Interpretable and Editable Programmatic Tree Policies for Reinforcement Learning

Hector Kohler, Quentin Delfosse, Riad Akrour, Kristian Kersting, Philippe Preux

Interpretable and Editable Programmatic Tree Policies for Reinforcement Learning

Hector Kohler[1*] **Quentin Delfosse**[2,3*] **Riad Akrour**[1]
Kristian Kersting[2,4] **Philippe Preux**[1]
[1] Université de Lille, Inria [2]TU Darmstadt [3] ATHENE [4] Hessian AI, DFKI
{hector.kohler,riad.akrour,philippe.preux}@inria.fr
{quentin.delfosse,kersting}@cs.tu-darmstadt.de

Abstract

Deep reinforcement learning agents are prone to goal misalignments. The black-box nature of their policies hinders the detection and correction of such misalignments, and the trust necessary for real-world deployment. So far, solutions learning interpretable policies are inefficient or require many human priors. We propose INTERPRETER, a fast distillation method producing INTerpretable Editable tRee Programs for ReinforcEmenT lEaRning. We empirically demonstrate that INTERPRETER compact tree programs match oracles across a diverse set of sequential decision tasks and evaluate the impact of our design choices on interpretability and performances. We show that our policies can be interpreted and edited to correct misalignments on Atari games and to explain real farming strategies.

1 Introduction

Why interpretability? The last decade has seen a surge in the performance of machine learning models, in supervised learning [Krizhevsky et al., 2012, Vaswani et al., 2017] and in reinforcement learning (RL) [Mnih et al., 2015, Schulman et al., 2017, Bhatt et al., 2024]. These achievements rely on deep neural networks that are often described as black-box models [Murdoch et al., 2019, Guidotti et al., 2018, Arr, 2020], trading interpretability for performance. In many real world tasks, predictive models can hide undesirable biases, such as the ones listed by Guidotti et al. [2018], hindering trustworthiness towards AI agents. Gaining trust is one of the main goals of interpretability [Arr, 2020], along with informativeness requests, *i.e.* the ability for a model to provide information on why and how decisions are taken. The computational complexity of such informativeness requests can be measured objectively, and Barceló et al. [2020] showed that multi-layer neural networks cannot answer these requests, at least not in polynomial time, whereas explicit structures, *e.g.*, decision trees can.

Interpretable RL using transparent models as policies. In addition to trust and informativeness, the importance of interpretability is further highlighted in RL for addressing shortcut learning, where agents learn to exploit spurious correlations instead of mastering the intended tasks. This leads to poor generalization, often observed as goal misgeneralization in deep RL [di Langosco et al., 2022]. Explainable methods, *e.g.* importance maps, have been used to pinpoint such flawed strategies [Schramowski et al., 2020, Ras et al., 2022, Roy et al., 2022, Saeed and Omlin, 2023]. However, while these methods reveal which inputs affect decisions, they do not clarify *how* they are used within the decision-making process, which is necessary for detecting and correcting misalignments. Delfosse et al. [2024] show that using interpretable concepts as policy states (*e.g.* extracted objects instead of pixels), and transparent policies, (*i.e.* for which the input transformation can

[*]Equal contributions. Code available: https://github.com/KohlerHECTOR/interpreter-py

17th European Workshop on Reinforcement Learning (EWRL 2024).

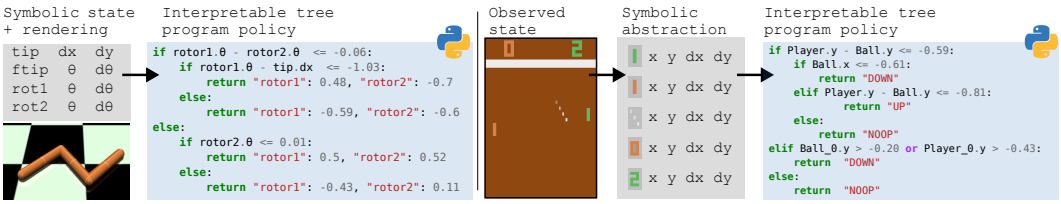


Figure 1: **INTERPRETER provides editable interpretable policy**, as a Python tree programs, illustrated on the Swimmer (left) and Pong (right) environments.

be followed, [Milani et al., 2022, Glanois et al., 2021]) ease the detection and correction of such misalignment, as transparent policies allow experts interventions.

Transparency is not enough. Having an algorithm learning a transparent policy is not enough to achieve interpretability. Imitation-based solutions like [Verma et al., 2018, Bastani et al., 2018, Landajuela et al., 2021] might only require an oracle but return either transparent policies with too many decision rules to be considered interpretable [Bastani et al., 2018] or are only tested on small toy problems [Verma et al., 2018, Landajuela et al., 2021, Topin et al., 2021]. On the other hand, Delfosse et al. [2023b, 2024] outputs transparent policies for various tasks but do require carefully designed human primitives. Another approach is to search for a transparent policy among human-designed polynomial equations with deep RL, and use a large language model (LLM) a posterirori to explain the equations [Luo et al., 2024], but such explanations are more likely to emerge from the LLM's acquired knowledge of the RL tasks rather than its ability to understand polynomials.

Contributions. In this work, we introduce INTERPRETER, a policy distillation method that extracts compact editable tree programs (*cf.* Figure 1). Our algorithm fits regularized oblique trees [Murthy et al., 1994] to neural oracle such as DQN and PPO agents [Mnih et al., 2015, Schulman et al., 2017] and convert them to editable Python programs. Specifically, our contributions are:

1. We introduce INTERPRETER, that extracts, without human priors, compact editable tree programs matching oracles for various RL tasks in few minutes.
2. We conduct ablation studies to identify the responsibility of INTERPRETER's components on the extracted policies' performances.
3. We show that INTERPRETER tree programs can be interpreted and edited by human experts.

We now introduce the background on interpretable reinforcement learning.

2 Background and notations

A Markov decision process (MDP) $\mathcal{M}$ is a tuple $\langle S, A, R, P, \gamma \rangle$ [Puterman, 2014]. We consider continuous states $S \subsetneq \mathbf{R}^p$ and discrete or continuous actions ($\dim(A) \geq 1$). $R : S \times A \to \mathbb{R}$ is the scalar reward function; and $P : S \times A \to \Delta S$ are the transition probabilities ($p(s_{t+1} = s | a_t = a) \sim P$). A discrete (resp. stochastic) policy is a mapping $\pi : S \to A$ (resp. $\pi : S \to \Delta A$). A policy takes actions in an MDP through time and receives rewards $R(s_t, \pi(s_t))$. Given a policy π, the value of π in the state s after taking action a is the expected discounted cumulative reward: $Q^\pi(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim P}[Q^\pi(s', \pi(s')]$, with $0 < \gamma < 1$.

Reinforcement learning algorithms [Sutton and Barto, 2018] look either for the optimal Q-function Q^* ($Q^*(s, a) \geq Q(s, a)$ for any Q, s, a) [Mnih et al., 2013, 2015, van Hasselt et al., 2016]; or for the optimal policy $\pi^* = \text{argmax}_\pi \, \mathbb{E}[\sum_t \gamma^t R(s_t, \pi(s_t))]$ [Schulman et al., 2017, Haarnoja et al., 2018]. Often we have $Q^* = Q^{\pi^*}$ and $\pi^*(s) = \text{argmax}_a \, Q^*(s, a)$. Our goal is to find an interpretable policy T^* whose performances are close to or equal to π^*. We first use reinforcement learning to get π^* and/or Q^{π^*} and then consider two imitation learning methods to find T^*.

Imitation learning transforms a reinforcement learning task into a sequence of supervised learning ones. At each iteration i of Dagger [Ross et al., 2010], T_i is fitted to a dataset of states collected with T_{i-1} and actions given by π^* on those states [Ross et al., 2010]. Q-Dagger [Bastani et al., 2018] further re-weights state-action samples proportional to $\mathbb{E}_{a \in A} Q^{\pi^*}(s, a) - \min_{a \in A} Q^{\pi^*}(s, a)$.

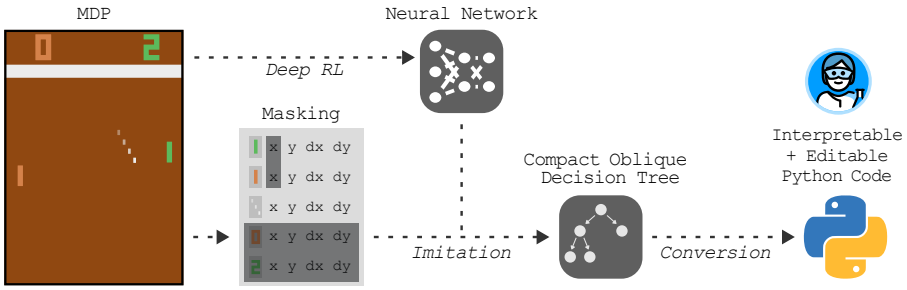


Figure 2: **INTERPRETER's Distillation process.** The MDP state-action space is simplified (idle features and equivalent actions are masked), then an oblique tree policy imitates the oracle. Finally, the policy is then translated to readable and executable code: experts can verify and edit.

3 Method

Imitation Learning routine. To find an interpretable tree program policy T^*, INTERPRETER uses two different imitation subroutines, depending on the nature of the oracle π^*. For MDPs with discrete actions, if both the oracle policy π^* and the associated state-action value function Q^{π^*} are accessible, the oracle data are collected with the Q-Dagger subroutine from [Bastani et al., 2018], described in section 2) and corresponding to line 2 of algorithm 1. On the other hand, if the action space is continuous or only the oracle's policy π^* is accessible, the oracle data are collected with the Dagger subroutine (algorithm 1 of [Ross et al., 2010]). In later case, Bastani et al. [2018] still recommend using the Q-Dagger routine (over Dagger) with $\log(\pi^*(s,a))$ to reconstruct Q^{π^*}. In practice, INTERPRETER performances do not depend on the imitation subroutine (*cf.* experiment 5). Despite the associated runtime and memory costs for storing neural state-action value functions and performing the forward passes (*cf.* line 15 of algorithm 1), we still use Q-Dagger in the case above, as it has better guarantee over Dagger when the $Q^{\pi^*}(s,a)$ are well-estimated [Bastani et al., 2018].

Oblique decision trees. One can imitate oracles with programs that make tests of linear combinations of features. Many oracles learn oblique or more complex decision rules over an MDP state space. This is illustrated in Figure 3 where a PPO neural oracle creates oblique partitions of the state-space for the Pong environments. Programs that test only individual features would fail to fit this partition (*cf.* Figure 3). We thus modify CART [Breiman et al., 1984], an algorithm returning an axes-parallel trees for regression and supervised classification problems, for it to return oblique decision trees. In addition to single feature tests, our oblique trees consider linear combinations of two features with weights 1 and -1, *e.g.* for MDP states $s_i \in \mathbb{R}^p$, the oblique features values are $s_i^{oblique} = \{s_{i1} - s_{i0}, s_{i2} - s_{i0}, ..., s_{ip} - s_{i0}, ..., s_{ip-1} - s_{ip}\} \in \mathbb{R}^{p^2}$. For example, using an oracle dataset with n state-actions pairs: $(\bar{S}, \bar{A} = \pi^*(\bar{S})) \subsetneq \mathbb{R}^{n\cdot(p+\dim(A))}$, we obtain oblique decision trees by fitting $(\bar{S}, \bar{S}^{oblique}, \bar{A} = \pi^*(\bar{S})) \subsetneq \mathbb{R}^{n\cdot(p(p+1)+\dim(A))}$. Given $\bar{S}$, computing $\bar{S}^{oblique}$ can be done efficiently by computing the values of the lower (or upper) triangles in the $\bar{S}\otimes\bar{S} - (\bar{S}\otimes\bar{S})^T$ tensor (excluding the diagonals) (*cf.* line 14 of Algorithm 1). We further demonstrate the superiority of oblique trees in our experimental evaluation on a diverse set of RL tasks.

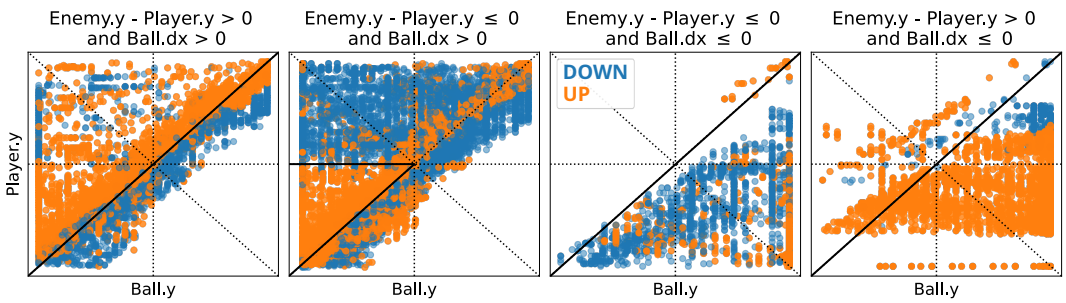


Figure 3: **Oracle decision rules are oblique** illustrated on PPO for different state space partitions of the Pong environment. Decisions boundaries are both oblique and parallel.

The complexity of building the tree (line 17 of algorithm 1) is $O(pn\log_2(n))$ when no maximum tree depth is given, and $O(pnD)$ with a maximum tree depth of D [Sani et al., 2018]. In particular, at iteration i of INTERPRETER the complexity of building the tree is $O(p(p+1)itD)$, as rollouts of t MDP transitions are aggregated (line 16) and oblique features are added to states (line 14). This means that at each iteration i, the cost of computing an oblique tree is $p+1$ times the cost of computing an axes-parallel tree. In INTERPRETER we pass K the maximum number of leaf nodes as an argument. A tree with K leaf nodes has $2K-1$ total nodes and a depth of at most $D=K-1$.

Table 1: **Automated masking reduces the number of features in MDP**, illustrated on 8 Atari environments.

MDP	Ast.	Box.	Free.	Kang.	Pong	Sea.	SpaceI.	Ten.
Full	100	8	48	196	12	172	176	16
Simplified	90	8	22	28	8	54	164	16

Conversion into ready-to-use programs. After the imitation learning subroutine, INTERPRETER converts the best evaluated oblique tree in line 18 into a Python program (*cf.* line 19), such as the ones depicted in Figure 1. To do so, we turn the internal nodes of the tree into if-else statements. This algorithmic step does not involve any randomness nor learning. This Python based representations of our interpretable policies, unlike a tree plot, INTERPRETER can easily be edited, as shown in section 4. Finally, we perform pruning to further simplify our programs by merging redundant logical branches. We provide details and examples in Appendix F.

Algorithm 1: INTERPRETER

1 **function** INTERPRETER (π^*, $\mathcal{M}$, K, N, t, $Q^{\pi^*}=None$)
2 **if** IsNotNone(Q^{π^*}) & IsDiscrete(A) **then**
3 SampleWeighting() $\leftarrow \mathbb{E}_{a\in A}Q^{\pi^*}(s,a) - \min_{a\in A} Q^{\pi^*}(s,a)$
4 **else**
5 SampleWeghting() $\leftarrow 1$
6 $\mathcal{M} \leftarrow$ MaskIdleFeatures($\mathcal{M}$)
7 $(\bar{S}, \bar{S}^{oblique}, \bar{A}, \bar{W}) \leftarrow \emptyset$
8 **for** $i = 1, 2, ..., N$ **do**
9 **if** $i = 1$ **then**
10 $\bar{S}_i \leftarrow$ rollouts(π^*, $\mathcal{M}$, t)
11 **else**
12 $\bar{S}_i \leftarrow$ rollouts(T_{i-1}, $\mathcal{M}$, t)
13 $\bar{A}_i \leftarrow \pi^*(\bar{S}_i)$
14 $\bar{S}_i^{oblique} \leftarrow$ Triangles($\bar{S}_i \otimes \bar{S}_i - (\bar{S}_i \otimes \bar{S}_i)^T$)
15 $\bar{W}_i \leftarrow$ SampleWeighting($\bar{S}_i, Q^{\pi^*}$)
16 $(\bar{S}, \bar{S}^{oblique}, \bar{A}, \bar{W}) \leftarrow (\bar{S}, \bar{S}^{oblique}, \bar{A}, \bar{W}) \cup (\bar{S}_i, \bar{S}_i^{oblique}, \bar{A}_i, \bar{W}_i)$
17 $\bar{T}_i \leftarrow$ CART($(\bar{S}, \bar{S}^{oblique}, \bar{A}, \bar{W}), K$)
18 $T^* \leftarrow$ BestTreeEvaluation($\{\bar{T}_1, ..., \bar{T}_N\}$, $\mathcal{M}$)
19 Prog $\leftarrow$ TreeToProgram(T^*)
20 **return** T^*, Prog

Let us now evaluate INTERPRETER's performances, interpretability, and correction possibilities.

4 Experimental Evaluation

In this section, we evaluate the compact tree programs provided by INTERPRETER. Specifically, our experimental evaluation aims at answering the following research questions: **(Q1)** Can compact INTERPRETER tree programs match the oracles performances? **(Q2)** What are the key design choices of INTERPRETER? **(Q3)** Can INTERPRETER's programs be interpreted and modified by experts?

Metrics. For INTERPRETER, we denote the maximum number of leaf nodes, K, the number of different fitted trees N, and the number of transitions, t. Given an MDP $\mathcal{M}$, the oracle policy π^*, and potentially its Q-value function, Q^{π^*}, from $\mathcal{M}$ to fit each tree, we report in the value of the cumulative reward of the INTERPRETER tree programs, normalized by the oracle, and often average it over multiple MDPs that have similar properties, such as M MuJoCo robots or M Atari games.

Benchmarks. We tested INTERPRETER on a set of common RL benchmarks: classic control, Atari games, MuJoCo robot simulations [Todorov et al., 2012, Bellemare et al., 2012, Schulman et al., 2015, van Hasselt et al., 2016, Schulman et al., 2017, Haarnoja et al., 2018]. In particular, for Atari games, we use the stochastic version where actions have a non-zero probability to be repeated as per the recommendations of [Machado et al., 2018] for best practices of RL training. We use `gymnasium` [Towers et al., 2023] implementations of those benchmarks: we use `-v4` version of MuJoCo environments, `-v5` version of Atari games, and the latest versions of classic control.

Object-centric Atari. To reduce the computational burden, and as object detection is not the core focus of this work, we use the neurosymbolic states efficiently extracted by OCAtari [Delfosse et al., 2023a] for INTERPRETER trees to map neurosymbolic states to actions. OCAtari extracts these states from the RAM for each Atari environment, with similar accuracies as other extraction methods Lin et al. [2020], Zhao et al. [2023]. These states list the depicted game objects x, y-coordinates rather than pixels. For further details, we refer to the authors' original publication.

Neural oracles. Most interpretable RL algorithms extract transparent policies from black-box oracles, such as deep neural networks [Bastani et al., 2018, Verma et al., 2018, Delfosse et al., 2024]. For MuJoCo and classic control, we re-use oracles from `stable-baselines3` zoo [Raffin, 2020]; the final policies obtained from a SAC agent training [Haarnoja et al., 2018], and from DQN [Mnih et al., 2015] and PPO [Schulman et al., 2017] for classic control. For Atari tasks, we PPOs with `stable-baselines3` [Raffin et al., 2021] and the hyperparameters from [Schulman et al., 2017]. The oracles' training curves are depicted in Figure 8, in the Appendix.

INTERPRETER hyperparameters. Unless stated otherwise, we do between 3 and 5 runs of INTERPRETER in every experiment. Given an oracle policy π^* and optionally its associated Q^{π^*}, each run fits a total of $N = 10$ trees by aggregating $t = 10^4$ transitions at each iteration. We vary the imitation learning subroutines, the fitted tree classes, and the maximum number of nodes allowed in each tree ($2K$ with K the maximum number of leaf nodes passed to INTERPRETER). We use the `scikit-learn` [Pedregosa et al., 2011] implementation of the CART decision tree algorithm [Breiman et al., 1984] with default hyperparameters and K maximum leaf nodes. All experiments are run on a single `Intel Core i7-8665U@1.90GHz`. Our code is given in supplementary material.

4.1 INTERPRETER performances match the oracle performances (Q1)

We test INTERPRETER on the aforementioned benchmarks using algorithm 1. As shown in Figure 4, INTERPRETER tree programs composed of as few as 16 nodes can outperform their neural oracle on classical control tasks and on Asterix, Pong and SpaceInvaders Atari games (*cf.* Appendix 9). In particular, for PPO neural oracles on classic control tasks, INTERPRETER consistently outperform their neural counterparts. In general, INTERPRETER performances increase with the number of nodes. With 64 nodes, INTERPRETER programs consistently match or surpass neural oracles' performances on 6 out of 8 Atari tasks, and obtain comparable scores for the 2 others. For MuJoCo, INTERPRETER match oracles for HalfCheetah and Swimmer, but fail at controlling Walker2d and Hopper. Importantly, INTERPERTER's trees and programs can be extracted within in a few minutes. The greatest runtime bottleneck is MDP rollouts. Finally, when the training of neural RL agents fails (*i.e.* has not converged), *e.g.*, DQN curves on Figure 8, the imitation process is noisy, and the INTERPRETER trees have poor oracle normalized performances, *cf.* DQN error bars in Figure 4.

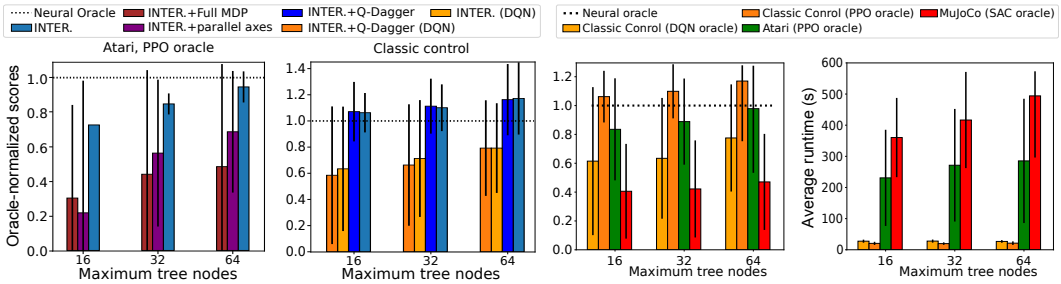


Figure 4: **INTERPRETER matches oracles thanks to its design choices.** From left to right: ablated INTERPRETER, INTERPRETER with different oracles and imitations, performances and runtimes.

4.2 INTERPRETER Ablation (Q2)

Imitation learning subroutines.

We use INTERPRETER to extract decision tree policies for classic control problems, varying the imitation learning subroutine. To do so, we force INTERPRETER use Q-Dagger [Bastani et al., 2018] (*cf.* line 2 of Algorithm 1), even when the neural oracle is a stochastic policy π^* from a PPO agent. In that case, we use $\log \pi^*$ as Q-function. As depicted in Figure 5, for a given oracle and a given maximum number of nodes, Q-Dagger and Dagger minimize the Q-Dagger loss $\mathbb{E}_{a \in A} Q^{\pi^*}(s,a) - \min_{a \in A} Q^{\pi^*}(s,a)$ similarly. Bastani et al. [2018] show that Q-Dagger trees have fewer nodes but are as good as Dagger trees on their own implementation of the Pong environment. Our results on the original Pong Atari Learning Environment contradict this claim, as shown in Figure 9, where for a given number of nodes both Dagger and Q-Dagger trees perform similarly in average. This can be due to the fact that either (i) Dagger trees are shorter than Q-Dagger trees when the oracle is not a well-estimated Q-function and/or (ii) the fitted trees do oblique tests and/or (iii) the nodes regularization is done by bounding the number of leaves rather than the depth.

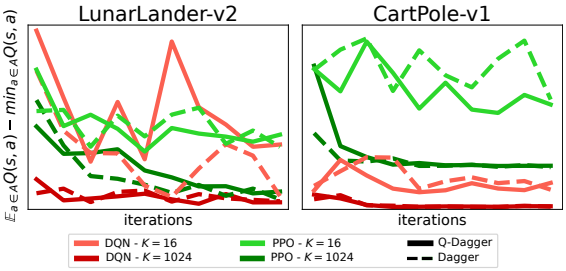


Figure 5: **Q-Dagger does not improve sampling,** shown by the similar loss (to Dagger) during the extraction for different oracles and imitations.

Oblique decision trees. We compare INTERPRETER's tree program performances when fitting classical axes-parallel trees [Breiman et al., 1984] – that have internal nodes such as "is the x-coordinate of the player $\leq v$?" – with fitting oblique trees [Murthy et al., 1994] that have internal nodes like "is the x-coordinate of the player — the previous x-coordinate of the player $\leq v$?". On Atari games (Figure 4), using oblique trees over ones for which decisions are parallel to the axes is critical to match oracle performances. As demonstrated in a per-game ablation (*cf.* Figure 9 in the Appendix), no axes-parallel tree can get close to the oracle performances, even with a high number of internal nodes. This is supported by our early observation (*cf.* Figure 3). Further, the linear combinations of input features used in oblique tree programs do not hinder interpretability, as these features are still very human understandable (representing *e.g.* distance over one axis). However, for some complicated control problems, such as Tennis (Figure 9 in the Appendix) or Walker2d (Figure 8, right), no oblique tree program matches the oracle performances even with 64 nodes.

Removing idle state features. As explained in Section 3, the oblique tree programs' input features is way higher than the one of parallel ones. This number will particularly explode in environments such as Kangaroo, Seaquest, and SpaceInvaders, that have up to 200 total state features (*cf.* table 1). For these environments, the oblique tree extraction gives Out-of-Memory errors (reported as a random scores) when fitting oblique decision trees programs, during the main loop of INTERPRETER (line 17 of algorithm 1). However, many consider features can be constant. For example, the x-coordinates of the player and the enemy in Pong, or the ladders coordinates in Kangaroo. We mask such features, as are incorporated in the decision boundaries of if-else conditions. As shown in Table 1, for Kangaroo and Seaquest, the number of feature can be divided (up to 5 times). As depicted in Figure 4, there is a substantial performance gain obtained by masking idle features for INTERPRETER oblique tree programs' extraction in Atari environments.

4.3 INTERPRETER tree programs interpretability (Q3)

Inference speed. One proxy for the interpretability of machine learning models without relying on human feedback is the inference computational complexity, which can be evaluated with code runtimes [Lipton, 2016, Barceló et al., 2020]. INTERPRETER extracts oblique decision trees and convert them to Python programs. We here compare the average inference speed of different policies during MDP rollouts, *i.e.* how fast the policy outputs actions given states. All the inference speeds are measured without using pre-compiled code, neural network based oracles are instances of `PyTorch` modules [Paszke et al., 2019]. The INTERPRETER trees and programs have 64 oblique test nodes. Results show that Python programs inference takes in average $79\mu s$, compared to $236\mu s$ for `scikit-learn` tree class and $466\mu s$ `PyTorch` networks (*cf.* Table 2 in the appendix for detailed results). The inference time is above all important for real world deployments of algorithms. As explain in the introduction, these deployments also require high interpretability and trust levels.

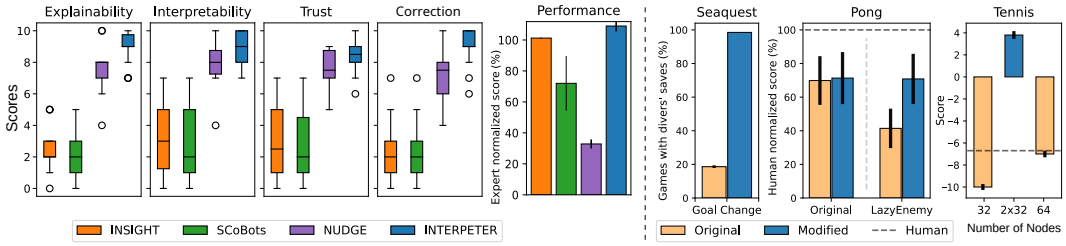


Figure 6: **INTERPRETER is the most interpretable policy form and can easily be modified.** Left: a user study shows that INTERPRETER is more explainable, interpretable, trustworthy and adjustable than other baselines, without sacrificing performances, contrary to the runner-up for these metrics (NUDGE). Right: Its Python policies allow for easy corrections on 3 tasks variations.

User study. We compared the interpretability and performances of INTERPRETER against INSIGHT [Luo et al., 2024], SCoBots [Delfosse et al., 2024] and NUDGE [Delfosse et al., 2023b]. We conducted a user study involving 19 machine learning practitioners. They were asked to evaluate the *explainability* (*i.e.* ability to detect each input feature importance), *interpretability* (*i.e.* how each element is used by the policy to select an action) and *trust* (*i.e.* ability to check if the agent selects the correct action for the correct reasons) levels of policies extracted by each method on Pong (as these policies are accessible for each method). Results are reported in Figure 6, and show that INTERPRETER achieves the highest scores on each measurement. Contrary to its runner-up on these metrics (NUDGE), INTERPRETER does not sacrifice performances for interpretability.

4.4 INTERPRETER tree programs edition (Q3)

We further showcase 3 modification possibilities on policies of INTERPRETER.

Seaquest contains ill-defined reward [Delfosse et al., 2024], as the game goal in the instruction manual is to "retrieve as many treasure-divers as you can"[2]. However, the game does provide any reward to the agent for saving each collected divers, but rather for killing enemies. Thus, both the neural and Python policies do not bring collected divers back to the surface. By simply adding: `if diver_0.x > 0: return "UP"` at the start of the 32 nodes INTERPRETER tree program, we correct this suboptimal behavior. As shown in Figure 6 (Left), the agents are able to save at least one diver in 98.5% of the cases (compared to 18.6% for the original agent). The complete and modified INTERPRETER program is provided in Figure 11 in the Appendix.

Pong. RL policies can learn to achieve a misaligned goal, *i.e.* to rely on the enemy's vertical position for their decision process (instead of the ball's one) [Delfosse et al., 2024], as the two object's vertical positions are 99.9% correlated. Our simple Pong program policy indeed uses `Enemy.y` in 1 out of 6 conditional tests (*cf.* the Pong policy in Figure 12). They introduce a *LazyEnemy* modified version of the environment, where the Enemy remains still after returning the ball, showing that many policies fail to generalize to this environment. We modify our policy by simply replacing `Enemy.y` with `Ball.y`, and test this modified policy on both environments. This leads to equivalent performances on the original training environment and prevents performance drops in the *LazyEnemy* variation (*cf.* Figure 6). Compared to Delfosse et al. [2024], that retrain an oracle, while hiding the enemy, we make one simple modification of the program, and do not need retraining.

Tennis. For this environment, INTERPRETER's tree programs (*i.e.* 16 to 64 nodes) cannot match the oracle's performances. Tennis is a complicated variation of Pong, as it adds the y-coordinates. However, oracle strategies can easily be divided into two sub-strategies, one where the agent is above the net (on the upper part of the screen), and one where it is under (on the lower side). To show the easy curriculum learning possibility offered by INTERPRETER, we extract two 32 nodes-based code, based on a partition of the environment, based on the player's position. We are thus able to extract two policies, correctly performing only on one environment variation each, that we call within a meta policy. This policy calls each expert policy, depending on the evaluation of `player.y - enemy.y > 0`. This meta-policy (2x32) outperforms the 32 and 64 nodes ones (*cf.* Figure 6). We have shown that the INTERPRETER code-based policies allow for easy corrections.

[2] https://atariage.com/instruction_seaquest.

4.5 Real life use case of INTERPRETER tree programs for fertilization of soils (Q3)

In this last experiment, we distill a human expert policy for soil fertilization on the `gym-DSSAT gym` environment Gautron et al. [2023]. Here, an RL agent has to learn to manage a crop, based on an accurate simulated mechanistic model of plant growth. We consider the task that consists in optimizing plant nitrogen absorption while penalizing the application of fertilizer to minimize the economical and the environmental costs. We extract an INTERPRETER's Python program, depicted in Figure 7. This program outputs the exact same actions as the human heuristic given the soil state and obtain the same cumulative reward in average (corresponding to an accuracy of 100%). It also provides an interpretation of the human expert heuristic that delivers a certain amount of fertilizer ($\{27, 35, 54\}$) after $\{39, 45, 80\}$ days after seeding, respectively). The feature importance coincides with agronomic principles and have been validated by an expert from the *Consultative Group on International Agricultural Research*. The nitrogen requirements of corn vary depending on the growth stage. They are important during the vegetative phase (plant growth) and the reproductive phase (from flowering to grain filling). This is why it is essential to consider the number of days after planting and the growth stage of the corn, as nitrogen requirements are highest during grain filling.

```
if nitrogen - days_planting < -17.50:
   if nitrogen - grain_weight < 13.50:
      if nitrogen - days_planting < -39.50:
         if nitrogen - maize_growing < -5.00:
            return "fertilizer_quant": 0.0
         else:

            return "fertilizer_quant": 27.0
      else:
         return "fertilizer_quant": 0.0
   else:
      if nitrogen - biomass < -930.64:
         return "fertilizer_quant": 54.0
      else:
         return "fertilizer_quant": 35.0
```

Figure 7: **INTERPRETER can explain human heuristic policies.** INTERPRETER program with 100% accuracy on human oracle policy.

5 Related work

Explainable policies. Algorithms for policy verification, such as the fast oracle matching VIPER [Bastani et al., 2018], only concentrate on optimizing an axes-parallel tree structure disregarding interpretability by growing many nodes trees. Thus, VIPER trees are explainable in the sense that one can always compute the set of rules verified by an MDP state that lead to an action, but deep trees are not interpretable in the simulatability sense [Lipton, 2016], because humans cannot themselves make the computations to explain actions chosen by the tree policy. Beyond VIPER, work from the neuro-symbolic RL community can learn explainable policies without oracles but require either carefully designed low level policy primitives to facilitate learning [Qiu and Zhu, 2022] or large language models to attempt to explain learned policies [Luo et al., 2024].

Interpretable policies. On the other hand, some algorithms are designed to direcly optimize (with or without oracle knowledge) policies that are intrinsically interpretable. The recent SCoBots [Delfosse et al., 2024] output interpretable policies as sets of rules using ECLAIRE [Zarlenga et al., 2021] to match a PPO oracle. However, SCoBots require LLMs to define and experts to restrict the search space of sets of rules. Furthermore, LLMs might rely on external integrated knowledge about the game, acquired during their training phase, to explain the policy instead of providing accurate descriptions of it. Despite that, SCoBots, to the best of our knowledge, is the first algorithm that can consistently and automatically output interpretable policies for Atari games by using object-centric representations [Bellemare et al., 2012, Delfosse et al., 2023a]. Prior to that, PIRL, Custard and NUDGE [Verma et al., 2018, Topin et al., 2021, Delfosse et al., 2023b] were also able to learn interpretable policies (programs, axes-parallel trees, and first-order logic respectively), but only on toy problems, for which environments were specifically created. Outside of RL, program synthesis has also been explored, *e.g.* on classification tasks [Ellis et al., 2021, Wüst et al., 2024].

In addition to distinguishing existing work by the level of interpretabiliy of their returned policies and by their requirements for human and LLM interventions, we also distinguish the problems they can solve. VIPER, NUDGE, INSIGHT and Custard only work for MDPs with discrete actions, while the other algorithms work with any MDP. The above classification of existing interpretable RL algorithms are summarized in Appendix table 3.

Symbolic states. Recent interpretable RL assume access to a deep learning based object extractor that extract object-centric states from RGB inputs in game domains, such as SPACE [Lin et al., 2020], SPOC [Delfosse et al., 2023c] or a finetuned FastSAM [Luo et al., 2024]. They train a neural network based policy using existing deep RL algorithm from this object-centric states. Then, they distill this policy into either directly interpretable (or transparent) first order logic-based NUDGE agents [Delfosse et al., 2023b] or rule-based SCoBots [Delfosse et al., 2024], or into (not interpretable) polynomial equations, within INSIGHT [Luo et al., 2024].

6 Limitations and future work

Decision tree algorithm. CART [Breiman et al., 1984] is a widely used decision tree learning algorithm, however it chooses splits greedily w.r.t. a train set, which is suboptimal [Murthy and Salzberg, 1995]. There is a whole line of work on decision tree learning algorithm, some specialize in oblique trees [Murthy et al., 1994], others have better generalization performances [Mazumder et al., 2022, Demirovic et al., 2022] and even better interpretability [Kohler et al., 2024]. One direct future direction for INTERPRETER would be to try different decision tree algorithms in algorithm 1.

More expressive and general tree programs. Our INTERPRETER algorithm uses CART with linear combinations of at most two features as input, we could also add linear combinations of more features with coefficients different from 1 or include more complex functions of features such as sinusoidal functions. It should also be possible to add an evolutionary routine [Eiben and Smith, 2015] to include loops in the policy search space. One could for example try to gather rules applied on the same object types, to obtain *e.g.* conditional tests on all the enemies or on every missile in environments like Seaquest or SpaceInvaders.

Complexity and state space. Exploring the solution space of policies defined over symbolic states is necessary for interpretabiliy but comes with limitations. For example, in Seaquest, if the oxygen bar level is not encoded in the symbolic states, the agent cannot learn to make decisions based on the latter. In MsPacman, the walls can be considered part of the background [Lin et al., 2020, Delfosse et al., 2023c], but are necessary to navigate the maze. Generally, identifying what symbolic features are necessary to master each task is a difficult problem [Delfosse et al., 2024]. Furthermore, the complexity of the input space grows with the number of symbols in the state-space. We have proposed to mask idle state features, but there is no guarantee that this will sufficiently reduce the complexity for the decision tree extraction to be done in a limited time. One way to overcome this limitation would be to use *e.g.* a deep learning alternatives that would return the oblique tests to consider for the tree programs given the whole state-action dataset [Kossen et al., 2021], but such deep learning architecture complexity do not scale as much with the number of symbolic features.

Evaluating interpretability. We have here evaluated the interpretability of INTERPRETER programs with an inference speed proxy and with limited user study. We should benchmark the interpretability of our programs with a more diverse pool of users, such as non machine learning practitioners, and could also make use of recently developed LLM code generation capabilities to explain our programs in natural language [Bashir et al., 2024]. We then should evaluate the reliability of using LLMs to accurately explain the policy without relying on accumulated knowledge.

Code. We have only provided the code to reproduce our experiments and a tutorial on a simple demo task. We are building a Python package to use INTERPRETER with `gym` and `PyTorch` oracles.

7 Conclusion

We have introduced INTERPRETER that distills deep RL oracle into interpretable programs to increase alignment and trust in automated sequential decision-making tasks. To do so, INTERPRETER produces tree programs that make oblique tests of meaningful state features. We empirically showed a state-of-the-art interpretability-performance trade-off: our programs match oracles and can be explained and edited by humans. Furthermore, INTERPRETER is a simple algorithm: its components such as decision tree learning, features combinations, and programming language of the extracted programs, can be varied easily. For future work, we believe that benchmarking and safeguarding interpretability and alignment of policies to *e.g.* human values are interesting avenues. We hope our work paves the way for future interpretable RL research.

References

Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 2020.

Pablo Barceló, Mikaël Monet, Jorge Pérez, and Bernardo Subercaseaux. Model interpretability through the lens of computational complexity. In *Neural Information Processing Systems (NeurIPS)*, 2020.

Pablo Barceló, Mikaël Monet, Jorge Pérez, and Bernardo Subercaseaux. Model interpretability through the lens of computational complexity. *Advances in neural information processing systems*, 2020.

Zahra Bashir, Michael Bowling, and Levi H. S. Lelis. Assessing the interpretability of programmatic policies with large language models, 2024.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Neural Information Processing Systems*, 2018.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents (extended abstract). In *International Joint Conference on Artificial Intelligence*, 2012.

Aditya Bhatt, Daniel Palenicek, Boris Belousov, Max Argus, Artemij Amiranashvili, Thomas Brox, and Jan Peters. Crossq: Batch normalization in deep reinforcement learning for greater sample efficiency and simplicity. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=PczQtTsTIX.

Leo Breiman, Jerome Friedman, R.A. Olshen, and Charles J. Stone. *Classification And Regression Trees*. Taylor and Francis, New York, 1984.

Quentin Delfosse, Jannis Blüml, Bjarne Gregori, Sebastian Sztwiertnia, and Kristian Kersting. Ocatari: Object-centric atari 2600 reinforcement learning environments. *ArXiv*, 2023a.

Quentin Delfosse, Hikaru Shindo, Devendra Singh Dhami, and Kristian Kersting. Interpretable and explainable logical policies via neurally guided symbolic abstraction. 2023b.

Quentin Delfosse, Wolfgang Stammer, Thomas Rothenbacher, Dwarak Vittal, and Kristian Kersting. Boosting object representation learning via motion and object continuity. In Danai Koutra, Claudia Plant, Manuel Gomez Rodriguez, Elena Baralis, and Francesco Bonchi, editors, *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML)*, 2023c.

Quentin Delfosse, Sebastian Sztwiertnia, Wolfgang Stammer, Mark Rothermel, and Kristian Kersting. Interpretable concept bottlenecks to align reinforcement learning agents. *arXiv*, 2024.

Emir Demirovic, Anna Lukina, Emmanuel Hebrard, Jeffrey Chan, James Bailey, Christopher Leckie, Kotagiri Ramamohanarao, and Peter J. Stuckey. Murtree: Optimal decision trees via dynamic programming and search. *Journal of Machine Learning Research*, 23(26):1–47, 2022. URL http://jmlr.org/papers/v23/20-520.html.

Lauro Langosco di Langosco, Jack Koch, Lee D. Sharkey, Jacob Pfau, and David Krueger. Goal misgeneralization in deep reinforcement learning. In *International Conference on Machine Learning ICML*, 2022.

Agoston E Eiben and James E Smith. *Introduction to evolutionary computing*. Springer, 2015.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*, 2021.

Romain Gautron, Emilio J Padrón, Philippe Preux, Julien Bigot, Odalric-Ambrym Maillard, Gerrit Hoogenboom, and Julien Teigny. Learning crop management by reinforcement: gym-dssat. In *AIAFS 2023-2nd AAAI Workshop on AI for Agriculture and Food Systems*, 2023.

Claire Glanois, P. Weng, Matthieu Zimmer, Dong Li, Tianpei Yang, Jianye Hao, and Wulong Liu. A survey on interpretable reinforcement learning. *ArXiv*, 2021.

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Hector Kohler, Riad Akrour, and Philippe Preux. Interpretable decision tree search as a markov decision process, 2024.

Jannik Kossen, Neil Band, Clare Lyle, Aidan Gomez, Tom Rainforth, and Yarin Gal. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=wRXzOa2z5T.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, 2012.

Mikel Landajuela, Brenden K Petersen, Sookyung Kim, Claudio P Santiago, Ruben Glatt, Nathan Mundhenk, Jacob F Pettit, and Daniel Faissol. Discovering symbolic policies with deep reinforcement learning. In *International Conference on Machine Learning*, pages 5979–5989. PMLR, 2021.

Zhixuan Lin, Yi-Fu Wu, Skand Vishwanath Peri, Weihao Sun, Gautam Singh, Fei Deng, Jindong Jiang, and Sungjin Ahn. SPACE: unsupervised object-oriented scene representation via spatial attention and decomposition. In *International Conference on Learning Representations*, 2020.

Zachary Chase Lipton. The mythos of model interpretability. *ArXiv*, 2016.

Lirui Luo, Guoxi Zhang, Hongming Xu, Yaodong Yang, Cong Fang, and Qing Li. Insight: End-to-end neuro-symbolic visual reinforcement learning with language explanations. *ArXiv*, 2024.

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents (extended abstract). In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI*, 2018.

Rahul Mazumder, Xiang Meng, and Haoyue Wang. Quant-BnB: A scalable branch-and-bound method for optimal decision trees with continuous features. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 15255–15277. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/mazumder22a.html.

Stephanie Milani, Nicholay Topin, Manuela M. Veloso, and Fei Fang. A survey of explainable reinforcement learning. *ArXiv*, 2022.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015.

W. James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 2019.

Sreerama Murthy and Steven Salzberg. Lookahead and pathology in decision tree induction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'95, page 1025–1031, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1558603638.

Sreerama K Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of artificial intelligence research*, 1994.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Wenjie Qiu and He Zhu. Programmatic reinforcement learning without oracles. In *International Conference on Learning Representations*, 2022.

Antonin Raffin. Rl baselines3 zoo, 2020.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021.

Gabrielle Ras, Ning Xie, Marcel van Gerven, and Derek Doran. Explainable deep learning: A field guide for the uninitiated. *Journal of Artificial Intelligence Research*, 2022.

Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, 2010.

Nicholas A. Roy, Junkyung Kim, and Neil C. Rabinowitz. Explainability via causal self-talk. 2022.

Waddah Saeed and Christian W. Omlin. Explainable AI (XAI): A systematic meta-survey of current challenges and future opportunities. *Knowledge-Based Systems*, 2023.

Habiba Sani, Ci Lei, and Daniel Neagu. *Computational Complexity Analysis of Decision Tree Algorithms: 38th SGAI International Conference on Artificial Intelligence, AI 2018, Cambridge, UK, December 11–13, 2018, Proceedings*, pages 191–197. 11 2018. ISBN 978-3-030-04190-8. doi: 10.1007/978-3-030-04191-5_17.

Patrick Schramowski, Wolfgang Stammer, Stefano Teso, Anna Brugger, Franziska Herbert, Xiaoting Shao, Hans-Georg Luigs, Anne-Katrin Mahlein, and Kristian Kersting. Making deep neural networks right for the right scientific reasons by interacting with their explanations. *Nature Machine Intelligence*, 2020.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

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

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

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

Nicholay Topin, Stephanie Milani, Fei Fang, and Manuela Veloso. Iterative bounding mdps: Learning interpretable policies via non-interpretable methods. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, 2023.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2018.

Antonia Wüst, Wolfgang Stammer, Quentin Delfosse, Devendra Singh Dhami, and Kristian Kersting. Pix2code: Learning to compose neural visual concepts as programs. *ArXiv*, 2024.

Mateo Espinosa Zarlenga, Zohreh Shams, and Mateja Jamnik. Efficient decompositional rule extraction for deep neural networks. In *eXplainable AI approaches for debugging and diagnosis.*, 2021.

Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast segment anything. *arXiv*, 2023.

A Neural oracles

In this section we show the training curves of the neural oracles used in sections 4.1, 4.2. For the the MuJoCo and classic control benchmarks (center and right on Figure 8), the neural oracles policies as well as training data are taken directly from the `stable-baselines3 zoo`, i.e we do not run the training ourselves. For example, all the data for the SAC oracle on Swimmer can be found at this url https://github.com/DLR-RM/rl-trained-agents/tree/ca4371d8eef7c2eece81461f3d138d23743b2296/sac/Swimmer-v3_1. For OC Atari we train the PPO agent from `stable-baselines3` ourselves on a DGX cluster. We use the default [Schulman et al., 2017] hyperparameters on 2e7 timesteps. What we observe is that for most benchmarks the deep reinforcement learning algorithms converged except for the DQN agents on classic control tasks (Figure 8, center). We also show that some SAC oracle are too complex to be matched by oblique tree prorgrams even high a high number of nodes (right of Figure 8).

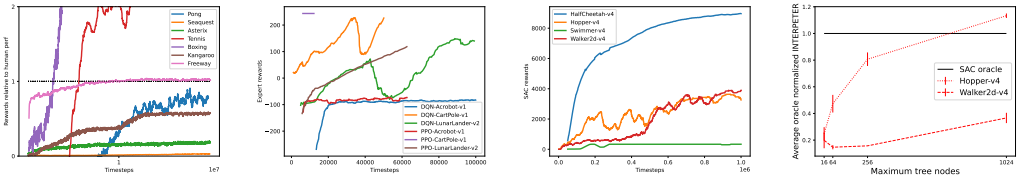


Figure 8: Detailed oracle training curves for Atari, mujoco and classic control environments, as well as the performance evolution of the oracles for different tree sizes for complex control tasks.

B Per game ablation

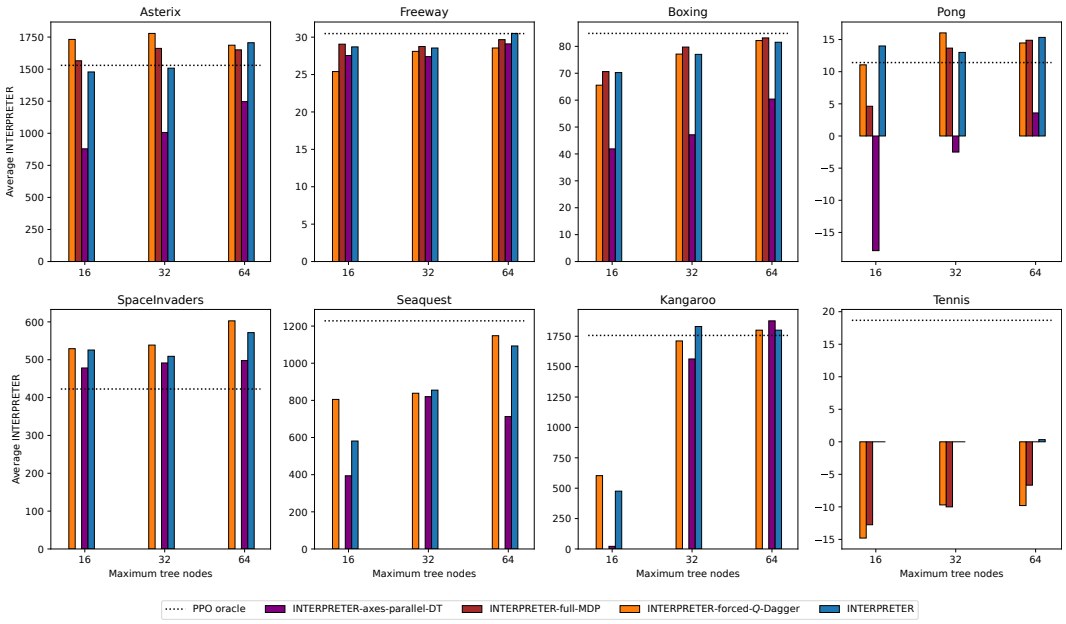


Figure 9: Detail ablation of INTERPRETER.

In this section, we ablate INTERPRETER on OC Atari games. We remove each element of method 3 to get four distinct algorithms. Figure 9 clearly shows that each independent component of INTERPRETER participates in its performances.

INTERPRETER: the default implementation of algorithm 1.

INTERPRETER-axes-parallel-DT: we fit axes-parallel trees to oracles rather than oblique ones.

INTERPRETER-full-MDP: we do not mask idle MDP features during the imitation. In addition the performances we show in 1 the share of idle features for each game.

INTERPRETER-forced-Q-Dagger: even though the given oracle is the stochastic policy π^* returned by a PPO, we also pass $Q^{\pi^*} = \log \pi^*$ to "force" the Q-Dagger weighting during the imitation subroutine.

C Detailed inference time per game

Table 2: **INTERPRETER Programs are more resource efficient.** Inference speed comparisons (in $10^{-5}s$) of programs, `scikit-learn` decision trees and `PyTorch` netowrks on different gym environments. For all environments, our compact programs show faster inference time.

	Asterix	Boxing	Freeway	Kanga.	Pong	Seaquest
Program	1.05$\pm$0.44	1.47$\pm$0.58	0.85$\pm$0.24	1.24$\pm$0.33	1.01 $\pm$0.24	0.97$\pm$0.17
`sklearn` tree	26.0$\pm$1.69	30.6$\pm$5.91	20.8$\pm$0.42	21.7$\pm$0.41	20.4$\pm$1.78	23.1$\pm$0.49
Neural Net.	38.1$\pm$6.5	58.5$\pm$14.4	38.5$\pm$5.3	38.4$\pm$8.8	33.2 $\pm$6.5	38.3$\pm$7.25

	SpaceInv	Tennis	HalfCh.	Hopper	Swimmer	Walker2d
Program	1.07$\pm$0.22	0.9$\pm$0.36	0.18$\pm$0.04	0.25$\pm$0.04	0.19$\pm$0.03	0.25$\pm$0.06
`sklearn` tree	40.9$\pm$9.22	20.3$\pm$0.45	16.5$\pm$1.72	20.2$\pm$1.5	19.4$\pm$1.3	23.8$\pm$4.01
Neural Net.	41.9$\pm$10.4	34.8$\pm$2.8	54.2$\pm$24.0	66.6$\pm$22.2	56.3$\pm$20.8	60.9$\pm$18.5

D Feature importances for shortcut learning identification.

In this section we look at the feature importances of tree programs with 16 total nodes returned by INTERPRETER on MuJoCo and Atari. Some clear importances are for the Hopper robot that needs to move forward by jumping on one leg: INTERPRETER bases its control on the z-coordinates of the torso. Some other clear importances are: for Pong where program bases its decision on the y-distance between the player's pad and the ball; for Freeway where the most important is the chicken's y-speed; for Asterix it is the y-distance to the collectible... For Seaquest and Kangaroo where the goals are respectively to save divers and to get up to save its joey, we notice that the most important INTERPRETER concepts for the oracle do not include those goals. When visualizing the oracle network or the INTERPRETER tree program playing those games we indeed notice that they resepctively fight sharks and fight monkeys which are rewarded by the MDP but that are not the original games goals.

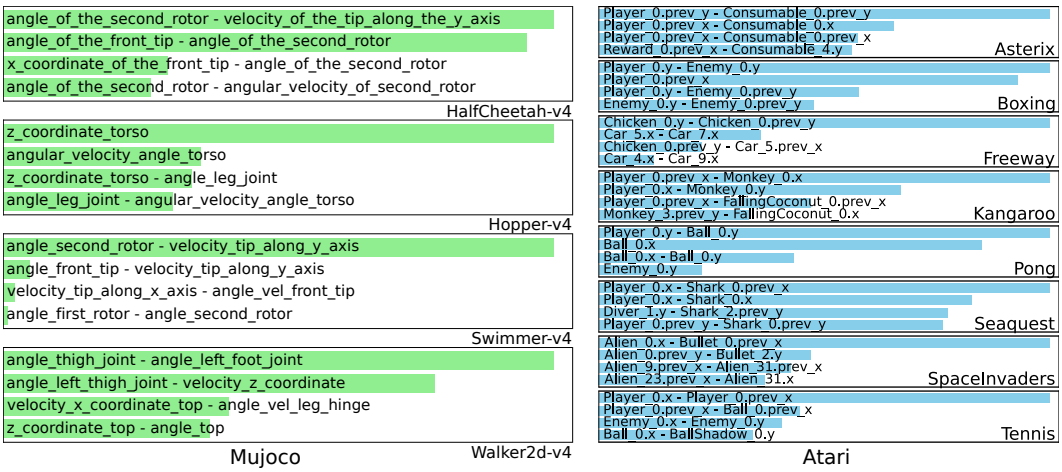


Figure 10: Feature importances on Mujoco (Left) and Atari (Right) environments.

E Extracted programs

In this section, we show some programs autonomously learned by INTERPRETER.

```
if diver_0.x > 0: # edited
    return "UP"
if Player_0.prev_x - Shark_0.prev_x  <= 0.51:
    if Player_0.prev_y - Shark_0.prev_y  <= -0.56:
        if Shark_0.y - Submarine_1.prev_x  > 1.24 and Player_0.prev_x - Shark_0.x  <= -0.71:
            return "DOWNRIGHTFIRE"
        else:
            return "DOWNFIRE"
    else:
        if Diver_1.prev_y - Shark_2.y  <= 0.92:
            if Player_0.prev_x - Shark_0.x  <= -0.58 or Shark_0.prev_y - Submarine_1.prev_x  <= -3.89:
                return "UPRIGHTFIRE"
            else:
                if Shark_0.y <= 0.93:
                    return "UPFIRE"
                else:
                    return "DOWNFIRE"
        else:
            return "DOWNRIGHTFIRE"
else:
    if Shark_0.y <= 0.53:
        if Player_0.prev_x - Submarine_0.prev_y  <= 0.13:
            return "UPLEFTFIRE"
        else:
            if Diver_1.y - Shark_0.prev_y  <= 1.87:
                if Player_0.prev_y - PlayerMissile_0.prev_y  <= -0.31:
                    return "DOWNFIRE"
                else:
                    return "DOWNLEFT"
            else:
                return "DOWNLEFT"
    else:
        if Shark_0.y - PlayerMissile_0.prev_y  <= 0.18:
            if Player_0.prev_x - Shark_0.x  <= 1.22:
                return "UPFIRE"
            else:
                return "UPLEFTFIRE"
        else:
            return "DOWNLEFTFIRE"
```

Figure 11: Program to play Seaquest returned by INTERPRETER. The first two lines have been edited by hand to allow the save of divers.

```
if Player.y - Ball.y  <= -0.59:
    if Ball.x <= -0.61:
        return "RIGHT"
    else:
        if Player.y - Ball.y  <= -0.81:
            return "LEFT"
        else:
            return "NOOP"
else:
    if Ball.x <= 0.05:
        if Enemy.y <= 0.25:
            return "LEFT"
        else:
            return "RIGHT"
    else:
        if Ball.x - Ball.y > -0.20 and Player.y - Ball.y > -0.43:
            return "RIGHT"
        else:
            return "NOOP"
```

Figure 12: Program to play Pong returned by INTERPRETER. Achieving a score of 15.5.

F Program simplification

We here show how programs can further be simplified, by merging redundant branches. To reduce the number of nodes/if-else statements in the program, one can for example replace the first block of the following code by the second one:

```
if Ball.x - Ball.y  <= -0.20:
    return "NOOP"
else:
    if Player.y - Ball.y  <= -0.43:
        return "NOOP"
    else:
        return "UP"

# can be written as

if (Ball.x  -  Ball.y) > -0.20 and (Player.y - Ball.y > -0.43):
        return  "UP"
else:
        return  "NOOP"
```

This mainly improves human interpretability, as decisions for specific actions are gathered together.

G Related work summary table

In table 3 we summarize existing explainable RL work and compare w.r.t to their required oracle knowledge, their domain ranges, on what problems they were tested, and on the level of explainability they provide. Among the algorithms that require at least an oracle policy, INTERPETER is the most versatile and well-tested method.

Table 3: Existing interpretable RL algorithms and their specifications.

Name	Prior Knowledge	$\mathcal{M}$	Benchmarks	Programs
INTERPRETER	π or Q	All	All	Interpretable
VIPER	π and Q	$A \in \mathbb{Z}^p$	Toy	Explainable
PIRL	π or Q and Primitives	All	Toy	Interpretable
SCoBots	π or Q and Primitives and LLM	All	Atari	Interpretable
NUDGE	Primitives	$A \in \mathbb{Z}^p$	Atari	Interpretable
LEAPS-based	Domain Specific Languages	$A \in \mathbb{Z}^p$	Karel	Interpretable
Demo2Code & Code as Policies	LLM	$A \in \mathbb{R}^q$	Robotics	Interpretable
INSIGHT	LLM	$A \in \mathbb{Z}^p$	Atari	Explainable
Custard	Primitives	$A \in \mathbb{Z}^p$	Toy	Interpretable
π-PRL	Primitives	All	All	Explainable
Differentiable Trees	Detailed tree sturcture	$A \in \mathbb{Z}^p$	Toy	Explainable
π-PRL	Primitives	All	All	Explainable

H User study details

Hereafter is provided the detailed questions used during our user studies. We have reached out to AI experts. We have collected and aggregated the answers of 19 participants.

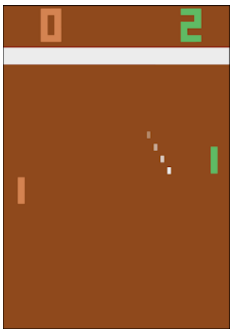
Interpretable policies - user study

You are participating in a user study on interpretability of reinforcement policies.
The policy is the function responsible for choosing an action for a specific state.
You will be presented with 4 different policy types, coming from 4 different methods.
You are asked to explain which policy is, in your opinion, the most interpretable.

Sign in to Google to save progress. Further information

* Specifies a required question

Visual rendering of the Pong environment.



Description of the Pong environment.
These policies are from agents trained on the Pong environments. In this environment, the agent controls the green paddle (on the right).
Its goal is to touch the ball (with its paddle) by adjusting its vertical position (on the y-axis) and return past the enemy's paddle (orange, on the left).
At each step, the agent observes the position (horizontal: x axis and vertical: y axis) of the ball, its green paddle, and the enemy's orange paddle. The values of the positions are normalized, but can be translated back to pixel values. We provide you with the policies using the normalized values.
It has to select an action between move the paddle **UP**, move it **DOWN** or do nothing (**NOOP**).

Please provide your name: * *

My answer

Do you accept that your answer may be used in a research publication ? * *

- ◯ Yes
- ◯ No No

Evaluation of interpretability
In the following, you are asked to evaluate for each method:
1. the level of**inherent explainability**: are you able to understand how crucial property (e.g. player's vertical position) of the state is for the agent's decision. If this element was to change it a certain way, would that impact the decision.
2. the level of **interpretability** : can you follow how each element is used by the policy to select an action.
3. the level of **trust** you have in each of these policies: can you check that it selects the correct action for the correct reasons. These action lead to a consistent behavior.
4. your ability to easily correct the programs, in case of suboptimal behavior

Please look at each policy first
We kindly ask you to **look at all of the 4 evaluated policies before answering the questions**. You do not have to carefully read them, just understand each principle before you report your evaluations

Policy 1 description
This policy is encoded with polynomials equation, that rely on the last 4 observed states. For this policy, x-player,4 corresponds to the last observed x position. It first extract 4 learned features (t1 to t4) and combine them to evaluate each action. The action with the highest evaluation is selected.

Policy 1

$t_1 = (-0.31x_{player,1} - 0.19x_{player,2} - 0.19x_{player,3} - 0.34x_{player,4} - 0.018x_{ball,1} - 0.014x_{ball,2} - 0.013x_{ball,3} - 0.014x_{ball,4} - 1.0x_{enemy,1} + 0.36x_{enemy,4} - 0.02y_{player,1} - 0.021y_{player,2} - 0.022y_{player,3} - 0.018y_{player,4} - 0.018y_{ball,1} - 0.018y_{ball,2} - 0.015y_{ball,3} - 0.018y_{ball,4} - 0.13y_{enemy,1} - 0.017y_{enemy,2} - 0.018y_{enemy,3} - 0.017y_{enemy,4} - 0.95)^2$
$t_2 = 1$
$t_3 = (-0.26x_{player,1} - 0.14x_{player,2} - 0.34x_{player,3} - 0.12x_{player,4} - 1.0x_{enemy,4} - 0.027y_{player,1} - 0.14y_{player,2} - 0.23y_{player,3} - 0.39y_{player,4} - 0.013y_{ball,1} - 0.013y_{ball,2} - 0.012y_{ball,3} - 0.012y_{enemy,1} - 0.2y_{enemy,2} - 0.28y_{enemy,3} - 0.31y_{enemy,4} - 0.83)^3$
$t_4 = (0.11$xplayer,1 $+0.24x_{player,2} + 0.17x_{player,3} + 0.14x_{player,4} + 0.013x_{enemy,4} + 0.37y_{player,1} + 0.15y_{player,2} + 0.085y_{player,4} + 0.51y_{enemy,1} + 0.013y_{enemy,2} + 1.0)^3$

NOOP $= -0.15t_1 - 0.53t_2 + 0.24t_3 + 0.31t_4$
UP $= -1.1t_1 + 0.15t_2 - 0.21t_3 - 0.041t_4$
DOWN $= 1.3t_1 + 0.95t_2 + 0.11t_3 - 0.32t_4$

Please check all policies before answering
Gentle reminder, please give a look at the other policies before answer the following questions.

Explainability level * *

	0	0	1	2	3	4	5	5	6	6	7	8	9	9	10	
Not explainable	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Fully explainable

Interpretability level * *

	0	0	1	2	3	4	5	5	6	6	7	8	9	9	10	
Not interpretable	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Fully interpretable

Trust level * *

	0	0	1	2	3	4	5	5	6	6	7	8	9	9	10	
I do not trust	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	I fully trust

Adjustment level * *

	0	0	1	2	3	4	5	5	6	6	7	8	9	9	10	
I cannot change anything	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	I can easily correct

Policy 2 description
This policy is encoded in a set of consecutive rules. The first rules for each the condition is True is selected (otherwise NOOP). It uses functions such as LT (linear trajectory), D (distance), C(center), ... etc.
For this policy, the actions **LEFT** and **LEFTFIRE** correspond to **DOWN**, while **RIGHT** and **RIGHTFIRE** correspond to **UP**.

Policy 2 (1/2)

IF (0.0917 | 1.0000)[(ED(Player1,Ball1) <= -1.2444) AND (Enemy1.y[t-1] <= -0.051)] OR (0.0955 | 1.0000)[(D(Player1,Ball1).y <= 0.6356) AND (ED(Player1,Ball1) <= -1.2444) AND (LT(Enemy1,Player1).y > 0.0884) AND (LT(Player1,Ball1).y <= 0.0781)] OR (0.0966 | 1.0000)[(LT(Ball1,Ball1).x <= -2.4804) AND (LT(Ball1,Enemy1).x > -0.8434) AND (LT(Player1,Ball1).x > 0.3434)] OR (0.0971 | 1.0000)[(Ball1.y <= 0.3469) AND (D(Player1,Enemy1).y <= 1.2407) AND (LT(Ball1,Enemy1).y > 1.2496)] OR (0.0997 | 1.0000)[(D(Player1,Ball1).y <= -0.7089) OR (0.6706 | 1.0000)[(D(Enemy1,Ball1).y <= -0.6761) AND (DV(Ball1).x <= -0.2477) AND (Enemy1.y <= 0.1317) AND (LT(Enemy1,Player1).y > -0.8251) AND (LT(Player1,Ball1).y <= -0.1017)] OR (0.8575 | 1.0000)[(LT(Ball1,Player1).x <= -0.4534) AND (LT(Enemy1,Ball1).y > 0.5509)] OR (0.8866 | 1.0000)[(Ball1.x > 1.1111) AND (DV(Ball1).x <= -0.372) AND (LT(Ball1,Enemy1).y > 0.4359) AND (LT(Enemy1,Player1).x <= -0.5486)] **THEN RIGHTFIRE**
IF (0.1963 | 1.0000)[(DV(Player1).y <= 1.0061) AND (LT(Enemy1,Ball1).y <= -0.3765) AND (Player1.y <= 0.5449)] OR (0.2026 | 1.0000)[(ED(Player1,Ball1) <= -1.283) AND (LT(Enemy1,Ball1).y > -0.3765) AND (LT(Player1,Player1).x > 0.3894) AND (V(Player1).x <= 1.2976)] OR (0.2822 | 1.0000)[(D(Enemy1,Ball1).y <= -1.1379) AND (DV(Player1).y > 0.0033) AND (Player1.y <= 0.3218) AND (V(Player1).x <= 0.216)] OR (0.3124 | 1.0000)[(D(Enemy1,Ball1).y <= 0.0936) (LT(Player1,Enemy1).x > 0.0673)] OR (0.5998 | 1.0000)[(C(Player1,Enemy1).y > 1.6676) AND (D(Player1,Ball1).y <= 0.6949) AND (LT(Enemy1,Player1).y <= -0.0033)] OR (0.6133 | 1.0000)[(C(Player1,Enemy1).y > 1.275) AND (DV(Ball1).x > 0.2497) AND (LT(Ball1,Enemy1).y > -0.8891) AND (LT(Enemy1,Ball1).y <= 0.0257)] OR (0.6140 | 1.0000)[(D(Player1,Ball1).y > 0.1017) AND (Enemy1.y[t-1] > -0.6471) AND (LT(Ball1,Enemy1).y <= -0.2139) AND (LT(Ball1,Enemy1).y > -0.7072) AND (LT(Ball1,Player1).y > 0.9352)] OR (0.6162 | 1.0000)[(Ball1.y > 0.9122) AND (LT(Ball1,Enemy1).y <= -0.8103)] OR (0.6177 | 1.0000)[(Ball1.y > -0.3441) AND (LT(Ball1,Enemy1).y <= -0.6384) AND (LT(Ball1,Enemy1).x <= -0.5385)] **THEN LEFTFIRE**
IF (0.2451 | 1.0000)[(Ball1.x > -0.4959) AND (Enemy1.y <= -0.3729) AND (LT(Enemy1,Ball1).y <= 0.6016)] OR (0.5661 | 1.0000)[(D(Player1,Enemy1).y <= 0.418) AND (DV(Player1).y > -0.4524) AND (Enemy1.y <= -0.1894) AND (LT(Player1,Ball1).y > 0.0981) AND (V(Enemy1).x <= -0.2862) AND (V(Enemy1).x > -0.4639) AND (V(Player1).x <= 0.892)] OR (0.5780 | 1.0000)[(Ball1.x[t-1] <= 0.2369) AND (D(Player1,Ball1).y <= 0.2995) AND (DV(Player1).y > -0.9082) AND (D(Enemy1,Ball1).y <= 2.9158) AND (D(Enemy1,Ball1).y > -0.83) AND (DV(Ball1).x <= 0.2497) AND (LT(Enemy1,Player1).y <= -0.5572) AND (LT(Enemy1,Player1).y > -2.0275) AND (LT(Player1,Ball1).y <= 0.0504) AND (LT(Player1,Ball1).y > -0.0967) AND (Player1.y[t-1] <= 0.2106)] OR (0.8152 | 1.0000)[(D(Enemy1,Ball1).y <= 0.2476) AND (ED(Player1,Ball1) <= 0.2956) AND (LT(Enemy1,Player1).x <= -0.3492) AND (LT(Player1,Ball1).y > 0.2783)] OR (0.8266 | 1.0000)[(DV(Player1).y > 1.3707) AND (LT(Enemy1,Ball1).y > 0.5967)] OR (LT(Ball1,Player1).x <= -0.6139) AND (LT(Enemy1,Player1).x > -0.5088) AND (Player1.y[t-1] > 0.2106)] OR (0.8315 | 1.0000)[(ED(Player1,Ball1) <= -1.1579) AND (LT(Enemy1,Player1).y <= -2.0275)] OR(LT(Enemy1,Player1).y <= -2.0277)] **THEN NOOP**

Policy 2 (2/2)

IF (0.4241 | 1.0000)[(Ball1.x > 1.0875) AND (D(Player1,Ball1).y <= 0.2204) AND (D(Player1,Ball1).y > -0.3926) AND (LT(Enemy1,Ball1).y <= -0.4178) AND (V(Player1).x <= -0.1897)] OR (0.4791 | 1.0000)[(LT(Enemy1,Ball1).y > 0.5628)] OR (0.5181 | 1.0000)[(Ball1.x > 0.4258) AND (D(Enemy1,Ball1).y <= -0.7787) AND (D(Player1,Ball1).y <= 0.2401) AND (DV(Ball1).x <= -0.9937)] OR (0.5634 | 1.0000)[(D(Player1,Ball1).y <= 0.6356) AND (DV(Ball1).y > -0.3641) AND (ED(Player1,Enemy1) <= -0.2204) AND (Enemy1.y <= -0.7399) AND (LT(Enemy1,Ball1).y > 0.1768)] OR (0.6460 | 1.0000)[(D(Player1,Ball1).y <= 0.2995) AND (LT(Enemy1,Ball1).y > 0.5306) AND (LT(Enemy1,Player1).x > -0.5486)] OR (0.6760 | 1.0000)[(LT(Enemy1,Ball1).y > 0.147) AND (V(Ball1).x > 8.2212)] OR (0.6782 | 1.0000)[(DV(Ball1).y > 6.8821)] OR (0.7189 | 1.0000)[(DV(Ball1).x <= -0.9937) AND (LT(Enemy1,Ball1).y > 0.1883)] OR (0.7267 | 1.0000)[(DV(Ball1).x <= -0.745) AND (LT(Enemy1,Ball1).y > 0.2356) AND (V(Player1).x <= 1.0272)] OR (0.7375 | 1.0000)[(D(Player1,Ball1).y <= 0.2599) AND (DV(Ball1).x <= -0.745) AND (LT(Enemy1,Ball1).y > 0.1842)] OR (0.7800 | 1.0000)[(DV(Ball1).x <= -8.5779)] OR (0.7800 | 1.0000)[(LT(Ball1,Player1).y <= -0.1638) AND (LT(Enemy1,Ball1).y > 0.5865)] OR (0.7937 | 1.0000)[(D(Player1,Ball1).y <= -0.096) AND (DV(Ball1).x <= -0.9937) AND (LT(Enemy1,Ball1).y > 0.1659)] OR (0.7990 | 1.0000)[(DV(Ball1).y > 6.3909)] OR (0.8330 | 1.0000)[(DV(Ball1).y > 7.005)] OR (0.8762 | 1.0000)[(LT(Enemy1,Ball1).y > 0.5628) AND (Player1.y[t-1] > 1.1239)] OR (0.8990 | 1.0000)[(D(Player1,Ball1).y <= 0.2797) AND (LT(Enemy1,Ball1).y > 0.5628)] OR (0.9015 | 1.0000)[(LT(Enemy1,Ball1).y > 0.1601) AND (V(Ball1).x > 9.8652)] OR (0.9087 | 1.0000)[(D(Player1,Ball1).y <= 0.3192) AND (LT(Enemy1,Ball1).y > 0.5628)] OR (0.9217 | 1.0000)[(D(Player1,Ball1).y <= 0.2401) AND (LT(Enemy1,Ball1).y > 0.5865)] OR (0.9351 | 1.0000)[(LT(Player1,Ball1).y <= -0.0181) AND (V(Ball1).x > 9.8652)] **THEN RIGHT**
IF (0.5094 | 1.0000)[(LT(Enemy1,Player1).x > 1.206) AND (V(Ball1).x > 0.0798)] OR (0.5098 | 1.0000)[(DV(Player1).y > 0.8238) AND (ED(Player1,Ball1) <= -1.6523) AND (Enemy1.y[t-1] <= 1.0954)] OR (0.5451 | 1.0000)[(C(Player1,Enemy1).y <= -1.7581) AND (D(Player1,Ball1).y <= 0.9519) AND (DV(Ball1).y <= 0.1272) AND (ED(Player1,Ball1) <= 0.4612) AND (ED(Player1,Ball1) > -1.2069) AND (LT(Enemy1,Ball1).y <= 0.2721) AND (LT(Enemy1,Enemy1).x > -0.0393)] OR (0.5670 | 1.0000)[(D(Player1,Enemy1).y > 1.2818) AND (ED(Player1,Ball1) > -0.0316) AND (V(Enemy1).x <= -0.1085)] OR (0.5696 | 1.0000)[(D(Player1,Ball1).y > 0.2401) AND (LT(Enemy1,Player1).y > 0.0049)] OR (0.6164 | 1.0000)[(D(Player1,Enemy1).y > -0.3842) AND (DV(Ball1).y <= 0.1272) AND (DV(Enemy1).y <= 0.5079) AND (Enemy1.y[t-1] <= -0.372) AND (LT(Ball1,Enemy1).y > -0.7974) AND (LT(Ball1,Player1).y <= -0.6658) AND (LT(Enemy1,Enemy1).x > -3.0168)] OR (0.6242 | 1.0000)[(C(Player1,Enemy1).y > 0.6792) AND (D(Enemy1,Ball1).y > 0.0936) AND (LT(Ball1,Player1).y > 2.4043)] OR (0.6396 | 1.0000)[(Ball1.x[t-1] <= -0.0231) AND (D(Player1,Enemy1).y <= 0.6031) AND (DV(Ball1).x > 0.001) AND (Enemy1.y > 0.1775) AND (LT(Player1,Ball1).y <= -0.117)] OR (0.6446 | 1.0000)[(DV(Ball1).y <= 0.25) AND (DV(Player1).y > 0.8238) AND (ED(Player1,Ball1) > -0.4118) AND (Enemy1.y[t-1] <= 1.0954) AND (LT(Ball1,Ball1).x > 0.3106) AND (LT(Player1,Enemy1).y > -1.0363) AND (V(Enemy1).x > 0.2469) AND (V(Player1).x > 0.4864)] OR (0.6735 | 1.0000)[(Enemy1.y > -0.3271) AND (LT(Ball1,Ball1).x <= -1.8784) AND (LT(Player1,Ball1).x > 0.5749)] OR (0.6754 | 1.0000)[(C(Enemy1,Ball1).y > -1.4068) AND (Player1.y > 0.2995)] **THEN LEFT**
IF (0.5577 | 1.0000)[(DV(Ball1).x > 0.747) AND (ED(Player1,Ball1) <= -0.0818) AND (LT(Ball1,Enemy1).x <= -0.5385) AND (LT(Player1,Enemy1).x > -0.7338)] OR (0.5953 | 1.0000)[(DV(Ball1).x > 0.747) AND (ED(Player1,Enemy1) <= -0.6485) AND (LT(Player1,Enemy1).x > -0.7338)] **THEN FIRE**

Explainability level * *

0 0 1 2 3 4 5 5 6 6 7 8 9 9 10

Not explainable — Fully explainable

Interpretability level * *

0 0 1 2 3 4 5 5 6 6 7 8 9 9 10

Not interpretable — Fully interpretable

Trust level * *

0 0 1 2 3 4 5 5 6 6 7 8 9 9 10

I do not trust — I fully trust

Adjustment level * *

0 0 1 2 3 4 5 5 6 6 7 8 9 9 10

I cannot change anything — I can easily correct

Policy 3 description

This policy is encode as logic rules. It uses predicates (in purple) evaluated on the detected objects (in red). Each rule value is evaluated according to the valuation function of each predicate. Then, the rule is multiplied with its weight (on the left). Each rule thus has a valuation. The valuations of the rules that encode the same action are then aggregated using a softor to obtain the final valuation of each action.
The action with the highest valuation is selected

Policy 3

0.38: **UP**:- type(O1, Ball), type(O2, Player), above(O1, O2), going_right(O1)
0.42: **UP**:- type(O1, Ball), type(O2, Player), above(proj_y(O1, O2), O2)
0.25: **UP**:- type(O1, Enemy), type(O2, Player), above(proj_y(O1, O2), O2)

0.47: **DOWN**:- type(O1, Ball), type(O2, Player), above(O2, O1), going_right(O1)
0.32: **DOWN**:- type(O1, Ball), type(O2, Enemy), below(proj_y(O1, O2), O2), going_right(O1)
0.21: **DOWN**:- type(O1, Player), type(O2, Ball), below(proj_y(O2, O1), O1), going_right(O2)

0.23: **NOOP**:- type(O1, Enemy), type(O2, Ball), closeby(O1, O2)
0.03: **NOOP**:- type(O1, Ball), type(O2, Player), closeby(O1, O2)
0.01: **NOOP**: type(O1, Ball), type(O2, Ball), above(O1, O2)

Explainability level * *

0 0 1 2 3 4 5 5 6 6 7 8 9 9 10

Not explainable — Fully explainable

Interpretability level * *

0 0 1 2 3 4 5 5 6 6 7 8 9 9 10

Not interpretable — Fully interpretable

Trust level * *

0 0 1 2 3 4 5 5 6 6 7 8 9 9 10

I do not trust — I fully trust

Adjustment level * *

0 0 1 2 3 4 5 5 6 6 7 8 9 9 10

I cannot change anything ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ I can easily correct

Policy 4 description
This policy is encoded within python code. Each object is passed to the function, and there properties are accessed through their corresponding attributes. The selected action is the returned one.

Policy 4

```
if Player.y - Ball.y  <= -0.59:
    if Ball.x <= -0.61:
        return "UP"
    elif Player.y - Ball.y  <= -0.81:
        return "DOWN"
    else:
        return "NOOP"
else:
    if Ball.x <= 0.05:
        if Enemy.y <= 0.25:
            return "DOWN"
        else:
            return "UP"
    else:
        if (Ball.x - Ball.y)  > -0.20 and (Player.y - Ball.y > -0.43):
            return "UP"
        else:
            return "NOOP"
```

Explainability level * *

0 0 1 2 3 4 5 5 6 6 7 8 9 9 10

Not explainable ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ Fully explainable

Interpretability level * *

0 0 1 2 3 4 5 5 6 6 7 8 9 9 10

Not interpretable ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ Fully interpretable

Trust level * *

0 0 1 2 3 4 5 5 6 6 7 8 9 9 10

I do not trust ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ I fully trust

Adjustment level * *

0 0 1 2 3 4 5 5 6 6 7 8 9 9 10

I cannot change anything ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ ○ I can easily correct

Open feedback (optional) Sans titre
What would you do to improve the explainability, interpretability and trust of each of these.

Answer

My answer

Sending Page 1 of 1 Delete all entries