

---

# Interpretable Concept Bottlenecks to Align Reinforcement Learning Agents

---

Quentin Delfosse<sup>\*1</sup> Sebastian Sztwiertnia<sup>\*.1</sup> Mark Rothermel<sup>1</sup>  
Wolfgang Stammer<sup>1,2</sup> Kristian Kersting<sup>1,2,3,4</sup>

<sup>1</sup>Computer Science Department, TU Darmstadt, Germany

<sup>2</sup>Hessian Center for Artificial Intelligence (hessian.AI), Darmstadt, Germany

<sup>3</sup>Centre for Cognitive Science, TU Darmstadt, Germany

<sup>4</sup>German Research Center for Artificial Intelligence (DFKI), Darmstadt, Germany  
{delfosse, stammer, kersting}@cs.tu-darmstadt.de

## 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<sup>2</sup> 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]. 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 target goal is to catch a ball with a 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 paddle, rather than the position of the ball itself. This phenomenon is generally described by the umbrella term *shortcut learning* [Geirhos et al., 2020], in which decision rules are learned on a 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 [Advani and Saxe, 2017, 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 [Lapuschkin et al., 2019, Guidotti et al., 2019, Schramowski et al., 2020, Ras et al., 2022, Roy et al., 2022, Saeed and Omlin, 2023]. However, many XAI methods’ explanation do not faithfully present the model’s underlying decision process [Hooker et al., 2019, Chan et al., 2022, DeYoung et al., 2020]. 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

---

<sup>\*</sup>Equal contribution

<sup>2</sup>Code available at <https://github.com/k4ntz/SCoBots>

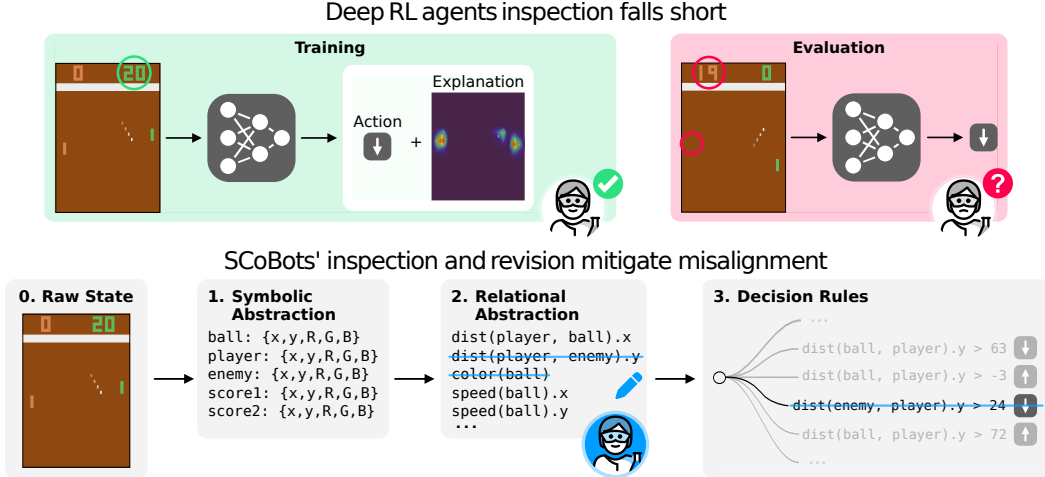


Figure 1: **Successive Concept Bottleneck Agents (SCoBots) allow for easy inspection and revision.** Top: Deep RL agents trained on Pong produce high playing scores with importance map explanations that suggest sensible underlying reasons for taking an action (✓). However, when the enemy is hidden, the 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.

focuses on models that provide inherent concept-based explanations. Prominent examples are concept bottleneck 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].

Contrary to existing CBMs’ fields of applications, RL requires relational reasoning [Kaiser et al., 2019]. We therefore introduce Successive Concept Bottleneck Agents (SCoBots, *cf.* Fig. 1, bottom) bringing concept bottleneck approach to RL. SCoBots integrate successive concept bottlenecks into their decision processes, where each bottleneck layer provides representations that integrate the concepts of the previous bottleneck layer. Specifically, provided a set of predefined concept functions, SCoBots automatically extract relational concepts based on objects and their properties of the first bottleneck layers. Finally, the set of relational and object concept representations are used for action selection. SCoBots thus are inherently interpretable RL agents and contrary 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, 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 justifications for the action selections and the potential of mitigating a variety of RL specific issues, from reward sparsity to misalignment problems, via simple guidance. In the course of our evaluations, we identify previously an unknown shortcut behavior of deep RL agents on *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 inspecting 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 first formally introduce SCoBots, its components and discuss its guidance mechanism. We continue with experimental evaluations, via several metrics and analysis. Before concluding, we touch upon related work.

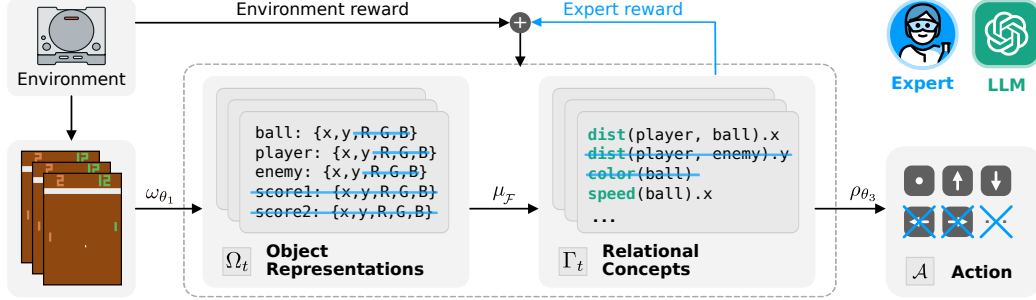


Figure 2: **An overview of Successive Concept Bottleneck Agents (SCoBots).** SCoBots decompose the policy into consecutive interpretable concept bottlenecks (ICB). Objects and their properties are first extracted from the raw input, human-understandable functions are then employed to derive relational 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

Let us start with providing the notations of RL problems and concept bottleneck models (CBMs). In this work, we represent RL tasks as Markov decision processes (MDP),  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P_{s,a}, R_{s,a} \rangle$ , 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. Classic deep RL policies,  $\pi_\theta(s) = P(A = a | S = s)$  parametrized by  $\theta$ , 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  $g : x \rightarrow c$  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  $f : c \rightarrow y$  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 only from static data, but 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. 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

SCoBots separates the extraction of concepts and action selection into several inspectable steps that are grounded in human-understandable concept representations (*cf.* 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 use an initial concept extraction method to extract objects and their properties [Stammer et al., 2021, Koh et al., 2020]. Specifically, we start from an initial bottleneck model,  $\omega_{\theta_1}(\cdot)$ , parameterized by  $\theta_1$ . Given a state,  $s_t$ , this model extracts  $c_i$  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 extracted properties, (*e.g.* the object category, position (*i.e.*  $x, y$  coordinates), etc.). As done in previous works on CBMs, the bottleneck model of our SCoBots,  $\omega_{\theta_1}$ , 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,  $\mu$ , is parameterized by a predetermined set of transparent Python code-based relational functions,  $\mathcal{F}$ , used to extract a set of  $d_t$  relational concepts. These relations are based on each object (and its properties) for unary relations or on combinations of objects for n-ary relations and are denoted as  $\mu_{\mathcal{F}}(\Omega_t) = \Gamma_t = \{g_t^k\}_{k=1}^{d_t}$ . To avoid relying on human-defined strong priors, we make use of the

accumulated knowledge learned by Large Language Models (LLM) to create  $\mathcal{F}$ . We ask an LLM to provide the code of functions that encode “useful” relationships to solve Atari games, using the chain-of-thoughts principle [Wei et al., 2022]. All prompts and LLM’s answers are in App. A.7. The LLM provided an extensive set of relational functions (*e.g.* **distance**, **speed**). We selected a reduced set to reduce the computational burden of the agents (the list is in App. A.6). It constitutes the set of all concepts used by our agents. We believe that further analysis on the extraction of such LLM-provided functions is a exciting avenue for future work. Note that  $\mathcal{F}$  can include the identity, to let the relation extractor pass through initial object concepts from  $\Omega_t$  to the relational concepts  $\Gamma_t$ .

Finally, SCoBots employ an action selector,  $\rho_{\theta_2}$ , parameterized by  $\theta_2$ , on the relational concepts to select the best action,  $a_t$ , given the state  $s_t$  (*i.e.* the set of frames). Up to now, all extracted concepts in SCoBots, both  $\Omega_t$  and  $\Gamma_t$ , represent human-understandable concept representations. To guarantee understandability also within the action selection, 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 neural networks and decision trees (which in contrast represent inherently interpretable models). To trade off these issues of flexibility and performance vs. interpretability, we thus first break down the action selection process by initially training a small neural network as action selector,  $\tilde{\rho}_{\theta_2}$  (via gradient-based RL optimization). After this,  $\tilde{\rho}_{\theta_2}$  is distilled into a decision tree,  $\rho_{\theta_2}$ , which represents SCoBots final action selector. This is achieved via a neural network to decision tree distillation algorithm, *e.g.*, VIPER Bastani et al. [2018]. Lastly, note that one can add a residual link from the initial object concepts,  $\Omega_t$ , to the relational concepts,  $\Gamma_t$  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 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 the optimization of each policy component independently. Separating the training procedure of different components (similar to that of Koh et al. [2020]) particularly reduces the complexity of the overall optimization problem.

## 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 used within the SCoBots decision process: concept pruning and object-centric rewarding.

**Concept pruning.** The type and amount of concepts required by a successful policy may vary greatly across tasks. For instance, the objects’ **colors** are irrelevant in Pong but required to distinguish vulnerable ghosts from dangerous ones in MsPacman. At the same time, overloading the action selector with too many irrelevant concepts can lead to difficult optimization (*e.g.* shortcut learning) and difficult inspection of the decision tree ( $\rho$ ). Even for a single RL task, the need for specific concepts might change during training, *e.g.* in progressive environments (where agents are required to master early stages before being provided with 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,  $\bar{\Omega}$ . Additionally, by (ii) selecting a subset of the relational functions,  $\bar{\mathcal{F}}$ , or (iii) specifying which objects specific functions should be applied on, experts can implicitly define the relational concepts’ subset,  $\bar{\Gamma}$ . Guided SCoBots thus formally refine the mdp formulation:  $s_t \xrightarrow{\omega_{\theta_1}} \Omega_t \rightarrow \bar{\Omega}_t \xrightarrow{\mu_{\bar{\mathcal{F}}}} \bar{\Gamma}_t \xrightarrow{\rho_{\theta_2}} a_t$ . Furthermore, users can (iv) prune out redundant actions, resulting in a new action space  $\bar{\mathcal{A}}$ .

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 misalignment issues, such as those of agents playing Pong (cf. Fig. 1), where the agent 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. We provide a more extensive explanation, together with correction tests, in the experiments section.

**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:

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

where  $R^{\text{exp}} : \Gamma \rightarrow \mathbb{R}$  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, e.g. based on the distance between the agent and other 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.9 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. We also provide two ablation studies, of the neural network based action selector with guidance (NN) and without (NN+NG). We evaluate all agents on the latest v5 version of the environments<sup>3</sup> 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. 3 in App. A.5).

We focus our evaluations on the more interesting aspects of the agent’s reasoning process underlying

<sup>3</sup>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).

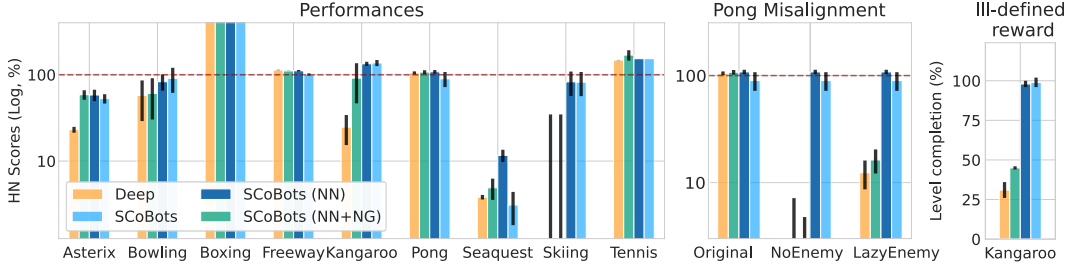


Figure 3: **Object-centric agents can master different Atari environments, and interactive SCoBots allow for corrections.** Human-normalized scores of deep and SCoBots agents trained using PPO on 9 ALE environments, including 3 SCoBots variants: non-guided deep policy (NN+NG), guided deep policy (NN) and guided decision tree policy (SCoBots). For numerical values, *cf.* App. A.2. 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 completion score of 100% on the intended goal’s evaluation in Kangaroo (right).

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 ( $\Omega_t$ ) based on the information from OCArari [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 LLM-provided 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.7 for further details).

SCoBots’ action selectors are encoded into decision trees, distilled from guided deep policy networks using VIPER [Bastani et al., 2018]. It is built on the idea of model distillation [Hinton et al., 2015] and imitation learning. The policy extraction is based on the imitation algorithm DAGGER [Ross et al., 2011] extended to use the Q-function as an oracle (Q-DAGGER). VIPER modifies Q-DAGGER to extract decision trees that are shallow and therefore easier to verify. Rollouts are collected from the neural network based policy and used to train these decision trees via imitation learning.

For the initial pruning of our guidance-based evaluations, we prune out the concepts that we consider unimportant to master each game (*cf.* Tab. 5 for a per-environment list). Furthermore, to mitigate RL-specific problems, we provide SCoBots with simulated human-expert reward signals, explained in each relevant answer below. For further details about the setup and the hyperparameters, *cf.* App. A.5.

### SCoBots learn competitive policies (Q1).

We first test SCoBots, as well as two variations, the concept based agent that use a neural network for the action selector, with (SCoBots (NN)) and without guidance (SCoBots (NN+NG)). 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 App. A.2. We observe that SCoBot agents perform at least on par with deep agents on all games but Seaquest (due to a failing distilled agent for 1 out of 3 seeds), despite the constraints of multiple bottlenecks within their architectures. Our results further suggest that even SCoBots with neural network action selectors (NN) and those without additional guidance (NN+NG) reach competitive performances on most tasks. Overall, our experimental evaluations show that RL policies based on interpretable concepts 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

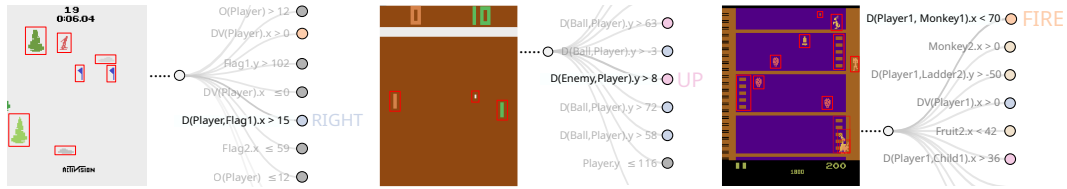


Figure 4: **SCoBots allow to follow their decision process**, because of their object centricty and interpretable concepts. The decision processes of SCoBots (extracted from the decision trees) with the associated states from a *guided* SCoBot on Skiing (left), and from normal SCoBots on Pong (middle) 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.

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.

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.8). 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.8). 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.8), 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 ( $\bar{\Omega}$ ). 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<sup>4</sup>, “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

<sup>4</sup>[www.retrogames.cz/play\\_195-Atari2600.php](http://www.retrogames.cz/play_195-Atari2600.php)



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 App. A.9). 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:

$$R^{\text{exp}} = \sum_{o \in \{\text{Flag}_1, \text{Flag}_2\}} D(\text{Player}, o)^t - D(\text{Player}, o)^{t-1} + V(\text{Player}).y. \quad (2)$$

These rewards are further detailed in App. A.9. 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.9.3 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 ( $\omega$ ) of OCArari 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 RL, 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 the past experience or MDP components that affect the current behavior, and (3) policy-level methods which describe 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).

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 hastily, 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

- Madhu S. Advani and Andrew M. Saxe. High-dimensional dynamics of generalization error in neural networks. *Neural Networks*, 2017.
- 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 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 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.
- 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. 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.
- Jay DeYoung, Sarthak Jain, Nazneen Fatema Rajani, Eric Lehman, Caiming Xiong, Richard Socher, and Byron C. Wallace. ERASER: A benchmark to evaluate rationalized NLP models. In *Conference of the Association for Computational Linguistics (ACL)*, 2020.
- 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.
- 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.
- Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys*, 2019.
- 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 E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *Arxiv preprint*, 2015.
- Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.
- 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.
- 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.
- 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.
- Agneza Krajna, Mario Brčić, 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.
- 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, 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.
- 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.
- 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.
- Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS*, 2011.
- 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.
- 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.
- 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.
- George A. Vouros. Explainable deep reinforcement learning: State of the art and challenges. *ACM Computing Surveys*, 2022.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 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.
- 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.

Çağlar Aytakin. 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.9.3), detailed numerical results (A.2), learning curves of our RL agents (A.3), the hyperparameters 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.

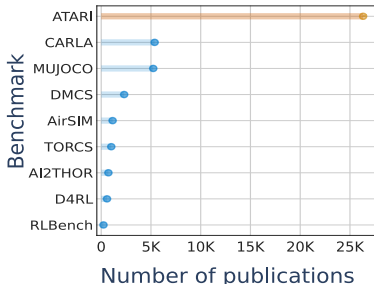


Figure 5: **The Atari Learning Environments is more used in scientific research than the next 8 other benchmarks together.** Graph borrowed from [Delfosse et al., 2023a].

### 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 reimplemented 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 3 training seeds ( $[0, 16, 32]$ +rank) with 8 actors each and evaluated in a two-stage process. First, during training, each training seed’s performance is evaluated every 500k frames on 4 differently seeded (42+training seed) evaluation environments for 8 episodes each. Second, after training, for every training seed, the best performing checkpoint is evaluated on 4 seeded (123, 456, 789, 1011) test environments for 5 episodes each. Finally, for every environment, the final return is determined by averaging the training seeds’ evaluation results. We use deterministic actions for both evaluation and testing stages.

Game	SCoBots NN+NG+v5	SCoBots NN+v5	SCoBots v5	PPO ours-v5	PPO ours-v4	PPO Sch.-v4
<b>Asterix</b>	5080±614.8	5043.3±729.9	4600.0±544.1	2126.7±148.6	10433.3±2772.9	4532
<b>Bowling</b>	106.6±41.8	137.4±24.1	148.4±40.4	102.2±39.1	51.4±18.0	40.1
<b>Boxing</b>	97.1±2.2	69.0±10.9	61.7±10.5	90.3±3.0	99.5±1.4	94.6
<b>Freeway</b>	32.8±0.1	32.9±0.7	29.8±0.9	33.6±0.2	33.6±0.3	32.5
<b>Kangaroo</b>	2776.6±1332.4	4050.0±217.8	4140.0±332.3	790.0±280.8	13296.7±1111.1	9929
<b>Pong</b>	17.2±1.9	17.5±1.8	11.0±6.3	16.4±1.5	20.9±0.2	20.7
<b>Seaquest</b>	1055.3±272.6	2411.3±377.0	692.3±259.9	837.3±46.7	1262.0±446.1	1204.5
<b>Skiing</b>	-23004±10333	-6530±3326	-6629.4±3258.1	-23004±10333	-22983±10333	-13901
<b>Tennis</b>	2.4±3.6	0.0±0.0	0.0±0.0	-0.9±0.1	-1.2±0.3	-14.8

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.



Game	Asterix	Bowling	Boxing	Freeway	Kangaroo	Pong	Seaquest	Skiing	Tennis
Random	210	23.1	0.10	0.00	52.0	-20.7	68.4	-17098	-23.8
Human	8503	160.7	4.3	29.6	3035	14.6	20182	-4336	-8.3

Table 2: The used random and human default scores, taken from van Hasselt et al. [2016].

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

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

### A.3 Learning curves

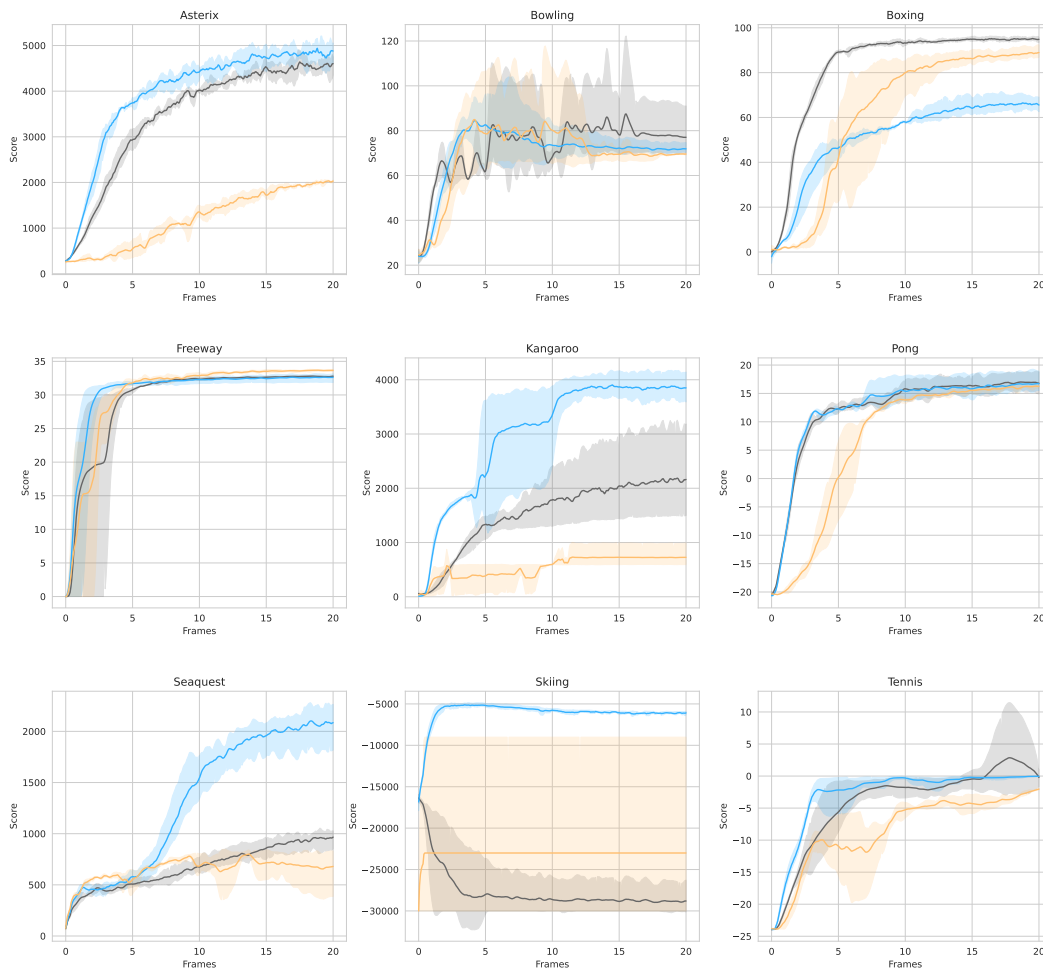


Figure 6: Training Return over frames ( $\times 10^6$ ) seen for ■ SCoBots (NN+v5), ■ SCoBots (NN+NG+v5) and ■ neural agents (ours-v5).

### A.4 Agent reasoning through the decision trees

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

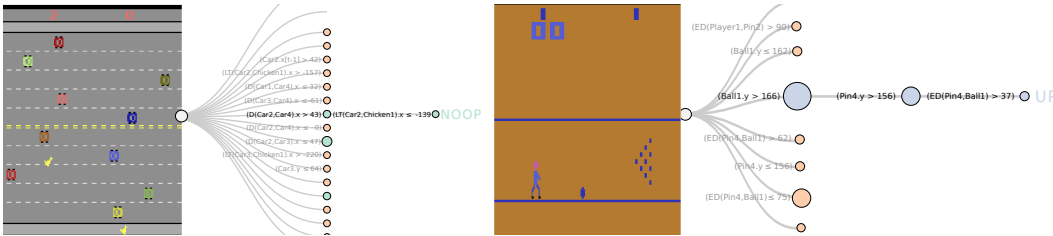


Figure 7: Decision trees from Freeway and Bowling.

### 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 SCoBot agent on 20M frames and using 8 actors takes approximately 8 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  $1 \times 10^{-3}$  (compared to  $2.5 \times 10^{-4}$ ). 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 ( $2 \times 64$ ) 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.

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<sup>5</sup>. 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 3.

The decision tree policy extraction was conducted using the VIPER algorithm with the following hyperparameters: 30.000 state-action pairs are generated from a guided deep policy SCoBot playing the respective game. Decision trees with the maximum depth of 7 are then consecutively generated and evaluated over 25 iterations. The best performing decision tree policy is then selected. This process is applied for every SCoBot training seed (A.2). All distilled decision tree policies are then evaluated as described in A.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 5 shows a detailed overview of the concepts and actions pruned for every environment evaluated.

<sup>5</sup>github.com/openai/baselines

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

Table 3: PPO Hyperparameter Values.  $\alpha$  linearly decreases from 1 to 0 over the course of training.

	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, when available and relevant
	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)
	top k	$Top_k((o_1, \dots, o_n), k, class)$	only $k$ closest ( $D_e$ to player) objects visible

Table 4: Descriptions of properties and relations used by SCoBots.

	Features	Asterix	Bowling	Boxing	Freeway	Kangaroo	Pong	Skiing	Seaquest	Tennis
Properties	class	✓	✓	✓	✓	✓	NE	✓	✓	✓
	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
	k closest objects	X	4	X	4	2	X	2	X	X
	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 5: Feature selection and pruning for guided SCoBots. ✓ denotes the included features, whereas X the features that are pruned out. NE refers to NoEnemy

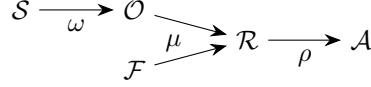


Figure 8: Functional summary of the SCoBots model architecture:  $\omega$  maps states to sets of objects,  $\mu$  maps sets of objects to relation vectors by applying relational functions, and  $\rho$  maps relation vectors to actions.

## 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.”

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 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  $f : O^k \rightarrow \mathbb{R}^n$  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  $\pi : \mathcal{S} \rightarrow \mathcal{A}$  given relation function set  $F$  is defined as  $\pi := \rho \circ \mu(\cdot, F) \circ \omega$ , where

1.  $\omega : \mathcal{S} \rightarrow \mathcal{O}$  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.  $\mu$  is the *feature selector*

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

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

That is,  $\mu$  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.  $\rho : \mathcal{R} \rightarrow \mathcal{A}$  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  $\omega$  and the feature selector  $\mu$ . 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 Relational functions provided by LLMs.

To obtain a set of relational function, we have asked chatGPT-4, in a chain-of-thought manner for relational functions, on March 26th 2024 using the following prompts:

*User* \_\_\_\_\_

I am programming an RL agent to play Atari games such as Pong, where the goal is: Knock the ball past the opponent’s orange paddle on the left (good), and don’t let it go past your green paddle on the right (bad). The detected objects inherit from the following class:

```

1 class GameObject:
2     def __init__(self):
3         self.rgb = (0, 0, 0)
4         self._xy = (0, 0)
5         self.wh = (0, 0)
6         self._prev_xy = None

```

Can you give me a set of relational functions that the RL agent can apply on any two objects and that can be used within its policy ?

*ChatGPT*

For your Reinforcement Learning (RL) agent designed to play games like Pong, relational functions between game objects can significantly aid in crafting an effective policy. These functions should enable your agent to understand the relative positions and movements of the paddle and ball to make strategic decisions. Here are some relational functions that can be applied to any two objects, such as the player's paddle, enemy paddle, and the ball:

1. **Relative Position:** Determines the relative position of one object to another on both the X (horizontal) and Y (vertical) axes. This is crucial for understanding whether the ball is above, below, or on a direct path with the paddle.
2. **Distance Between Objects:** Calculates the Euclidean distance between two objects. This can be particularly useful for determining how far the ball is from the paddle, aiding in movement decisions to intercept or hit the ball.
3. **Approaching Angle:** Estimates the angle at which the ball is approaching the paddle. This can help in predicting where the ball will hit the paddle and in adjusting the paddle's position to control the ball's rebound direction.
4. **Relative Velocity:** Compares the velocity (speed and direction) of the ball relative to the paddle. This can inform the agent about the speed at which it needs to move the paddle to intercept or hit the ball effectively.
5. **Future Intersection Point:** Predicts the future intersection point of the ball with the paddle based on the current trajectory of the ball. This advanced calculation helps in positioning the paddle in the right place at the right time.
6. **Is Ball Approaching or Receding:** Determines whether the ball is moving towards or away from a given object, like the player's paddle. This can be crucial for deciding when to move the paddle for an interception or to prepare for the next shot.
7. **Predicted Time Until Intersection:** Estimates the time it will take for the ball to reach the paddle or cross the player's or enemy's goal line, based on its current trajectory and speed. This helps in timing the paddle's movements.
8. **Middle point:** Center of the line between the two objects. This can be used to detect if the ball is on the center point between the ball and the paddle
9. **Overlap Detection:** Checks if two objects are overlapping or if their bounding boxes intersect. This is particularly useful for collision detection between the ball and the paddle.

Implementing these functions will require access to the properties of each `GameObject`, such as their positions (`_xy`), dimensions (`wh`), and possibly their previous positions (`_prev_xy`) to calculate velocities and trajectories. By integrating these relational functions into your RL agent's policy, you can enhance its ability to make strategic decisions based on the dynamic states of the game objects involved in Pong.

*User*

Can you give me the code for each of these functions ?

*ChatGPT*

Certainly! Below are Python functions that implement the relational concepts for objects in your Pong game. These assume that each object (player paddle, enemy paddle, ball) is an instance of `GameObject` or a subclass with the necessary properties.

First, ensure your `GameObject` class has all the necessary attributes. You mentioned a simplified version; let's expand it a bit for our purposes:

```

1 class GameObject:
2     def __init__(self, x=0, y=0, w=0, h=0):
3         self.rgb = (0, 0, 0)
4         self._xy = (x, y)
5         self.wh = (w, h)
6         self._prev_xy = (x, y)

```

Now, let's define the relational functions:

```

1 import math
2
3 def relative_position(object1, object2):
4     """Returns the relative position of object1 to object2."""
5     return object2._xy[0] - object1._xy[0], object2._xy[1] - object1._xy[1]
6
7 def distance_between_objects(object1, object2):
8     """Calculates the Euclidean distance between two objects."""
9     return math.sqrt((object2._xy[0] - object1._xy[0])**2
10                    + (object2._xy[1] - object1._xy[1])**2)
11
12 def approaching_angle(object1, object2):
13     """Estimates the angle at which object1 is approaching object2."""
14     dx, dy = relative_position(object1, object2)
15     return math.atan2(dy, dx)
16
17 def relative_velocity(object1, object2):
18     """Compares the velocity of object1 relative to object2."""
19     dx1 = object1._xy[0] - object1._prev_xy[0]
20     dy1 = object1._xy[1] - object1._prev_xy[1]
21     dx2 = object2._xy[0] - object2._prev_xy[0]
22     dy2 = object2._xy[1] - object2._prev_xy[1]
23     return dx1 - dx2, dy1 - dy2
24
25 def future_intersection_point(object1, object2, object1_velocity):
26     """Predicts future intersection point of object1 with object2
27     based on object1's velocity."""
28     # This is a simplified version assuming straight-line motion
29     # and doesn't account for changes in direction or speed.
30     dx, dy = object1_velocity
31     time_to_intersection = (object2._xy[0] - object1._xy[0]) / dx \
32                          if dx != 0 else float('inf')
33     predicted_y_intersection = object1._xy[1] + dy *
34     time_to_intersection
35     return object2._xy[0], predicted_y_intersection
36
37 def is_ball_approaching_or_receding(ball, paddle):
38     """Determines if the ball is moving towards
39     or away from the paddle."""
40     dx, _ = relative_velocity(ball, paddle)
41     return "Approaching" if dx < 0 else "Receding"
42
43 def predicted_time_until_intersection(object1, object2,
44     object1_velocity):
45     """Estimates time until object1 intersects with object2
46     based on current trajectory."""
47     dx, _ = object1_velocity
48     return (object2._xy[0] - object1._xy[0]) / dx
49     if dx != 0 else float('inf')
50
51 def middle_point(object1, object2):
52     """Finds the center point of the line between the two objects."""
53     return (object1._xy[0] + object2._xy[0]) / 2, (object1._xy[1] +
54     object2._xy[1]) / 2

```

```

53 def overlap_detection(object1, object2):
54     """Checks if object1 and object2 are overlapping."""
55     x1_min, y1_min = object1._xy
56     x2_min, y2_min = object2._xy
57     x1_max, y1_max = x1_min + object1.wh[0], y1_min + object1.wh[1]
58     x2_max, y2_max = x2_min + object2.wh[0], y2_min + object2.wh[1]
59     return not (x1_max < x2_min or x2_max < x1_min
60                or y1_max < y2_min or y2_max < y1_min)

```

These functions are starting points and might need adjustments based on your specific game mechanics, object properties, and how you've implemented motion in your game. For instance, the function for predicting the future intersection point assumes linear motion and doesn't account for the possibility of the ball changing direction upon hitting walls or paddles. You'll also want to fine-tune them based on the scale of your game's coordinates and velocity measurements.

*User*

Can you also give unary relationships, that apply e.g. to one object ?

*ChatGPT*

Certainly! Unary relational functions focus on properties and behaviors of a single object. These functions can help your RL agent understand the current state and potential future states of a single game object, such as the ball or paddle in Pong. Here are some useful unary relational functions:

1. **Speed and Direction:** Returns the speed and direction of an object based on its current and previous positions.

```

1 def speed(obj):
2     """Calculate the speed of an object based on its movement."""
3     dx = obj._xy[0] - obj._prev_xy[0]
4     dy = obj._xy[1] - obj._prev_xy[1]
5     speed = math.sqrt(dx**2 + dy**2)
6     return speed
7
8 def direction(obj):
9     """Calculate the direction of an object based on its movement."""
10    dx = obj._xy[0] - obj._prev_xy[0]
11    dy = obj._xy[1] - obj._prev_xy[1]
12    direction = math.atan2(dy, dx)
13    return direction

```

2. **Is Stationary:** Checks if the object is stationary (i.e., has not moved between the current and previous positions).

```

1 def is_stationary(obj):
2     """Determine if the object is stationary."""
3     return obj._xy == obj._prev_xy

```

3. **Predict Future Position:** Predicts the future position of the object if it continues in the same direction with the same speed.

```

1 def predict_future_position(obj, time_steps=1):
2     """Predicts the future position of the object
3     after a given number of time steps."""
4     dx = obj._xy[0] - obj._prev_xy[0]
5     dy = obj._xy[1] - obj._prev_xy[1]
6     future_x = obj._xy[0] + time_steps * dx
7     future_y = obj._xy[1] + time_steps * dy
8     return future_x, future_y

```



4. **Object Center:** Computes the center point of the object's bounding box. This can be useful for more accurately predicting collisions and rebounds.

```
1 def object_center(obj):
2     """Calculate the center of the object."""
3     center_x = obj._xy[0] + obj.wh[0] / 2
4     center_y = obj._xy[1] + obj.wh[1] / 2
5     return center_x, center_y
6 \end{mintlstlisting}
7
8 \item \textbf{Edge Distance to Walls}: Computes the minimum distance
9     from the object to the nearest wall of the playing field, assuming
10    you define the boundaries of your field.
11
12 \begin{lstlisting}
13 def edge_distance_to_walls(obj, field_width, field_height):
14     """Calculate the distance from the object
15     to the nearest edge of the field."""
16     dist_left = obj._xy[0]
17     dist_right = field_width - (obj._xy[0] + obj.wh[0])
18     dist_top = obj._xy[1]
19     dist_bottom = field_height - (obj._xy[1] + obj.wh[1])
20     return min(dist_left, dist_right, dist_top, dist_bottom)
```

5. **Velocity Magnitude:** Computes the magnitude of the object's velocity vector.

```
1 def velocity_magnitude(obj):
2     """Calculate the magnitude of the object's velocity."""
3     dx = obj._xy[0] - obj._prev_xy[0]
4     dy = obj._xy[1] - obj._prev_xy[1]
5     return math.sqrt(dx**2 + dy**2)
```

6. **Check if Moving Toward Wall:** Determines if the object is moving towards a specified wall. This can help decide if the paddle needs to change direction to avoid missing the ball.

```
1 def is_moving_toward_wall(obj, wall):
2     """Check if the object is moving toward a specified wall
3     ('left', 'right', 'top', 'bottom')."""
4     dx = obj._xy[0] - obj._prev_xy[0]
5     dy = obj._xy[1] - obj._prev_xy[1]
6     if wall == 'left' and dx < 0:
7         return True
8     if wall == 'right' and dx > 0:
9         return True
10    if wall == 'top' and dy < 0:
11        return True
12    if wall == 'bottom' and dy > 0:
13        return True
14    return False
```

7. **Is Object in Corner:** Determines if the object is located in one of the corners of the playing field. This can influence strategic decisions, especially for advanced levels of play.

```
1 def is_in_corner(obj, field_width, field_height):
2     """Determine if the object is in one of the corners of the field.
3     """
4     in_left = obj._xy[0] <= 0.1 * field_width
5     in_right = obj._xy[0] >= 0.9 * field_width - obj.wh[0]
6     in_top = obj._xy[1] <= 0.1 * field_height
7     in_bottom = obj._xy[1] >= 0.9 * field_height - obj.wh[1]
8     return (in_left or in_right) and (in_top or in_bottom)
```

These functions can help your RL agent analyze the game state and decide its next move by understanding the dynamics of single objects. Depending on your game implementation details, you might need to adjust these functions to match your specific coordinate system and game rules.

### A.8 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]).

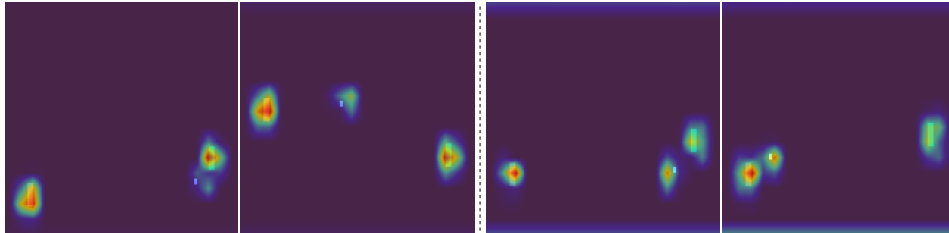


Figure 9: More importance maps of DQN agents playing Pong, with ReLU (left) and rational activation functions (right).

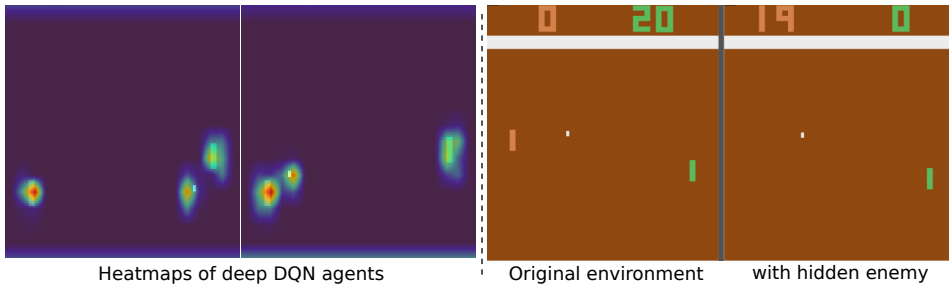


Figure 10: Importance maps of trained deep DQN agents playing Pong (left) show that all 3 moving objects are important for the agent’s decision, suggesting that the agent takes aims at sending the ball past the enemy. The same trained agent completely outperform its enemy in the original environment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

## A.9 RL specific caveats

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

### A.9.1 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:

```
1 player = _get_game_objects_by_category(game_objects, ["Player"])
2
3 # Get current platform
4 platform = np.ceil((player.xy[1] - player.h - 16) / 48) # 0: topmost,
5               3: lowest platform
6
7 # Encourage moving to the child
8 if not episode_starts and not last_crashed:
9     if platform % 2 == 0: # even platform, encourage left movement
10        reward = - player.dx
11    else: # encourage right movement
12        reward = player.dx
13
14    # Encourage upward movement
15    reward -= player.dy / 5
16 else:
17    reward = 0
```



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

### A.9.2 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):

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

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:

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

### A.9.3 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  $\sum_{n=1}^{\infty} (n \times 0.03 \times 0.97^{n-1}) = 33.3$ . The agent usually need 60 steps (in average) from the initialization of the point to the reward attribution, hence, it will need  $\sim 2200$  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.

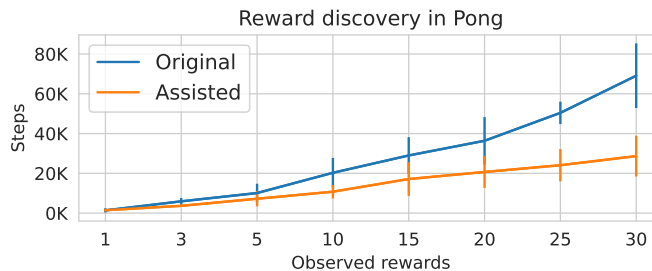


Figure 12: Expert user feedback allow for faster discovering of the reward signal in the sparse reward Pong environment.

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:

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

This extra reward signal lets a starting agent a winning shot every  $\sim 820$ , 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.9.4 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.

## NeurIPS Paper Checklist

### 1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [Yes]

Justification: In this paper, we claim that object-centric RL agents that use successive concept bottlenecks can challenge deep RL ones, and that they allow for easy detection and corrections of diverse RL-specific caveats. This is shown in Figure 3, both for the performances, as well as for the interpretability and correction.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation, as long as it is clear that these goals are not attained by the paper.

### 2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: The paper discusses SCoBots' limitations in the *Limitation* section included in the main paper.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

### 3. Theory Assumptions and Proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

### 4. Experimental Result Reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper presents the experimental setup and evaluation process in detail, both in the main paper and appendix.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.



## 5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The paper includes all information necessary to reproduce the experimental results and provides open access to a git code repository (using anonymous-github until publication, then the code will be publicly available on GitHub).

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

## 6. Experimental Setting/Details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: The experimental setup is described in the main paper, all hyperparameter values are listed in detail in the appendix and the code is available.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

## 7. Experiment Statistical Significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Error bars are reported in the main paper, detailed numerical results included in the appendix.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

## 8. Experiments Compute Resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The compute resources and time required are listed in detail in Appendix A5. The total compute time for our experiments was approximately 1200 CPU+GPU hours.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

## 9. Code Of Ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines?>

Answer: [Yes]

Justification: We discuss the Potential Harms and negative societal impacts that interpretable algorithms can provide, as a misintended person with discriminative behavior could increase the misalignment of such agents. We hope that overall, the transparency of agents' decisions will result in positive societal impact.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

## 10. Broader Impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The paper discusses societal impacts in an impact statement included in the main paper.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

#### 11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: We simply train transparent agents on Atari environments, we do not see societal negative impact that these agents could bring.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

#### 12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We cite every creator of the used environments and RL training repositories, all of which are publicly available and freely distributed.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.

- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, [paperswithcode.com/datasets](https://paperswithcode.com/datasets) has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

### 13. **New Assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: Our code is commented, but this does not constitute an asset.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

### 14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

### 15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: The paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.