparable performance,

reflecting a trade-off between interpretability and sample efficiency.

For instance, in Inverted Pendulum, the NN converges around episode 350, while SPLIT-PO reaches similar scores by episode 500. In Lunar Lander, the NN converges near sample 750; SPLIT-PO takes over 1000.

This gap stems from SPLIT-PO's inductive bias toward sparsity and its need to gradually construct tree structure and feature selection. Unlike high-capacity NNs, which learn rapidly early on, SPLIT-PO begins with a more constrained hypothesis space.

Improving sample efficiency is a natural target for future work for example through imitation learning, prioritized replay, or curriculum learning.

5.5 Limitations

Given the 10-week timeframe, development efforts focused on method design, leaving limited time for thorough model training and hyperparameter tuning. With many tunable components, learning rates, gating thresholds, feature selection temperatures, and regularization weights, more systematic optimization and ablation studies could have improved performance and deepened insights.

Learning stabilization techniques were also underexplored. Training was occasionally unstable, especially in complex environments or with larger trees. Greater focus on stabilization strategies (e.g., adaptive learning rates, gradient clipping, entropy regularization) and more advanced exploration methods might have improved reliability and early learning efficiency.

Execution issues with the original ICCT code [7] required implementation workarounds. Although the modified code is mathematically equivalent, empirical differences due to optimizer choices or initialization could affect performance, making direct comparisons less conclusive.

Lastly, SPLIT-PO struggles with high-dimensional inputs like images, where both sample efficiency and model capacity are critical. While interpretability remains a strength, this limits applicability in sensory-rich domains.

6 Responsible Research

6.1 Interpretability $\neq$ Correctness or Safety

A central aim of this work is to improve interpretability in reinforcement learning by introducing policy representations based on decision trees, which use sequential boolean decision making. However, interpretability alone does not guarantee that a policy is safe, correct, or ethically sound.

While tree-based policies are easier for humans to inspect and understand, they can still encode flawed, biased, or unsafe decision making. This is especially true if they are trained on suboptimal data or under weak reward signals. Interpretability should be seen as a tool for post hoc validation and auditing, not as a substitute for verification or formal safety guarantees.

This distinction is particularly important in high-stakes domains such as robotics and healthcare, where interpretability is only meaningful if the resulting decisions are also safe and effective. Future work could explore integrating formal verification techniques with interpretable models like SPLIT-PO to bridge the gap between transparency and trustworthiness. Given their significantly lower complexity compared to standard neural networks, our trained models may also offer increased efficiency in formal verification.

6.2 Research Integrity

Efforts were made throughout the project to ensure fairness, transparency and responsible reporting of results. All baseline models (ICCT, DDT, and MLP) were re implemented or adapted within the same actor-critic framework to ensure a fair experimental comparison.

Results were not cherry-picked; instead, performance metrics reflect averaged outcomes across multiple runs with different seeds. Benchmarks were selected from standard, publicly available environments in OpenAI Gym, and no task-specific tuning was performed beyond what was necessary for convergence.

The limitations of the proposed methods are discussed in Section 5 including the challenges of a 10 week project, hyper-parameter tuning, learning stabilization, executing original ICCT code and scaling the method to high-dimensional environments such as pixel-based observation spaces. These limitations are openly acknowledged to avoid overstating the generality or robustness of the approach.

6.3 Reproducibility

To support reproducibility, all experiments were carried out with publicly available libraries such as Py-Torch [3] and OpenAI gym [4]. The codebase used to implement SPLIT-PO, including training scripts, model architecture, and evaluation routines, will be made publicly available in a GitHub [5] repository with this thesis.

To account for stochasticity in training, each model was evaluated over 50 random seeds and average performance was reported. Hyperparameters, for both the actor-critic learning components and the tree, based structures—were kept consistent across baselines. A representative subset of these is documented in Appendix A to facilitate reproducibility. For completeness, all hyperparameters used in our experiments are available in the training scripts included in our public repository.

7 Conclusions and Future Work

In this work, we introduced SPLIT-PO: a Sparse Piecewise-Linear Interpretable Tree Policy Optimization framework for reinforcement learning. Our method combines the advantages of differentiable decision trees with dynamic structure learning and sparse linear controllers. Through a combination of soft gating and feature selection mechanisms, SPLIT-PO is able to adaptively prune unnecessary parts of the tree, resulting in sparse and interpretable policies without sacrificing performance.

Our experiments on standard continuous control benchmarks demonstrate that SPLIT-PO can match or closely

[3] https://pytorch.org

[4] https://gymnasium.farama.org

[5] https://github.com/erni12345/SPLIT-PO

approximate the performance of neural network baselines, while producing significantly smaller and more interpretable trees than fixed-structure models like ICCT*. Additionally, we showed that SPLIT-PO supports a flexible interpretability-performance trade-off, and achieves strong generalization with fewer decision nodes.

From a theoretical standpoint, we proved that SPLIT-PO is a universal function approximator, affirming that its expressive capacity is comparable to that of neural networks, even within its sparse and structured representation.

Future Work

Despite its strengths, SPLIT-PO opens several avenues for future research:

- **Sample efficiency:** SPLIT-PO needs many interactions to learn. Methods like imitation learning, prioritized replay, or curriculum learning could help.
- **High-dimensional inputs:** Applying SPLIT-PO to raw images is hard. Future work might use feature extractors or dimensionality reduction.
- **Discrete/hybrid actions:** Extending SPLIT-PO to discrete or mixed action spaces would increase its versatility.
- **Formal verification:** SPLIT-PO's compact, interpretable structure makes it a good fit for verification in safety-critical settings.
- **Non-linear controllers:** Exploring non-linear controllers with sparse features may improve the performance/interpretability tradeoff.
- **Constant-leaf trees:** Using trees with constant outputs (no selected features) could simplify interpretation and serve as a baseline for linear controller impact.

Overall, SPLIT-PO demonstrates that interpretable models in reinforcement learning can be both expressive and competitive, providing a strong foundation for transparent and reliable decision-making in real-world systems.

References

[1] Osbert Bastani, Yani Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems (NeurIPS)*. Curran Associates, Inc., 2018.

[2] Hendrik Blockeel, Lander Devos, Benoît Frénay, Georges Nanfack, and Sebastijan Nijssen. Decision trees: From efficient prediction to responsible ai. *Frontiers in Artificial Intelligence*, 6, 2023.

[3] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[4] Ujjwal Das Gupta, Erik Talvitie, and Michael Bowling. Policy tree: Adaptive representation for policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor–critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[6] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015.

[7] Rishi Paleja, Abhinav Goyal, Peter Stone, and Alexander Khazatsky. Interpretable reinforcement learning for robotics and continuous control. *arXiv preprint arXiv:2311.10041*, 2023.

[8] Ariel M. Roth, Nate Topin, Pooyan Jamshidi, and Manuela Veloso. Conservative q-improvement: Reinforcement learning for an interpretable decision-tree policy, 2019.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[10] 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):745–751, 2002.

[11] Alexandre Silva, Matthew Gombolay, Thomas Killian, Ivan Jimenez, and Seung-Hyeok Son. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1855–1865. PMLR, 2020.

[12] Alberto Suárez and James F. Lutsko. Globally optimal fuzzy decision trees for classification and regression. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(12):1297–1311, 1999.

[13] Daan Vos and Sicco Verwer. Optimizing interpretable decision tree policies for reinforcement learning. *arXiv preprint arXiv:2408.11632*, 2024.

A Hyper Parameters for training

Table 3: Hyperparameters Used for Different Experiment Configurations

Category	**Hyperparameter**	**SPLIT-PO, LunarLander, 2 features**
Optimization	Actor learning rate (`actor_lr`)	0.0001
	Critic learning rate (`critic_lr`)	0.001
	Batch size (`batch_size`)	128
	Replay buffer size (`buffer_size`)	100000
	Soft update coefficient (τ)	0.005
3	Regularization (λ)	1×10^{-6}
Environment	Environment name	LunarLanderContinuous-v3
	Max steps per episode	1000
	Number of training episodes	1200
	Number of evaluation episodes	15
Exploration	Exploration steps	10000
	Random steps	1000
	Initial noise scale	0.3
	Min noise scale	0.05
	Noise decay	0.99
Learning Dynamics	Discount factor (γ)	0.99
	Evaluation frequency	20
	Print frequency	10
Model Structure	Number of features	2
	Tree depth	4
	Initial temperature (`alpha_init`)	1.0
	Temperature	0.1
	Threshold for splitting	0.7

Table 4: Hyperparameters Used for Different Experiment Configurations

Category	**Hyperparameter**	**SPLIT-PO, InvertedPendulum, 4 features**
Optimization	Actor learning rate (`actor_lr`)	0.001
	Critic learning rate (`critic_lr`)	0.0003
	Batch size (`batch_size`)	256
	Replay buffer size (`buffer_size`)	100000
	Soft update coefficient (τ)	0.005
	Regularization (λ)	1×10^{-5}
Environment	Environment name	InvertedPendulum-v5
	Max steps per episode	400
	Number of training episodes	800
	Number of evaluation episodes	5
Exploration	Exploration steps	5000
	Random steps	1000
	Initial noise scale	0.3
	Min noise scale	0.05
	Noise decay	0.99
Learning Dynamics	Discount factor (γ)	0.99
	Evaluation frequency	10
	Print frequency	10
Model Structure	Number of features	4
	Tree depth	2
	Initial temperature (`alpha_init`)	0.2
	Temperature	0.1
	Threshold for splitting	0.5

Table 5: Hyperparameters Used for Different Experiment Configurations

Category	**Hyperparameter**	**SPLIT-PO, InvertedPendulum, 1 feature**
Optimization	Actor learning rate (`actor_lr`)	0.001
	Critic learning rate (`critic_lr`)	0.0003
	Batch size (`batch_size`)	256
	Replay buffer size (`buffer_size`)	100000
	Soft update coefficient (τ)	0.005
	Regularization (λ)	0.001
Environment	Environment name	InvertedPendulum-v5
	Max steps per episode	1000
	Number of training episodes	800
	Number of evaluation episodes	5
Exploration	Exploration steps	5000
	Random steps	1000
	Initial noise scale	0.3
	Min noise scale	0.05
	Noise decay	0.99
Learning Dynamics	Discount factor (γ)	0.99
	Evaluation frequency	10
	Print frequency	10
Model Structure	Number of features	1
	Tree depth	3
	Initial temperature (`alpha_init`)	0.2
	Temperature	0.1
	Threshold for splitting	0.5

Table 6: Hyperparameters Used for Different Experiment Configurations

Category	Hyperparameter	SPLIT-PO, InvertedPendulum, 3 features
Optimization	Actor learning rate (`actor_lr`)	0.001
	Critic learning rate (`critic_lr`)	0.0003
	Batch size (`batch_size`)	256
	Replay buffer size (`buffer_size`)	100000
	Soft update coefficient (τ)	0.005
	Regularization (λ)	0.01
Environment	Environment name	InvertedPendulum-v5
	Max steps per episode	1000
	Number of training episodes	800
	Number of evaluation episodes	5
Exploration	Exploration steps	5000
	Random steps	1000
	Initial noise scale	0.3
	Min noise scale	0.05
	Noise decay	0.99
Learning Dynamics	Discount factor (γ)	0.99
	Evaluation frequency	10
	Print frequency	10
Model Structure	Number of features	3
	Tree depth	3
	Initial temperature (`alpha_init`)	0.2
	Temperature	0.1
	Threshold for splitting	0.5

B Parameter Count Formulas

We include the full expressions used to compute the number of learned parameters for SPLIT-PO, ICCT*, and the neural network (NN) baseline.

SPLIT-PO. For a binary tree of depth d, input dimension m, output dimension o, and k features per leaf controller, the total number of parameters in SPLIT-PO is:

$$P_{\text{SPLIT}}(d) = (2^d - 1)(m + 3) + 2^d \cdot o(k + 1)$$

Each internal node has:

- A feature selection vector of size m
- One threshold scalar
- One sharpness parameter
- One gating parameter

Each leaf node has a sparse linear controller with $k \cdot o$ weights and an o-dimensional bias.

ICCT*. ICCT* omits the gating parameter, yielding:

$$P_{\text{ICCT}}(d) = (2^d - 1)(m + 2) + 2^d \cdot o(k + 1)$$

Neural Network Baseline. For an MLP with two hidden layers of 256 units and input/output dimensions m and o, the total parameter count is approximately:

$$P_{\text{NN}} \approx 256 \cdot (m + 256) + 256 \cdot (256 + o)$$

Bias terms are omitted for brevity but add $\approx 512 + o$ additional parameters.
These formulas provide a basis for comparing model compactness and scaling behavior across architectures.

C Neural Network Architecture

Our neural network baseline is a fully connected policy network with two hidden layers of 256 ReLU units each. It maps states to the parameters of a Gaussian distribution over actions. Given a state input of dimension n, the network is structured as follows:

- Fully connected layer: $n \to 256$ (with bias)
- Fully connected layer: $256 \to 256$ (with bias)
- Output heads:
  - Mean head: $256 \to m$
  - Log standard deviation head: $256 \to m$

Actions are sampled using the reparameterization trick and passed through a $\tanh$ squashing function, then scaled to match the action bounds of the environment. The log-probability is adjusted to account for the change of variables introduced by the $\tanh$ and rescaling steps.

This architecture follows common practice in continuous control, such as Soft Actor-Critic (SAC), and is used as a high-performing black-box baseline.

D Examples of learned trees

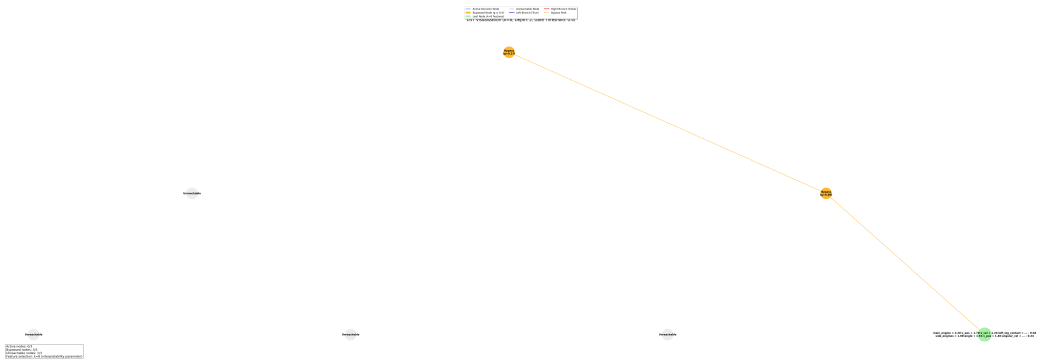

Figure 4: Resulting tree form training with number of features = k and depth = 2 for Lunar Lander

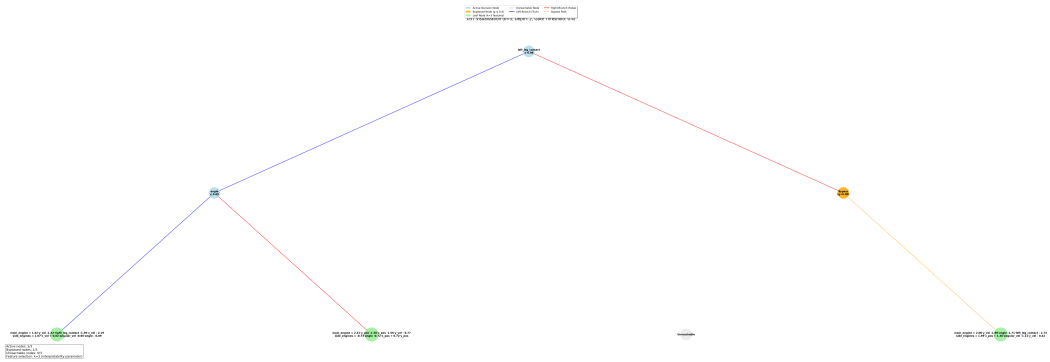

Figure 5: Resulting tree form training with number of features = 3 and depth = 2 for Lunar Lander

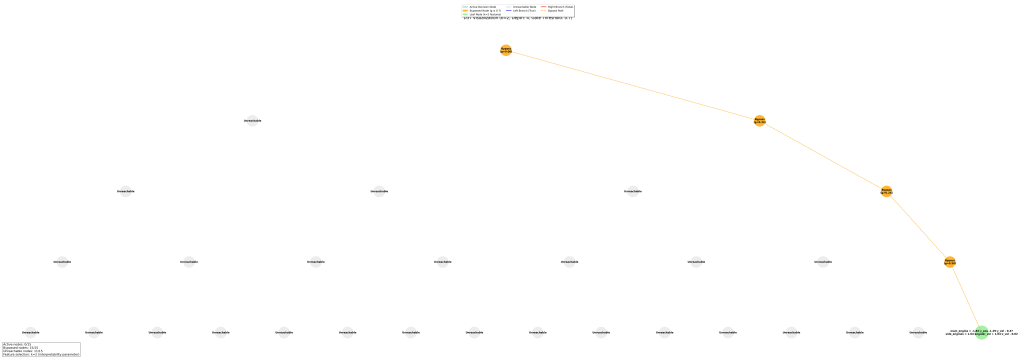

Figure 6: Resulting tree form training with number of features = 2 and depth = 4 for Lunar Lander