Interpretable Concept Bottlenecks to Align Reinforcement Learning Agents

Anonymous Author(s)
Accepted to NeurIPS 2024 main Conference

Abstract

Goal misalignment, reward sparsity and difficult credit assignment are only a few of the many issues that make it difficult for deep reinforcement learning (RL) agents to learn optimal policies. Unfortunately, the black-box nature of deep neural networks impedes the inclusion of domain experts for inspecting the model and revising suboptimal policies. To this end, we introduce *Successive Concept Bottleneck Agents (SCoBots)*, that integrate consecutive concept bottleneck (CB) layers. In contrast to current CB models, SCoBots do not just represent concepts as properties of individual objects, but also as relations between objects which is crucial for many RL tasks. Our experimental results[1]) provide evidence of SCoBots' competitive performances, but also of their potential for domain experts to understand and regularize their behavior. Among other things, SCoBots enabled us to identify a previously unknown misalignment problem in the iconic video game, Pong, and resolve it. Overall, SCoBots thus result in more human-aligned RL agents.

1 Introduction

Deep Reinforcement learning (RL) agents are prone to suffer from a variety of issues that hinder them from learning optimal or generalizable policies. Prominent examples are *reward sparsity* [Andrychowicz et al., 2017] and *difficult credit assignment* [Raposo et al., 2021, Wu et al., 2024]. A more pressing issue is the *goal misalignment* problem. It occurs when an agent optimizes a different *side-goal*, aligned with the original *target goal* during training [Koch et al., 2021], but not at test time. Such misalignments can be difficult to identify [di Langosco et al., 2022]. For instance, we here discover that such a misalignment can occur in the oldest and most iconic video game, *Pong* (*cf.* Fig. 1). In Pong, the agent's target goal is to catch a ball with its own paddle and to return it passed the enemy's one. The enemy is programmed to constantly follow the ball, thus, agents can learn to focus on the position of the enemy paddle's for placing their own, rather than the position of the ball itself. This *shortcut learning* [Geirhos et al., 2020] phenomenon, where a model learns decision rules for the provided training set, that fail to transfer to novel data sets. If left unchecked, it may lead to a lack of model generalization and unintuitive failures *e.g.* at deployment time [Zhang et al., 2021].

Recently eXplainable AI (XAI) methods have emerged to detect such shortcut behavior, by identifying the reasons behind a model's decisions [Schramowski et al., 2020, Roy et al., 2022, Saeed and Omlin, 2023]. However, many XAI methods' explanation do not faithfully present the model's underlying decision process [Chan et al., 2022]. For example, importance-map explanations indicate the importance of an input element without indicating *why* this element is important [Kambhampati et al., 2021, Stammer et al., 2021, Teso et al., 2023]. A recent branch of research therefore focuses on models that provide inherent concept-based explanations. Prominent examples are concept bottlenecks models (CBMs), which provide predictive performances on par with standard deep learning approaches for supervised image classification [Koh et al., 2020]. More importantly, CBMs allow to identify and revise incorrect model behavior on a concept level [Stammer et al., 2021].

[1]Code available at https://anonymous.4open.science/r/SCoBots-69C4

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

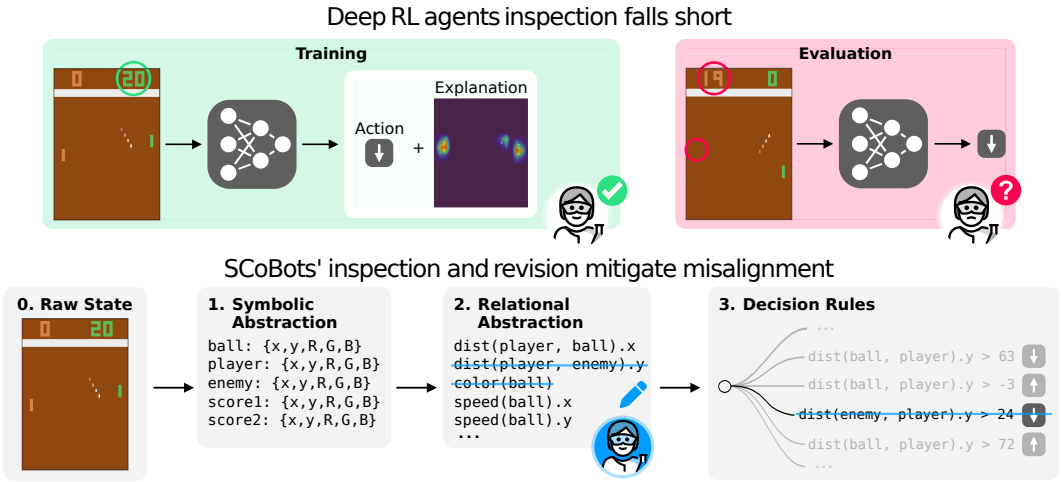

Figure 1: **Successive Concept Bottlenecks Agents (SCoBots) allow for easy inspection and revision.** Top: Deep RL agents trained on Pong produce high playing scores with importance map explanations that suggest sensible underlying reasons for taking an action (✅). However, when the enemy is hidden, the deep RL agent fails to even catch the ball without clear reasons (❓). Bottom: SCoBots, on the other hand, allow for multi-level inspection of the reasoning behind the action selection, *e.g.*, at a relational concept, but also action level. Moreover, they allow users to easily intervene on them (✏️) to prevent the agents from focusing on potentially misleading concepts. In this way, SCoBots can mitigate RL specific caveats like goal misalignment.

Contrary to exising CBMs' fields of applications, RL requires relational reasoning [Kaiser et al., 2019]. In this work, we therefore introduce Successive Concept Bottleneck Agents (SCoBots, *cf.* Fig. 1, bottom) bringing concept bottlenecks approach to RL. SCoBots integrate successive concept bottlenecks into their decision processes, where each bottleneck layer provides concept representations that integrate the concept representations of the previous bottleneck layer. Specifically, provided a set of predefined concept functions, SCoBots automatically extract relational concept representations based on objects and their properties of the initial bottleneck layers. Finally, the set of relational and object concept representations are used for optimal action selection. SCoBots thus represent inherently explainable RL agents and in comparison to deep RL agents, they allow for inspecting and revising their learned decision policies at multiple levels of their reasoning processes: from the single object properties, through the relational concepts to the action selection.

In our evaluations on the iconic Atari Learning Environments [Mnih et al., 2013] (ALE), we provide experimental evidence of the performance abilities of SCoBots that is on par with deep RL agents. More importantly, we showcase their ability to provide valuable explanations and the potential of mitigating a variety of RL specific issues, from reward sparsity to misalignment problems, via simple guidance from domain experts. In the course of our evaluations, we identify previously unknown shortcut behavior of deep RL agents, even on a simple game as Pong. By utilizing the interaction capabilities of SCoBots, this behavior is easily removed. Ultimately, our work thereby illustrates the severity of goal misalignment issues in RL and the importance of being able to mitigate these and other RL specific issues via *relational concept based models*. In summary, our contributions are:

(i) We introduce Successive Concept Bottleneck agents (SCoBots) that integrate relational concept representations into their decision processes.

(ii) We show that SCoBots allow to inspect their internal decision processes.

(iii) We show that the inherent inspectable nature of SCoBots can be utilized for identifying previously unknown misalignment problems of RL agents.

(iv) We show that they allow for human interactions for mitigating various RL specific issues.

We proceed as follows. We introduce SCoBots and discuss their specific properties. We continue with experimental evaluations and analysis. Before concluding, we touch upon related work.

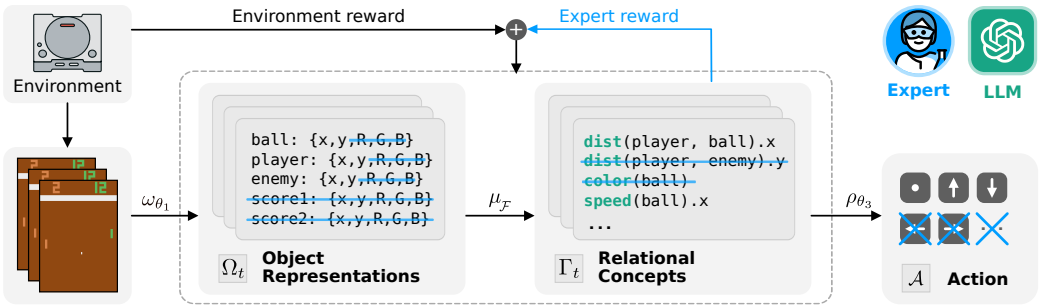

Figure 2: **An overview of Successive Concept Bottlenecks Agents (SCoBots).** SCoBots decompose the policy into consecutive interpretable concept bottlenecks (ICB). Objects and their properties are first extracted from the raw input, human-understandable functions are then employed to derive relational concepts, used to select an action. The understandable concepts enable interactivity. Each bottleneck allows expert users to, *e.g.*, prune or utilize concepts to define additional reward signals.

2 Successive Concept Bottleneck Agents

RL problems are modelled as Markov decision processes (MDPs), and also of concept bottleneck models (CBMs). In this work, we represent an RL problem through the framework of a Markov Decision Process, $\mathcal{M} =< \mathcal{S}, \mathcal{A}, P_{s,a}, R_{s,a} >$, with $\mathcal{S}$ as the state space, $\mathcal{A}$ the set of available actions, $P(s,a)$ the transition probability, and $R(s,a)$ the immediate reward function, obtained from the environment. Classic deep RL policies, $\pi_\theta(s) = P(A = a|S = s)$ parameterized by θ, are usually black-box models that process raw input states, *e.g.* a set of frames, to provide a distribution over the action space [Mnih et al., 2015, van Hasselt et al., 2016].

Concept bottleneck models (CBMs) initiate a learning process by extracting relevant concepts from raw input (*e.g.* image data). These concepts can represent the color or position of a depicted object. Subsequently, these extracted concepts are used for downstream tasks such as image classification. Formally, a bottleneck model $g : x \to c$ transforms an input $x \in \mathbb{R}^D$ with D dimensions into a concept representation $c \in \mathbb{R}^k$ (a vector of k concepts). Next, the predictor network $f : c \to y$ uses this representation to generate the final target output (*e.g.* $y \in \mathbb{R}$ for classification problems).

The main body of research on CBMs focuses on image classification tasks, where extracted concepts represent attributes of objects. In contrast, RL agents must learn not just from static data but also through interaction with dynamic environments. RL tasks thus often require relational reasoning, as they involve understanding the relationships between instances that evolve through time and interact with another. This is crucial for learning effective policies in complex, dynamic environments. Note that, in the following descriptions, we use specific fonts to distinguish objects' `properties` from their **`relations`**.

2.1 Building inspectable ScoBots

An underlying assumption of SCoBots is that their processing steps should be inspectable and understandable by a human user. This stands in contrast to *e.g.* unsupervised CBM approaches Jabri et al. [2019], Zhou et al. [2019], Srinivas et al. [2020], where there is no guarantee for learning human-aligned concepts. SCoBots rather take a different approach by dividing the concept learning and RL-specific action selection into several inspectable steps that are grounded in human-understandable concept representations. These steps are described hereafter and depicted in Fig. 2.

Similar to other RL agents, SCoBots process the last n observed frames from a sequence of images, $s_t = \{x_i\}_{i=t-n}^{t}$. For each frame, SCoBots need an initial concept extraction method to extract objects and their properties, as in previous works on CBMs [Stammer et al., 2021, Koh et al., 2020]. Specifically, we start from an initial bottleneck model, $\omega_{\theta_1}(\cdot)$, that is parameterized by parameter set, θ_1. Given a state, s_t, this model provides a set of c_i object representations per frame, $\omega_{\theta_1}(s_t) = \Omega_t = \{\{o_i^j\}_{j=1}^{c_i}\}_{i=t-n}^{t}$, where each object representation corresponds to a tensor of different extracted properties of that object, (*e.g.* its `category`, `position` (*i.e.* x, y coordinates), etc.). As done in previous works on CBMs, the bottleneck model of our SCoBots, ω_{θ_1}, can correspond to a model that was supervisedly pretrained for extracting the objects and their properties from images.

One of the major differences to previous work on CBMs is that SCoBots further extract and utilize *relational* concepts that are based on the previously extracted objects and their properties. Formally, SCoBots utilize a consecutive bottleneck model, $\mu_{\mathcal{F}}(\cdot)$, called the relation extractor. This model, μ, is parameterized by a predetermined set of transparent relational functions, $\mathcal{F}$, used to extract a set of d_t relational concepts. These relations are based on each individual object (and its properties) for unary relations or on combinations of objects for n-ary relations and are denoted as $\mu_{\mathcal{F}}(\Omega_t) = \Gamma_t = \{g_t^k\}_{k=1}^{d_t}$. Without strong prior knowledge, $\mathcal{F}$ can initially correspond to universal object relations such as **`distance`** and **`speed`**. However, this set can easily be updated on the fly by a human user with *e.g.* additional, novel relational functions. Note that $\mathcal{F}$ can include the identity, to let the relation extractor pass through initial object concepts from Ω_t to the relational concepts Γ_t.

Finally, SCoBots employ an action selector, ρ_{θ_2}, parameterized by θ_2, on the relational concepts to select the best action, a_t, given the initial state, s_t (*i.e.* the set of frames). Up to now, all extracted concepts in SCoBots, both Ω_t and Γ_t, represent human-understandable concept representations. To guarantee understandability also within the action selection step of SCoBots, we need to embed an interpretable action selector. While neural networks, a standard choice in RL literature, are performative and easy to train using differentiable optimization methods, they lack this required interpretability feature. However, Çağlar Aytekin [2022] have recently shown the equivalence between ReLU-based neural networks and decision trees, which in contrast represent inherently interpretable models. To trade-off these issues of flexibility and performance vs interpretability, SCoBots thus break down the action selection process by initially training a small ReLU-based neural network action selector, $\widetilde{\rho}_{\theta_2'}$, via gradient-based RL optimization. After this, $\widetilde{\rho}_{\theta_2'}$ is finally distilled into a decision tree ρ_{θ_2}. Lastly, note that one can add a residual link from the initial object concepts, Ω_t, to the relational concepts, Γ_t such that the action selector can, if necessary, also make decisions based on basic object properties, *e.g.*, the height of an object.

Overall, our approach preserves the MDP formulation used by classic deep approaches, but decomposes the classic deep policy $s_t \xrightarrow{\pi_\theta} a_t$ into a successive concept bottleneck one $s_t \xrightarrow{\omega_{\theta_1}} \Omega_t \xrightarrow{\mu_{\mathcal{F}}} \Gamma_t \xrightarrow{\rho_{\theta_2}} a_t$, where $\theta = (\theta_1, \theta_2, \mathcal{F})$ constitutes the set of policy parameters. For simplicity, we will discard the parameter notations in the rest of the manuscript. Further explanations of the input and output space of each module, as well as the properties and relational functions used in this work are provided in the appendix (*cf.* App.A.5 and App. A.6).

Instead of jointly learning object detection, concept extraction, and policy search, SCoBots enable independent optimization of each policy component. Separating the training procedure of different components reduces the complexity of the overall optimization problem [Koh et al., 2020].

2.2 Guiding SCoBots

The inspectable nature of SCoBots not only brings the benefit of improved human-understandability, but importantly allows for targeted human-machine interactions. In the following, we describe two guidance approaches for interacting with the latent representations of SCoBots: concept pruning and object-centric rewarding. We refer to the revised SCoBot agents as *guided* SCoBots in the following.

Concept pruning. The type and amount of concepts that are required for a successful policy may vary across tasks. For instance, the objects **`colors`** are irrelevant in Pong but required to distinguish vulnerable ghosts from dangerous ones in MsPacman. However, overloading the action selector with irrelevant concepts, whether these are object properties (Ω_t) or relational concepts (Γ_t), can lead to difficult optimization (*e.g.* the agent focusing on noise in unimportant states) as well as difficult inspection of the decision tree (ρ). Moreover, for a single RL task, the need for specific concepts might even change during training, in *e.g.* progressive environments (where agents need to master early stages before being provided with additional, more complex tasks [Delfosse et al., 2024]).

SCoBot's human-comprehensible concepts therefore allow domain experts to prune unnecessary concepts. In this way, expert users can guide the learning process towards relevant concepts. Formally, users can (i) select a subset of the object property concepts, $\overline{\Omega}$. Additionally, by (ii) selecting a subset of the relational functions, $\overline{\mathcal{F}}$, or (iii) specifying which objects specific functions should be applied on, experts can implicitly define the relational concepts subset, $\overline{\Gamma}$. Guided SCoBots thus formally refining the policy extraction to $s_t \xrightarrow{\omega_{\theta_1}} \Omega_t \rightarrow \overline{\Omega}_t \xrightarrow{\mu_{\overline{\mathcal{F}}}} \overline{\Gamma}_t \xrightarrow{\rho_{\theta_2}} a_t$.

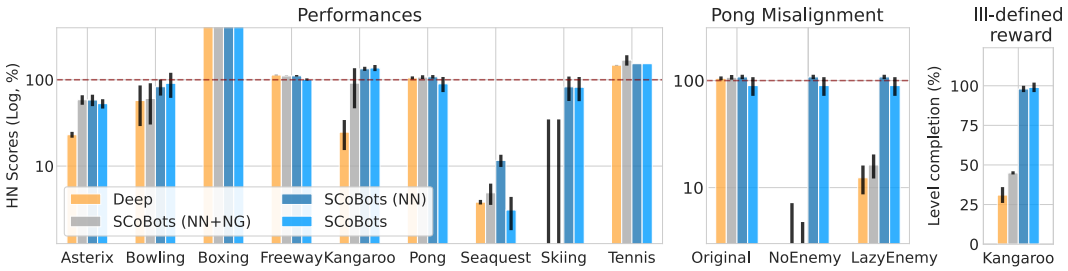

Figure 3: **Object-centric agents can master different Atari environments and interactive SCoBots allow for corrections.** Human-normalized scores of deep and SCoBots agents trained using PPO on 9 ALE environments, including 3 SCoBots variants: non-guided deep policy (NN+NG), guided deep policy (NN) and guided decision tree policy (SCoBots). SCoBots obtain similar or better scores than the deep agents, showing that object-centric agents can also solve RL tasks while making use of human-understandable concepts (left). Guiding SCoBots allow to correct misalignment in Pong (center) and to obtain the originally intended agents, depicted by a level complition score of 100% on the intended goal's evaluation in Kangaroo (right).

Furthermore, users can (iv) prune out redundant actions resulting in a new action space $\overline{\mathcal{A}}$.

To give brief examples of these four pruning possibilities we refer to Fig. 2, focusing here on the blue subparts. Particularly, in Pong a user can remove objects (*e.g.* scores) or specific object properties (*e.g.* R,G,B values) to obtain $\overline{\Omega_t}$. Second, the color relation, `color(·)`, is irrelevant for successfully playing Pong and can therefore be removed from $\mathcal{F}$. Third, the vertical distance function (`dist(·,·).y`) can be prevented from being applied to the (player, enemy) input couple. These pruning actions provide SCoBots with $\overline{\Gamma_t}$. Lastly, the only playable actions in Pong are `UP` and `DOWN`. To ease the policy search, the available `FIRE`, `LEFT` and `RIGHT` from the base Atari action space might be discarded to obtain $\overline{\mathcal{A}}$, as they are equivalent to `NOOP` (*i.e.* no action).

Importantly, being able to prune concepts can help mitigate misalignement issues such as those of agents playing Pong (*cf.* Fig. 1), where an agent can base its action selection on the enemy's position instead of the ball's one. Specifically, pruning the enemy position from the distance relation concept enforces SCoBot agents to rely on relevant features for their decisions such as the ball's position, rather than the spurious correlation with the enemy's paddle.

Object-centric feedback reward. Reward shaping [Touzet, 1997, Ng et al., 1999] is a standard RL technique that is used to facilitate the agent's learning process by introducing intermediate reward signals, aggregated to the original reward signal. The object-centric and importantly *concept-based* approach of SCoBots allows to easily craft additional reward signals. Formally, one can use the extracted relational concepts to express a new expert reward signal:

$$R^{\text{exp}}(\Gamma_t) := \sum_{g_t \in \Gamma_t} \alpha_{g_t} \cdot g_t, \tag{1}$$

where $R^{\text{exp}} : \Gamma \longrightarrow \mathbb{R}$ and $\Gamma_t = \mu(\omega(s_t))$ is the relational state, extracted by the relation extractor. The coefficient $\alpha_{g_t} \in \mathbb{R}$ is used to penalize or reward the agent proportionally to relational concepts. Our expert reward only relies on the state, as we make use of the concepts extracted from it. However, incorporating the action into the reward computation is straightforward. In practice, this expert reward signal can lead to guiding the agent towards focusing on relevant concepts, but also help smoothing sparse reward signals, which we will discuss further in our evaluations.

For example, experts have the ability to impose penalties, based on the distance between the agent and objects (*cf.* "expert reward" arrow in Fig. 2), with the intention of incentivizing the agent to maintain close proximity with the ball. As we show in our experimental evaluation, we can use this concept pruning and concept based reward shaping to easily address many RL specific caveats such as reward sparsity, ill-defined objectives, difficult credit assignment, and misalignment (*cf.* App. A.8 for details on each problem).

3 Experimental Evaluations

In our evaluations, we investigate several properties and potential benefits of the SCoBot agents. We specifically aim to answer the following research questions:
(Q1) Are concept based agents able to learn competitive policies on different RL environments?
(Q2) Does the inspectable nature of SCoBots allow to detect issues in their decision processes?
(Q3) Can concept-based guidance help mitigate common RL caveats, such as policy misalignments?

Experimental setup: We evaluate SCoBots on 9 Atari games (*cf.* Fig. 3 (left) for complete list) from the Atari Learning Environments [Bellemare et al., 2012] (by far the most used RL framework (*cf.* App. A.1), as well as a variations of Pong where the enemy is not visible yet active *i.e.* still able to return the ball (*NoEnemy*), and where the enemy stops moving after returning the ball (*LazyEnemy*). We provide human normalized scores (following eq. 3) that are averaged over 3 seeds for each agent configuration. We compare our SCoBot agents to deep agents with the classic convolutional network introduced by Mnih et al. [2015], that process a stack of 4 black and white frames and denote these as deep agents in the following. Note, that we evaluate all agents on the latest *v5* version of the environments[2] to prevent overfitting. All agents are trained for 20M frames under the Proximal Policy Optimization algorithm (PPO, [Schulman et al., 2017]), specifically the stable-baseline3 implementation [Raffin et al., 2021] and its default hyperparameters (*cf.* Tab. 2 in App. A.5).
We focus our SCoBot evaluations on the more interesting aspects of the agent's reasoning process underlying the action selection, rather than the basic object identification step (which has been thoroughly investigated in previous works *e.g.* [Koh et al., 2020, Stammer et al., 2021, Wu et al., 2024]). We thus assume access to a pretrained object extractor and provide our agents with object-centric descriptions of these states (Ω_t) based on the information from OCAtari [Delfosse et al., 2023a]. Specifically, in our evaluations, the set of object properties consists of object class (*e.g.* enemy paddle, ball etc.), (x^t, y^t) coordinates, coordinates at the previous position (x^{t-1}, y^{t-1}), height and width, the most common RGB values (*i.e.* the most representative color of an object), and object orientation. The specific set of functions, $\mathcal{F}$, that are used to extract object relations is composed of: euclidean distance, directed distances (on x and y axes), speed, velocity, plan intersection, the center (of the shortest line between two objects), and color name (*cf.* App. A.5.1 for implementation details). To distill SCoBots' learned policies, we use the decision-tree extraction algorithm VIPER [Bastani et al., 2018]. For the **human-guided** SCoBot evaluations (denoted as (guided) SCoBots in our evaluations), we illustrate human guidance and prune out the concepts that we consider unimportant to master the game (*cf.* Tab. 4). Furthermore, to mitigate RL specific problems, we also provide SCoBot agents with simulated human-expert feedback signals, the details of which we describe at the relevant sections below. More details about the setup and the hyperparameters are in App. A.5.

SCoBots learn competitive policies (Q1).
We present human-normalized (HN) scores of both SCoBot and deep agents trained on each investigated game individually in Fig. 3 (left). Numerical values are provided in Tab. 1 (*cf.* App. A.7). We observe that SCoBot agents perform at least on par with deep agents on all games, even slightly outperform these on 5 out of 9 (namely, Asterix, Boxing, Kangaroo, Seaquest and Tennis). Our results suggest that SCoBots provide competitive performances on every tested game despite the constraints of multiple bottlenecks within their architectures. Overall, our experimental evaluations show that RL policies based on interpretable concepts extraction decoupled from the action selection process can, in principle, lead to competitive agents.

Inspectable SCoBots' to detect misalignments (Q2).
The main target of developing ScoBots is to obtain competitive, yet *transparent* agents that allow for human users to identify their underlying reasoning process. Particularly, for a specific state, the inspectable bottleneck structure of SCoBots allows to pinpoint not just the object properties, but importantly the relational concepts being used for the final action decision. This is exemplified in Fig. 4 on Skiing, Pong and Kangaroo (*cf.* App. A.4 for explanations of SCoBots on the remaining games). Here we highlight the decision-making path (from SCoBots' decision tree based policies), at specific states of each game. For example, for the game state extracted from Skiing, the SCoBot agent selects `RIGHT` as the best action, because the signed distance from its character to the left flag is larger than a specific value ($+15$ pixels). Given the nature of the game this inherent explanation suggests that the agent is indeed basing its selection process on relevant *relational* concepts.

[2]This leads to deep agents overall slightly worse than in Schulman et al. [2017], however we obtain similar results when evaluating on the old *v4* versions (*cf.* Tab. 1).

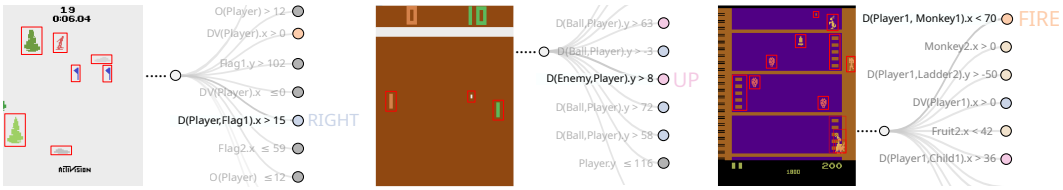

Figure 4: **SCoBots allow to follow their decision process,** because of their object centricity and interpretable concepts. The decision processes of SCoBots (extracted from the decision trees) with the associated states from a *guided* SCoBot on Skiing (left), and from normal SCoBots on Pong (middle) and Kangaroo (right). For example, in this Skiing state, our SCoBot selects `RIGHT`, as the signed distance between *Player* and the (left) *Flag1* (on the x axis) is bigger than 15. This agent selects the correct action for the right reasons.

A more striking display of the benefits of the inherent explanations of SCoBot agents is depicted by the Pong agent in Fig 4. The provided explanation suggests that the agent is basing its decision on the vertical positions of the enemy and of its own paddle (distance between the two paddles on the y axis). In fact, this suggests that the agent is largely ignoring the ball's position. Interestingly, upon closer inspection of the enemy movements, we observed that, previously unknown, the enemy is programmed to follow the ball's vertical position (with a small delay). Thus, the vertical positions of these two objects are highly correlated (Pearson's and Spearman's correlations coefficient above 99%, *cf.* App. A.7). Moreover, the ball's rendering contains flickering, explaining why SCoBots base their decision on this potentially more reliable feature, the enemy paddle's vertical position.

To validate these findings further, we perform evaluations on 2 modified Pong environments in which (i) the enemy is invisible, yet playing (*NoEnemy*, *cf.* Fig. 1), and on one environment where the enemy is not moving after returning the ball (*LazyEnemy)*. We reevaluate the deep and SCoBot agents that were trained on the original Pong environment on *NoEnemy* and observe catastrophic performance drops in HN scores (*cf.* Fig. 3) at a level of random policies for both types of agents. Evaluations with different DQN agents lead to similar performance drops (*cf.* App. A.7). These drops are particularly striking as initial importance maps produced by PPO and DQN agents highlight all 3 moving objects (*i.e.* the player, ball, and enemy) as relevant for their decisions (*cf.* Fig.1 top left and App. A.7), aligned with findings on importance maps of Weitkamp et al. [2018]. These maps suggest that the deep agents had based its decisions on *right* reasons. Our novel results on the *NoEnemy* and *LazyEnemy* environments however greatly calls to question the faithfulness and granularity of such post-hoc explainability methods in RL. They highlight the importance of inherently transparent RL models on the level of concepts, as provided via SCoBots, to identify such potential issues. In the following, we will investigate how to mitigate the discovered issues via SCoBot's bottlenecks.

Guiding SCoBots to mitigate RL-specific caveats (Q3).
We here make use of the interactive properties of SCoBots to address several famous RL specific problems: goal misalignment, ill-defined rewards, difficult credit assignment and reward sparsity.

Realigning SCoBots: For mitigating *goal misalignment* issues of the SCoBot agents trained on Pong, we can simply remove (prune) the enemy from the set of considered objects ($\widetilde{\Omega}$). By doing so, the enemy cannot be used for the action selection process. This leads to guided SCoBots that are able to play Pong and its *NoEnemy* version (*cf.* guided SCoBot in Fig 3 (center)). Furthermore, this shows that playing Pong without observing the enemy is achievable, by simply returning vertical shots, difficult for the enemy to catch.

Ill-defined reward: Defining a reward upfront that incentivizes agents to learn an expected behavior is a difficult problem. An example of *ill-defined reward* (*i.e.* a reward that will lead to an unaligned behavior if the agent maximizes the return) is present in Kangaroo. According to the documentation of the game[3], "the mother kangaroo on the bottom floor tries to reach the top floor where her joey is being held captive by some monkeys". However, punching the monkey enemies gives a higher cumulative reward than climbing up to save the baby. RL agents thus tend to learn to kick monkeys on the bottom floor rather than reaching the joey (*cf.* Fig. 4). For revising such agents, we provide an additional reward signal, based on the distance from the mother kangaroo to the joey (detailed in

[3] `www.retrogames.cz/play_195-Atari2600.php`

App. A.8). As can be seen in Fig. 3 (right), this reward allows the guided SCoBots to achieve the originally intended goal by indeed completing the level. In contrast, deep agents and particularly unguided SCoBots achieve relatively high HN scores, but do not complete the levels.

Difficult Credit Assignment Problem: The *difficult credit assignment problem* is the challenge of correctly attributing credit or blame to past actions when the agent's actions have delayed or non-obvious effects on its overall performance. We illustrate this in the context of Skiing, where in standard configurations, agents receive at each step a negative reward that corresponds to the number of seconds elapsed since the last steps (*i.e.* varying between -3 and -7). This reward aims at punishing agents for going down the slope slowly. Additionally, the game keeps track of the number of flag pairs that the agent has correctly traversed (displayed at the top of the screen, *cf.* Fig. 4), and provide a large reward, proportional to this number, at the end of the episode. Associating this reward signal with the number of passed flags is extremely difficult, without prior knowledge on human skiing competitions. In the next evaluations, we provide SCoBots with another reward at each timestep, proportional to the agent's progression to the flags' `position` to incentivize the agent to pass in between them, as well as a signal rewarding for higher speed:

$$R^{\text{exp}} = \sum_{o \in \{Flag_1, Flag_2\}} D(Player, o)^t - D(Player, o)^{t-1} + V(Player).y. \tag{2}$$

These rewards are further detailed in App. A.8. They allow guided SCoBot agents to reach human scores, whereas the deep and unguided SCoBot agents perform worse than random (*cf.* Fig. 3 (left)).

Note that providing additional reward signals, as done above, obviously also allows to mitigate *reward sparsity*, as we illustrate on Pong (*cf.* App. A.8.3 for a detailed explanation).

Overall, our experimental evaluations not only show that SCoBots are on par with deep agents in terms of performances, but that their concept-based and inspectable nature allows to identify and revise several important RL specific caveats. Importantly, in the course of our evaluations, we identified a previously unknown and critical misalignment of existing RL agents on the simple and iconic Pong environment, via the previously mentioned properties of SCoBots.

4 Limitations

In our evaluations, we made use of the integrated object extractor (ω) of OCAtari which provides us with ground truth data. Such extractors can however also be optimized using supervised [Redmon et al., 2016] or self-supervised [Lin et al., 2020, Delfosse et al., 2023c] object detection methods. Furthermore, our set of relational functions, $\mathcal{F}$, is predefined and fixed. To eliminate the reliance on expert users, this set may alternatively be obtained using the supervision from large language models [Wu et al., 2024], via prior task-knowledge integration [Reid et al., 2022], or neural guidance from a pretrained fully deep agent [Delfosse et al., 2023b]. Other environments require additional information extraction processes. *E.g.* in MsPacman an agent must navigate a *maze*. Providing concept representations for such environments is an important step for future work. Finally, during our revision assessments, we presuppose that the user has adequate knowledge and positive intentions.

5 Related Work

The basic idea of **Concept Bottleneck Models (CBMs)** can be found in work as early as Lampert et al. [2009] and Kumar et al. [2009]. A first systematic study of CBMs and their variations was delivered by Koh et al. [2020] and Stammer et al. [2021] which described a CBM as a two-stage model that first computes intermediate concept representations before generating the final task output. Where learning valuable initial object concept representations without strong supervision is still a tricky and open issue for concept-based models [Stammer et al., 2022, Steinmann et al., 2023, Sawada and Nakamura, 2022, Marconato et al., 2022, Lage and Doshi-Velez, 2020], receiving relational concept representations in SCoBots is performed automatically via the function set $\mathcal{F}$. Since the initial works on CBMs they have found utilization in several applications. Antognini and Faltings [2021] *e.g.* apply CBMs to text sentiment classification. However, these works consider single object concept representations and focus on supervised learning.

In fact, CBMs have found their way into RL only to a limited extent. Zabounidis et al. [2023] and Grupen et al. [2022] utilized CBMs in a multi-agent setting, where both identify improved interpretability through concept representations while maintaining competitive performance. Zabounidis et al. [2023] further report better training stability and reduced sample complexity. Their extension includes an optional "residual" layer which passes additional, latent information to the action selector part. SCoBots omit such a residual component for the sake of human-understandability, yet offer the flexibility to the user to modify the concept layer via updating $\mathcal{F}$ for improving the model's representations. Similar to how SCoBots invite the user to reuse the high-level concepts to shape the reward signal, Guan et al. [2023] allow the user to design a reward function based on higher-level properties that occur over a period of time. Lastly, SCoBots not only separate state representation learning from policy search, as done by Cuccu et al. [2020], but also enforce object-centricity, putting interpretability requirements on the consecutive feature spaces.

Explainable RL (XRL) is an extensively surveyed XAI subfield [Milani et al., 2023, Dazeley et al., 2022, Krajna et al., 2022] with a wide range of yet unsolved issues [Vouros, 2022]. Milani et al. [2023] introduce a taxonomy for XRL methods, with: (1) feature importance methods that generate explanations that point out decision-relevant input features, (2) learning process & MDP methods which present which past experience or MDP components affect the policy, and (3) policy-level methods, describing the agent's long-term behavior. Based on this, SCoBots extract relations from low-level features making high-level information available to explanations and thereby support feature importance methods. According to Qing et al. [2022]'s categorization of XRL frameworks, SCoBots are "model-explaining" (in contrast to reward-, state-, and task-explaining). Other XRL methods rely on decision trees [Fuhrer et al., 2024, Marton et al., 2024], logic [Jiang and Luo, 2019, Kimura et al., 2021, Delfosse et al., 2023b, Sha et al., 2024] or programs [Verma et al., 2018, Trivedi et al., 2021, Cao et al., 2022, Kohler et al., 2024, Wüst et al., 2024] to encode transparent policies.

The **misalignment problem** is an RL instantiation of the shortcut learning problem, a frequently studied failure mode that has been identified in models and datasets across the spectrum of AI from deep networks [Lapuschkin et al., 2019, Schramowski et al., 2020] to neuro-symbolic [Stammer et al., 2021, Marconato et al., 2023], prototype-based models [Bontempelli et al., 2023] and RL approaches [di Langosco et al., 2022]. A misaligned RL agent, first empirically studied by [Koch et al., 2021] represents a serious issue, especially when it is misaligned to recognized ethical values [Arnold and Kasenberg, 2017] or if the agent has broad capabilities [Ngo, 2022]. Nahian et al. [2021] ethically align agents by introducing a second reward signal. In comparison, SCoBots aid to resolve RL specific issues such as the misalignment problem through inherent interpretability.

6 Conclusion

In this work, we have provided evidence for the benefits of concept-based models in RL tasks, specifically for identifying issues such as goal misalignment. Among other things our proposed Successive Concept Bottleneck agents integrate relational concepts into their decision processes. With this, we have exposed previously unknown misalignment problems of deep RL agents in a game as simple as Pong. SCoBot agents allowed us to revise this issue, as well as different RL specific caveats with minimal additional feedback. Our work thus represents an important step in developing *aligned* RL agents, *i.e.* agents that are not just aligned with the underlying task goals, but also with human user's understanding and knowledge. Achieving this is particularly valuable for applying RL agents in real-world settings where ethical and safety considerations are paramount.

Avenues for future research are incorporating a high level action bottleneck [Bacon et al., 2017]. One can also incorporate attention mechanisms into RL agents, as discussed in [Itaya et al., 2021], or use language models (and *e.g.*, the game manuals) to generate the additional reward signal, as done by Wu et al. [2024]. Additionally, we are considering the use of shallower decision trees [Broelemann and Kasneci, 2019]. An interesting research question is how far the task reward signal can aid in learning games object-centric representations [Delfosse et al., 2023c] in the first place.

Impact statement

Our work aims at developing transparent RL agents, whose decision can be understood and revised to be aligned with the beliefs and values of a human user. We believe that such algorithms are critical to

uncover and mitigate potential misalignments of AI systems. A malicious user can, however, utilize such approaches for aligning agents in a harmful way, thereby potentially leading to a negative impact on further users or society as a whole. Even so, the inspectable nature of transparent approaches will allow to identify such potentially harmful misuses, or hidden misalignment.

References

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 2017.

Diego Antognini and Boi Faltings. Rationalization through concepts. *ArXiv*, 2021.

Thomas Arnold and Daniel Kasenberg. Value alignment or misalignment – what will keep systems accountable? In *AAAI Workshop on AI, Ethics, and Society*, 2017.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, 2017.

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems, NeurIPS*, 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.

Andrea Bontempelli, Stefano Teso, Katya Tentori, Fausto Giunchiglia, and Andrea Passerini. Concept-level debugging of part-prototype networks. In *International Conference on Learning Representations (ICLR)*, 2023.

Klaus Broelemann and Gjergji Kasneci. A gradient-based split criterion for highly accurate and transparent model trees. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 2019.

Yushi Cao, Zhiming Li, Tianpei Yang, Hao Zhang, Yan Zheng, Yi Li, Jianye Hao, and Yang Liu. Galois: boosting deep reinforcement learning via generalizable logic synthesis. *Advances in Neural Information Processing Systems*, 2022.

Chun Sik Chan, Huanqi Kong, and Guanqing Liang. A comparative study of faithfulness metrics for model interpretability methods. In *Conference of the Association for Computational Linguistics (ACL)*, 2022.

Karl Cobbe, Oleg Klimov, Christopher Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, 2019.

Giuseppe Cuccu, Julian Togelius, and Philippe Cudré-Mauroux. Playing atari with six neurons (extended abstract). In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, 2020.

Richard Dazeley, Peter Vamplew, and Francisco Cruz. Explainable reinforcement learning for broad-xai: a conceptual framework and survey. *Neural Computing and Applications*, 2022.

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. In *Advances in Neural Information Processing (NeurIPS)*, 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, Patrick Schramowski, Martin Mundt, Alejandro Molina, and Kristian Kersting. Adaptive rational activations to boost deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2024.

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.

Benjamin Fuhrer, Chen Tessler, and Gal Dalal. Gradient boosting reinforcement learning. *arXiv*, 2024.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard S. Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2020.

Niko Grupen, Natasha Jaques, Been Kim, and Shayegan Omidshafiei. Concept-based understanding of emergent multi-agent behavior. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

Lin Guan, Karthik Valmeekam, and Subbarao Kambhampati. Relative behavioral attributes: Filling the gap between symbolic goal specification and reward learning from human preferences. In *International Conference on Learning Representations (ICLR)*, 2023.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *AAAI Conference on Artificial Intelligence*, 2017.

Geoffrey Irving, Paul Francis Christiano, and Dario Amodei. Ai safety via debate. *ArXiv*, 2018.

Hidenori Itaya, Tsubasa Hirakawa, Takayoshi Yamashita, Hironobu Fujiyoshi, and Komei Sugiura. Visual explanation using attention mechanism in actor-critic-based deep reinforcement learning. *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021.

A. Jabri, Kyle Hsu, Benjamin Eysenbach, Abhishek Gupta, Alexei A. Efros, Sergey Levine, and Chelsea Finn. Unsupervised curricula for visual meta-reinforcement learning. *ArXiv*, 2019.

Zhengyao Jiang and Shan Luo. Neural logic reinforcement learning. In *International conference on machine learning*, 2019.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, K. Czechowski, D. Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, G. Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. *ArXiv*, 2019.

Subbarao Kambhampati, Sarath Sreedharan, Mudit Verma, Yantian Zha, and L. Guan. Symbols as a lingua franca for bridging human-ai chasm for explainable and advisable ai systems. In *AAAI Conference on Artificial Intelligence*, 2021.

Daiki Kimura, Masaki Ono, Subhajit Chaudhury, Ryosuke Kohita, Akifumi Wachi, Don Joven Agravante, Michiaki Tatsubori, Asim Munawar, and Alexander Gray. Neuro-symbolic reinforcement learning with first-order logic. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

Jack Koch, Lauro Langosco, Jacob Pfau, James Le, and Lee Sharkey. Objective robustness in deep reinforcement learning, 2021.

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, 2020.

Hector Kohler, Quentin Delfosse, Riad Akrour, Kristian Kersting, and Philippe Preux. Interpretable and editable programmatic tree policies for reinforcement learning. In *Workshop on Interpretable Policies in Reinforcement Learning@ RLC-2024*, 2024.

Agneza Krajna, Mario Brčič, Tomislav Lipić, and Juraj Doncevic. Explainability in reinforcement learning: perspective and position. *ArXiv*, 2022.

Neeraj Kumar, Alexander C. Berg, Peter N. Belhumeur, and Shree K. Nayar. Attribute and simile classifiers for face verification. *2009 IEEE 12th International Conference on Computer Vision*, 2009.

Isaac Lage and Finale Doshi-Velez. Learning interpretable concept-based models with human feedback. *ArXiv*, 2020.

Christoph H. Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Unmasking clever hans predictors and assessing what machines really learn. *Nature communications*, 2019.

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.

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.

Emanuele Marconato, Andrea Passerini, and Stefano Teso. Glancenets: Interpretable, leak-proof concept-based models. In *Advances in Neural Information Processing (NeurIPS)*, 2022.

Emanuele Marconato, Stefano Teso, and Andrea Passerini. Neuro-symbolic reasoning shortcuts: Mitigation strategies and their limitations. In *International Workshop on Neural-Symbolic Learning and Reasoning*, 2023.

Sascha Marton, Tim Grams, Florian Vogt, Stefan Lüdtke, Christian Bartelt, and Heiner Stuckenschmidt. Sympol: Symbolic tree-based on-policy reinforcement learning. *arXiv*, 2024.

Thomas Mesnard, Theophane Weber, Fabio Viola, Shantanu Thakoor, Alaa Saade, Anna Harutyunyan, Will Dabney, Thomas S. Stepleton, Nicolas Heess, Arthur Guez, Eric Moulines, Marcus Hutter, Lars Buesing, and Rémi Munos. Counterfactual credit assignment in model-free reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

Stephanie Milani, Nicholay Topin, Manuela Veloso, and Fei Fang. Explainable reinforcement learning: A survey and comparative review. *ACM Computing Surveys*, 2023.

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.

Md Sultan Al Nahian, Spencer Frazier, Brent Harrison, and Mark O. Riedl. Training value-aligned reinforcement learning agents using a normative prior. *ArXiv*, 2021.

A. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, 1999.

Richard Ngo. The alignment problem from a deep learning perspective. *ArXiv*, 2022.

Yunpeng Qing, Shunyu Liu, Jie Song, and Mingli Song. A survey on explainable reinforcement learning: Concepts, algorithms, challenges. *ArXiv*, 2022.

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.

David Raposo, Samuel Ritter, Adam Santoro, Greg Wayne, Théophane Weber, Matthew M. Botvinick, H. V. Hasselt, and Francis Song. Synthetic returns for long-term credit assignment. *ArXiv*, 2021.

Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Conference on Computer Vision and Pattern Recognition, CVPR 2016*, 2016.

Machel Reid, Yutaro Yamada, and Shixiang Shane Gu. Can wikipedia help offline reinforcement learning? *ArXiv*, 2022.

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.

Yoshihide Sawada and Keigo Nakamura. Concept bottleneck model with additional unsupervised concepts. *IEEE Access*, 2022.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR*, 2016.

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

Jingyuan Sha, Hikaru Shindo, Quentin Delfosse, Kristian Kersting, and Devendra Singh Dhami. Expil: Explanatory predicate invention for learning in games. *arXiv preprint*, 2024.

A. Srinivas, Michael Laskin, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *ArXiv*, 2020.

Wolfgang Stammer, Patrick Schramowski, and Kristian Kersting. Right for the right concept: Revising neuro-symbolic concepts by interacting with their explanations. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2021.

Wolfgang Stammer, Marius Memmel, Patrick Schramowski, and Kristian Kersting. Interactive disentanglement: Learning concepts by interacting with their prototype representations. In *Conference on Computer Vision and Pattern Recognition, (CVPR)*, 2022.

David Steinmann, Wolfgang Stammer, Felix Friedrich, and Kristian Kersting. Learning to intervene on concept bottlenecks. *ArXiv*, 2023.

Stefano Teso, Öznur Alkan, Wolfgang Stammer, and Elizabeth Daly. Leveraging explanations in interactive machine learning: An overview. *Frontiers in Artificial Intelligence*, 2023.

Claude F. Touzet. Neural reinforcement learning for behaviour synthesis. *Robotics Auton. Syst.*, 1997.

Dweep Trivedi, Jesse Zhang, Shao-Hua Sun, and Joseph J Lim. Learning to synthesize programs as interpretable and generalizable policies. *Advances in neural information processing systems*, 2021.

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.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2018.

George A. Vouros. Explainable deep reinforcement learning: State of the art and challenges. *ACM Computing Surveys*, 2022.

Laurens Weitkamp, Elise van der Pol, and Zeynep Akata. Visual rationalizations in deep reinforcement learning for atari games. In *BNCAI*, 2018.

Yue Wu, Yewen Fan, Paul Pu Liang, Amos Azaria, Yuanzhi Li, and Tom M Mitchell. Read and reap the rewards: Learning to play atari with the help of instruction manuals. *Advances in Neural Information Processing Systems*, 2024.

Antonia Wüst, Wolfgang Stammer, Quentin Delfosse, Devendra Singh Dhami, and Kristian Kersting. Pix2code: Learning to compose neural visual concepts as programs. In *The 40th Conference on Uncertainty in Artificial Intelligence*, 2024.

Renos Zabounidis, Joseph Campbell, Simon Stepputtis, Dana Hughes, and Katia P. Sycara. Concept learning for interpretable multi-agent reinforcement learning. *ArXiv*, 2023.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 2021.

Xiaomao Zhou, Tao Bai, Yanbin Gao, and Yuntao Han. Vision-based robot navigation through combining unsupervised learning and hierarchical reinforcement learning. *Sensors (Basel, Switzerland)*, 2019.

Çağlar Aytekin. Neural networks are decision trees. *ArXiv*, 2022.

A Appendix

As mentioned in the main body, the appendix contains additional materials and supporting information for the following aspects: further information on the fact that Atari is the most common set of games (A.1), details on the reward sparsity in Pong (A.8.3), detailed numerical results (A.2), learning curves of our RL agents (A.3), the hyperparmeters used in this work (A.5), formal definitions on the SCoBots policies (A.6) and further details on the misalignment problem in Pong.

A.1 Atari games are most common set of environments

We here show that the Atari games from the Arcade Learning Environment [Bellemare et al., 2012] is the most used benchmark to test reinforcement learning agents.

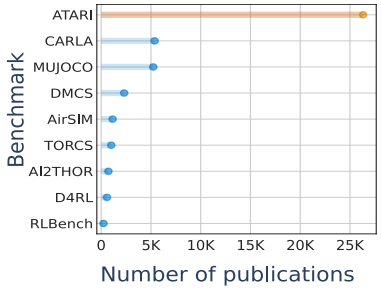

Figure 5: **The Atari Learning Environments is more used in scientific research than the next** 8 **other benchmarks together**. Graph borrowed from [Delfosse et al., 2023a].

A.2 Detailed numerical results

For completeness, we here provide the numerical results of the performances obtained by each agent type. For fair comparison, we reimplemeneted the deep PPO agents, and used the default hyperparameters for Atari from stable-baselines3 for the deep RL agents. A detailed overview of hyperparameters and their corresponding values can be found in A.5.

All agents are trained on gymnasium's atari learning environments using 8 actors and 3 training seeds ($[0, 16, 32]$+rank). Each training seed's performance is evaluated every 500k frames on 4 differently seeded (42+training seed) environments for 8 episodes each. After training, the best performing checkpoint is then ultimately evaluated on 4 seeded $(123, 456, 789, 1011)$ test environments. The final return is determined by averaging the return over 5 episodes per training-test seed pair and every training seed for the respective environment. We use deterministic actions for both evaluation and testing stages.

Game	**SCoBots** v5	**SCoBots** guided-v5	**PPO** ours-v5	**PPO** ours-v4	**PPO** Schulman et al.-v4	**Random**	**Human**
Asterix	$5080_{\pm 614.8}$	$5043.3_{\pm 729.9}$	$2126.7_{\pm 148.6}$	$10433.3_{\pm 2772.9}$	4532	210	8503
Bowling	$106.6_{\pm 41.8}$	$137.4_{\pm 24.1}$	$102.2_{\pm 39.1}$	$51.4_{\pm 18.0}$	40.1	23.1	160.7
Boxing	$97.1_{\pm 2.2}$	$69.0_{\pm 10.9}$	$90.3_{\pm 3.0}$	$99.5_{\pm 1.4}$	94.6	0.10	4.3
Freeway	$32.8_{\pm 0.1}$	$32.9_{\pm 0.7}$	$33.6_{\pm 0.2}$	$33.6_{\pm 0.3}$	32.5	0.00	29.6
Kangaroo	$2776.6_{\pm 1332.4}$	$4050.0_{\pm 217.8}$	$790.0_{\pm 280.8}$	$13296.7_{\pm 1111.1}$	9929	52.0	3035
Pong	$17.2_{\pm 1.9}$	$17.5_{\pm 1.8}$	$16.4_{\pm 1.5}$	$20.9_{\pm 0.2}$	20.7	-20.7	14.6
Seaquest	$1055.3_{\pm 272.6}$	$2411.3_{\pm 377.0}$	$837.3_{\pm 46.7}$	$1262.0_{\pm 446.1}$	1204.5	68.4	20182
Skiing	$-23004_{\pm 10333}$	$-6530_{\pm 3326}$	$-23004_{\pm 10333}$	$-22983_{\pm 10333}$	-13901	-17098	-4336
Tennis	$2.4_{\pm 3.6}$	$0.0_{\pm 0.0}$	$-0.9_{\pm 0.1}$	$-1.2_{\pm 0.3}$	-14.8	-23.8	-8.3

Table 1: ScoBots and guided ScoBots obtain similar results than deep agents averaged over 3 seeded runs. Neural refers to the agent that use a CNN instead of our ICB layers. We added the results reported in the original paper Schulman et al. [2017] (original), which are on par with ours, as well as ones from a Random baseline and Humans (from van Hasselt et al. [2016]). Our agents have been trained with only 20M frames.

Normalisation techniques. To compute human normalised scores, we used the following equation:

$$\text{score}_{\text{normalised}} = 100 \times \frac{\text{score}_{\text{ agent}} - \text{score}_{\text{ random}}}{\text{score}_{\text{ human}} - \text{score}_{\text{ random}}}. \tag{3}$$

A.3 Learning curves

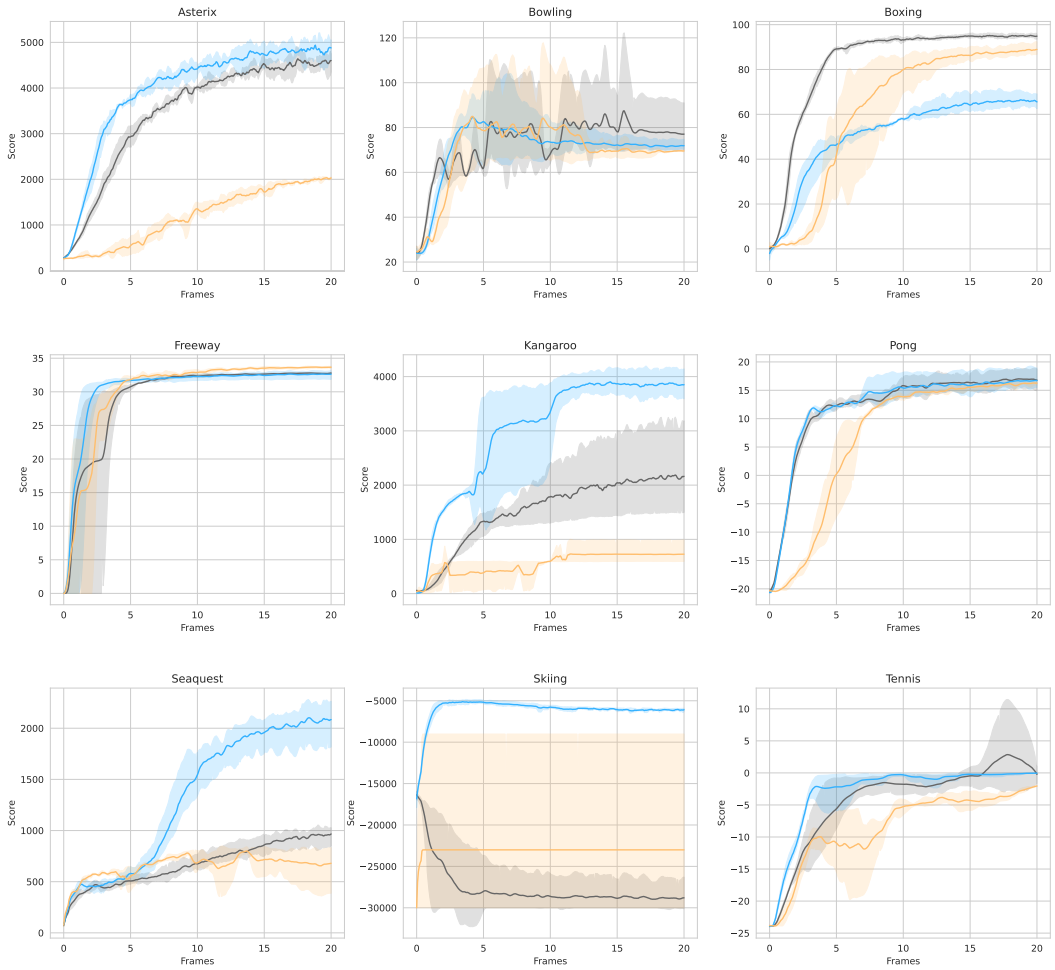

Figure 6: Training Return over frames ($\times 10^6$) seen for ■ guided SCoBots, ■ SCoBots and ■ neural agents.

A.4 Agent reasoning through the decision trees

In this section, we provide more decision trees from SCoBots on states from Freeway and Bowling.

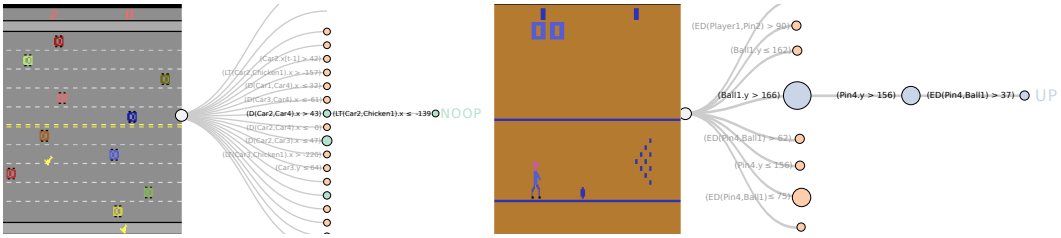

Figure 7: Decision trees from Freeway and Bowling.

A.5 Hyperparameters and experimental details

All Experiments were run on a AMD Ryzen 7 processor, 64GB of RAM and one NVIDIA GeForce RTX 2080 Ti GPU. Training a (i)SCoBots on 20M frames and using 8 actors takes approximately 4 hours. We use the same PPO hyperparameters as the Schulman et al. [2017] agents that learned to master the games. For the Adam optimizer, SCoBots start with a slightly increased learning rate of 1×10^{-3} (compared to 2.5×10^{-4}). The PPO implementation used and the respective MLP hyperparameters are based on stable-baselines3 Raffin et al. [2021]. SCoBots have the same PPO hyperparameter values as deep agents but use MLPs (2×64) with ReLU activation functions as policy and value networks. Deep agents use `CnnPolicy` in stable-baselines3 as their policy value network architecture, which aligns with the initial baseline implementation of PPO. Further, we normalize the advantages in our experiments, since it showed only beneficial effects on learning. This is the default setting in the stable-baseline3 implementation.

Actors N	8
Minibatch size	$32 * 8$
Horizon T	2048
Num. epochs K	3
Adam stepsize	$2.5 * 10^{-4} * \alpha$
Discount γ	0.99
GAE parameter λ	0.95
Clipping parameter ϵ	$0.1 * \alpha$
VF coefficient c_1	1
Entropy coefficient c_2	0.01

Table 2: PPO Hyperparameter Values. α linearly decreases from 1 to 0 over the course of training.

The Atari environment version used in gymnasium is `NoFrameskip-v4` for agents reproducing the reported PPO results, and `v5` for SCoBots and neural. `v5` defines a deterministic skipping of 5 frames per action taken and sets the probability to repeat the last action taken to 0.25. This is aligned with recommended best practices by Machado et al. [2018]. The experiments using `NoFrameskip-v4` utilize the same environment wrappers as OpenAI's initial baselines implementation[4]. This includes frame skipping of 4 and reward clipping. Frame stacking is not used. SCoBots are not trained on a clipped reward signal. For comparability, the neural agents we compare SCoBots to, are not as well. A list of all hyperparameter values used is provided in Table 2.

A.5.1 The properties and features used for SCoBots

In this paper, we used different properties and functions (to create features). They are listed hereafter. Further, Table 4 shows a detailed overview of the concepts and actions pruned for every environment evaluated.

A.6 SCoBots policies: formal definitions

The set $\mathcal{S}$ denotes the problem-specific set of raw environment states as defined by the RL problem, *e.g.*, the space of RGB images $[0, 255]^{512 \times 512 \times 3}$. Similarly, $\mathcal{A}$ represents the action space, *e.g.*, "move right" or "jump."

[4] `github.com/openai/baselines`

	Name	Definition	Description
Properties	class	`NAME`	object class (*e.g.* "Agent", "Ball", "Ghost")
	position	x, y	position on the screen
	position history	$x_t, y_t, x_{t-1}, y_{t-1}$	position and past position on the screen
	orientation	o	object's orientation if available
	RGB	R, G, B	RGB values
Relations	distance	$D_x(o_1, o_2), D_y(o_1, o_2)$	distance the x and y axis
	euclidean distance	$D_e(o_1, o_2)$	euclidean distance
	trajectory	$landing\_point_x(o_1, o_2)$	orthogonal projection of o_1 onto o_2 trajectory
	center	$center(o_1, o_2)$	center of the shortest line between o_1 and o_2
	speed	$s(x_t, y_t, x_{t-1}, y_{t-1})$	speed of object (pixel per step)
	velocity	$v(x_t, y_t, x_{t-1}, y_{t-1})$	velocity of object (vector)
	color	$color(o_1)$	the CSS2.1[5] color category (*e.g.* Red)
	top k	$Top_k((o_1, ..., o_n), k, class)$	only k closest (D_e to player) objects visible

Table 3: Descriptions of properties and relations used by SCoBots.

	Features	Bowling	Boxing	Pong	Freeway	Tennis	Skiing	Kangaroo	Asterix	Seaquest
Properties	class	✓	✓	no enemy	✓	✓	✓	✓	✓	✓
	position	✓	✓	✓	✓	✓	✓	✓	✓	✓
	position history	✓	X	✓	✓	✓	✓	✓	✓	✓
	orientation	X	X	X	X	X	✓	X	X	✓
	RGB	X	X	X	X	X	X	X	X	X
Relations	trajectory	X	X	X	X	✓	X	X	X	X
	distance	✓	✓	✓	✓	✓	✓	✓	✓	X
	euclidean distance	X	X	X	X	X	X	X	X	X
	center	X	X	X	X	X	✓	X	X	X
	speed	X	X	X	✓	✓	X	X	X	X
	velocity	X	X	✓	X	X	✓	✓	✓	X
	color	X	X	X	X	X	X	X	X	X
	k closest objects	4	X	X	4	X	2	2	X	X
Actions	NOOP	✓	✓	✓	✓	✓	✓	✓	✓	✓
	FIRE	✓	✓	✓	-	✓	-	✓	-	✓
	UP	✓	✓	-	✓	✓	-	✓	✓	✓
	RIGHT	-	✓	✓	-	✓	✓	✓	✓	✓
	LEFT	-	✓	✓	-	✓	✓	✓	✓	✓
	DOWN	✓	✓	-	✓	✓	-	✓	✓	✓
	UPRIGHT	-	✓	-	-	✓	-	✓	✓	-
	UPLEFT	-	✓	-	-	✓	-	✓	✓	-
	DOWNRIGHT	-	✓	-	-	✓	-	X	✓	-
	DOWNLEFT	-	✓	-	-	✓	-	X	✓	-
	UPFIRE	✓	✓	-	-	✓	-	X	-	-
	RIGHTFIRE	-	✓	X	-	✓	-	X	-	-
	LEFTFIRE	-	✓	X	-	✓	-	X	-	-
	DOWNFIRE	✓	✓	-	-	✓	-	X	-	-
	UPRIGHTFIRE	-	✓	-	-	X	-	X	-	-
	UPLEFTFIRE	-	✓	-	-	X	-	X	-	-
	DOWNRIGHTFIRE	-	✓	-	-	X	-	X	-	-
	DOWNLEFTFIRE	-	✓	-	-	X	-	X	-	-

Table 4: Feature selection and pruning for guided SCoBots. ✓ denotes the included features, whereas X the features that are pruned out.

Let O_0 be the set of all possible objects, identified by their ID and characterized by their properties like position, size, color, etc. Then, $\mathcal{O} := \{O \in \mathcal{P}(O_0) \mid \text{id}(o_1) \neq \text{id}(o_2) \forall o_1, o_2 \in O\}$ is the *object*

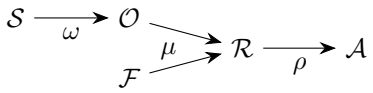

Figure 8: Functional summary of the SCoBots model architecture: ω maps states to sets of objects, μ maps sets of objects to relation vectors by applying relational functions, and ρ maps relation vectors to actions.

detection space, defined as the family of object sets in which each object (identified by its ID) occurs once at most.

$\mathcal{F}$ is the *relation function space*. It is the family of user-defined relation function sets F. Each relation function $f \in F$ is of the form $f : O^k \to \mathbb{R}^n$ for $k, n \in \mathbb{N}$, that is, it maps a fixed number of objects to a real-valued vector. As an instance, the function that maps two objects to the Euclidean distance between each other is a relation function.

The set $\mathcal{R} \subseteq \mathbb{R}^m$ is the *relation* (or *feature*) *space*. The dimension m is implied by $\mathcal{F}$. More specifically, $m = \sum_{f \in \mathcal{F}} \dim(\text{co}(f))$, *i.e.*, m is the sum of each relation function's codomain dimension.

Definition A.1. The overall model policy $\pi : \mathcal{S} \to \mathcal{A}$ given relation function set F is defined as $\pi := \rho \circ \mu(\cdot, F) \circ \omega$, where

1. $\omega : \mathcal{S} \to \mathcal{O}$ is the *object detector*, defined as the function that maps a raw state $s \in \mathcal{S}$ to a set of detected objects $O \in \mathcal{O}$.
2. μ is the *feature selector*

$$\mu : \mathcal{O} \times \mathcal{F} \to \mathcal{R} \tag{4}$$

$$(O, F) \mapsto \mathbf{r} := \left(f(o_1, ..., o_k)\right)_{f \in F}, \text{where } o_1, ..., o_k \in O. \tag{5}$$

   That is, μ applies each relation function $f \in \mathcal{F}$ to the respective detected object(s) in $O \in \mathcal{O}$, resulting in a real-valued relation vector.

3. $\rho : \mathcal{R} \to \mathcal{A}$ is the *action selector* that assigns actions to relation vectors.

See also Figure 8 for a summary. The policy parameters $\theta := (\theta_1, \theta_2, \theta_3)$ split up into the parameters of the object detector, the feature selector, and the action selector, accordingly. They are left out for brevity.

The architecture's bottleneck is induced by the object detector ω and the feature selector μ. From this function perspective, the bottleneck consists of two stages: an object-centric bottleneck stage at $\mathcal{O}$ and a *semantic* bottleneck stage at $\mathcal{R}$.

A.7 Pong misalignment problem

The misalignment problem had previously been identified in other games such as the Coinrun platform game [Cobbe et al., 2019], in which an agent's target-goal is to reach a coin which is always placed at the end of a level at training time. When the coin is repositioned at test time di Langosco et al. observed that trained deep RL agents avoid the coin and simply target the end of the level, indicating that agents in fact learn to follow a simpler side-goal rather than the underlying strategy.

For the correlation between the enemy's and the ball's y positions, we let an random agent play the game for 100000 frames and collect the positions of the enemy and ball every 10 frames. We then compute different correlation coefficients, and obtain **99.6%** as Pearson coefficient, **96.4%** as Kendall coefficient and **99.5%** as Spearman coefficient.

We also collected importance maps of deep DQN RL agents playing Pong using ReLU and Rational activation functions (borrowed from Delfosse et al. [2024].

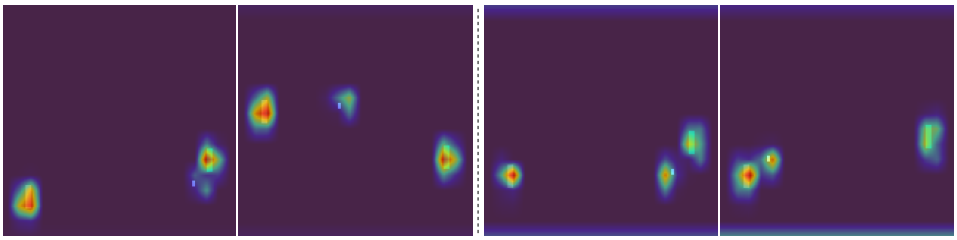

Figure 9: More importance maps of DQN agents playing Pong, with ReLU (left) and rational activation functions (right).

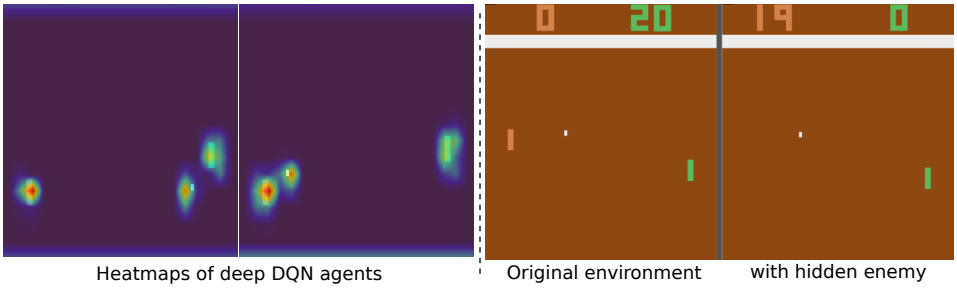

Figure 10: Importance maps of trained deep DQN agents playing Pong (left) show that all 3 moving objects are important for the agent's decision, suggesting that the agent takes aims at sending the ball past the enemy. The same trained agent completely outperform its enemy in the original environment while it cannot return the ball in a modified version of the game where the enemy is hidden.

A.8 RL specific caveats

In this section, we give more details on the different RL specific caveats, as well

A.8.1 Ill-defined Objectives

Defining an accurate reward signal when creating an RL task is challenging, as it requires the designer to specify a precise and balanced reward function that effectively guides the agent towards the desired behavior while avoiding unintended consequences [Henderson et al., 2017, Irving et al., 2018]. Shaping the reward after having observed (and understood) the non-intended behavior of the agent is very common [Andrychowicz et al., 2017].

To realign our SCoBots agent, we provide them with a reward signal proportional to the progression to the joey:

```
player = _get_game_objects_by_category(game_objects, ["Player"])

# Get current platform
platform = np.ceil((player.xy[1] - player.h - 16) / 48)  # 0: topmost, 3: lowest platform

# Encourage moving to the child
if not episode_starts and not last_crashed:
    if platform % 2 == 0:  # even platform, encourage left movement
        reward = - player.dx
    else:  # encourage right movement
        reward = player.dx

    # Encourage upward movement
    reward -= player.dy / 5
else:
    reward = 0
```

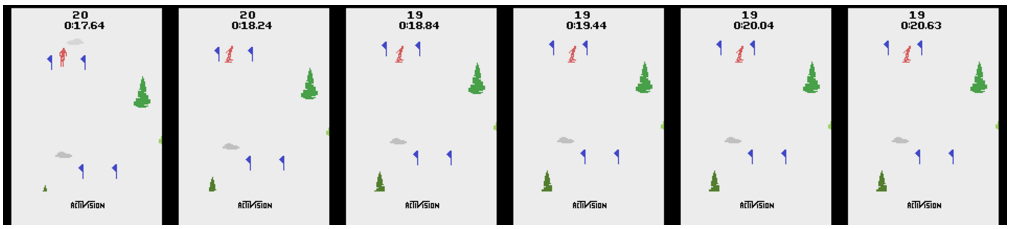

Figure 11: iSCoBots learn to land in between the Flag to maximize its reward, when only R_1^{exp} is provided.

A.8.2 Difficult Credit Assignment

When an outcome is delayed or uncertain, it is challenging to identify the action that is relevant for that outcome [Mesnard et al., 2021]. This problem arises in games like chess, where reward is passed to the agent only when the game is over. The problem can be addressed by more instant informative feedback, given directly after successful actions or after a positive state is reached.

In the main part of this manuscript, we address this problem on the game Skiing. To encourage the agent to go in between the flag, we add a reward that corresponds to the distance to both of the next flags (on the x axis):

$$R_1^{\text{exp}} = D(Player, Flag1).x + D(Player, Flag2).x. \tag{6}$$

This resulted in an agent that learned to stop itself in between the flags (depicted in Fig. 11), showing once again, how difficult it is to create a reward that would favor a specific behavior. To correct it, we therefore adjoined another signal to reward our SCoBot agents proportionally to the **speed** of the agent:

$$R_2^{\text{exp}} = V(Player).y. \tag{7}$$

A.8.3 The reward sparsity of Pong

Let us move on to the issue of *sparse reward* in the context of RL. In Pong, to score a point, the player needs to return the ball and have it go past the enemy. To do so, the player has to perform a spiky shot, obtained by touching the ball on one of its paddle's sides (otherwise, the shot is flat and the enemy is easily catching it). Its vertical position may vary between 34 and 190. Its paddle height is 16, but it needs to shoot from the side of the paddle (the 3 pixel of each border) to get a spiky shot that is likely to go beyond the opponent. If the balls arrives not close to the top or bottom borders, the enemy will still catch it, which has the probability of $\sim 22\%$ in our experiments. The probability of getting a successful shot is thus of $\sim 6/156 * 0.78 = 3\%$. The average number of corresponding tryouts for the agent to get rewarded is $\sum_{n=1}^{\infty} \left(n \times 0.03 \times 0.97^{n-1}\right) = 33.3$. The agent usually need 60 steps (in average) from the initialization of the point to the reward attribution, hence, it will need ~ 2200 steps to be rewarded. This is consistent with the experiment depicted in Figure 12. In this figure, a random agent (original) needs in average 2230 steps to observe reward.

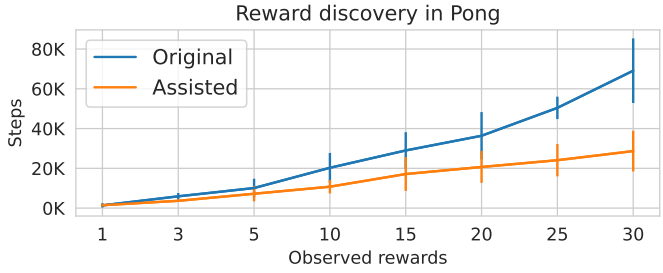

Figure 12: Expert user feedback allow for faster discovering of the reward signal in the sparse reward Pong environment.

Providing a SCoBot agent with an additional reward that is inversely proportional to the distance between its paddle and the ball incentivizes the agent to keep a vertical position close to the ball's

one:

$$R^{\text{exp}} = D(Player, Ball).y \tag{8}$$

This extra reward signal lets a starting agent a winning shot every ~ 820, multiplying by 2.7 their occurrences. Thus, the reward signal in Pong is relatively sparse, but our example illustrates how the interpretable concepts allow to easily guide SCoBots, making up for the reward sparsity. Interestingly, this also giving a clearer incentive to the agent to concentrate on the ball's position, and not on the enemy's one. We observe that such agent do not rely on the enemy to master the game. Alternative techniques such as prioritized experience replay [Schaul et al., 2016] allow offline methods to learn in such environments, but providing a smoother reward signal is another elegant way to address sparsity. Note that sparser environments exists (such as robotics ones).

A.8.4 Misalignment

In our experimental evaluations, we used the concept based reward shaping to realign the agents on the true (target) task objectives, preventing undesirable behaviors resulting from misaligned optimization. Shortcut learning can be mitigated through the implementation of such reward signals that necessitate the genuine objective's consideration.