Interpretable pipelines with evolutionarily optimized modules
for reinforcement learning tasks with visual inputs

Leonardo Lucio Custode
University of Trento
Trento, Italy
leonardo.custode@unitn.it

Giovanni Iacca
University of Trento
Trento, Italy
giovanni.iacca@unitn.it

ABSTRACT
The importance of explainability in AI has become a pressing concern, for which several explainable AI (XAI) approaches have been recently proposed. However, most of the available XAI techniques are post-hoc methods, which however may be only partially reliable, as they do not reflect exactly the state of the original models. Thus, a more direct way for achieving XAI is through interpretable (also called glass-box) models. These models have been shown to work well in RL tasks with raw input data, as in these contexts each decision depends on features extracted from raw data, and reasoning on the extracted high-level features. We test our approach in reinforcement learning environments from the Atari benchmark, where we obtain comparable results (with respect to black-box approaches) in settings without stochastic frame-skipping, while performance degrades in frame-skipping settings.

CCS CONCEPTS
• Computing methodologies → Artificial life; Multi-agent systems; Cooperation and coordination; Neural networks; • Theory of computation → Multi-agent learning.

KEYWORDS
Reinforcement Learning, Interpretability, Co-evolution, Atari

1  INTRODUCTION
While the progress in AI continues to achieve new milestones, there is a growing concern on the need for understanding the decision-making process of AI models, especially in critical applications. This awareness originated the subfield of explainable AI (XAI), which has the goal to design tools to explain the decisions made by AI models, usually by means of post-hoc techniques. However, while such techniques seem promising, they suffer from a fundamental issue: rather than reflecting the internal state of the explained model [29, 43], they mainly focus on explaining its output, and how its behavior depends (either locally or globally) on the features of the problem at hand. For this reason, another subfield of AI has been catching on in the past few years, namely that of interpretable AI (IAI) [5]. Differently from XAI, IAI focuses on the development of inherently interpretable models (also called “explainable by design” or “glass-box” models, as opposed to the traditional black-box ones), i.e., models that are directly understandable for humans without any post-hoc explanation.

Reinforcement learning (RL) is a particularly interesting setup for evaluating such models, since several real-world problems can be (and, in fact, have been) modelled as RL tasks [12], for instance in robotics [22], autonomous driving [25], unmanned aerial vehicles [1], production scheduling [39], resource allocation in the cloud [6], and medicine [16, 42]. In all these fields, explainability is an issue not only from a technical standpoint, but also from a legal and, to some extent, ethical perspective. The recent literature has proposed some seminal approaches for performing RL with interpretable models [11, 36], e.g., based on evolutionary computation [9, 10]. So far, these interpretable reinforcement learning (IRL) approaches have been mostly tested on relatively simple control RL tasks, such as those of the OpenAI gym benchmark [8], on which they have obtained fairly good results. However, these methods are not expected to work well in RL tasks with raw input data, as in these contexts each variable alone (e.g., a pixel) may not be meaningful enough to take decisions.

In this work, we introduce the concept of interpretable pipelines for tackling RL tasks with visual inputs. An interpretable pipeline is a multi-agent system where each agent is an interpretable model with well-defined responsibilities, which communicates with the other agents in the pipeline. We optimize such pipelines by means of a co-evolutionary approach, in which different evolutionary algorithms (EAs) run in parallel, each of which optimizes a single agent. We test our approach on three different Atari games, where we observe that the proposed method is able to achieve satisfactory performance in deterministic settings (i.e., without frame-skipping). On the other hand, our approach is not able to achieve satisfactory performance in environments with stochastic frame-skipping (yielding higher uncertainty about the future), which provides some hints for future work.

The paper is structured as follows. The next section makes a brief overview of the related work. Section 3 explains the methodology used in our experiments. In Section 5 we present the experimental setup and the results and, finally, in Section 6 we draw the conclusions of this work.
2 RELATED WORK

IRL has recently gained attention in the research community. Silva et al., in [36], employed differentiable decision trees trained by means of the PPO algorithm [35]. A differentiable decision tree is a decision tree that, instead of using binary conditions (also called hard splits), uses soft splits. Each soft split is defined as \( \sigma(x - o) \), where \( \sigma \) denotes the sigmoid function. Then, when computing the output for a given sample, the leaves are weighted according to the weights encountered during the path leading to that leaf, i.e., \( \sigma(x - o) \) for the “True” branch and \( (1 - \sigma(x - o)) \) for the “False” branch. This approach gives satisfactory results when using the differentiable version of the decision trees (which have low interpretability). However, when the produced trees are discretized into decision trees with hard splits (which have high interpretability), significant losses in performance occur.

Another interesting approach to IRL has been proposed in [11]. Here, the authors introduced a methodology that optimizes decision trees with non-linear splits by using an EA. The results show that the proposed approach works well in discrete-action settings. However, the highly non-linear splits limit the interpretability of the solutions produced.

In [10], the authors proposed a methodology based on Grammatical Evolution (GE) [30] and Q-learning [41] to produce decision trees that perform online learning. More specifically, the GE algorithm is used to optimize the inner splits of the decision trees, while Q-learning is used to learn the discrete actions for the leaves. This method was tested on three control tasks from the OpenAI gym benchmark, achieving state-of-the-art trade-offs in terms of performance and interpretability. In [9], this approach was further extended in order to handle RL tasks with continuous action spaces. Here, the authors employed a co-evolutionary system based on two independent evolutionary processes: the first one, based on GE, optimizes the decision trees; the second one, based on an Estimation of Distribution Algorithm [18], optimizes pools of continuous actions.

As mentioned earlier, the main limitation of these approaches is that while they can be effective in tasks with a small number of high-level features, they are not expected to work in environments with high-dimensional, low-level features, such as images. In fact, in the latter scenario, each input to the system does not provide significant information for the decision-making process. Moreover, even if one of those methods was able to obtain satisfactory performance by using a subset of the raw input data (i.e., a part of an image), making it generalize to other settings would be extremely hard. Furthermore, applying those methods straightforwardly on the raw data would achieve limited interpretability.

Concerning this latter aspect, it is important to note that interpretability intended as a binary property is ill-defined. For this reason, some works proposed approaches to quantitatively measure interpretability.

In [38], the authors learned a metric of interpretability by training a regression model on the results of a survey. The resulting metric was:

\[
M(\ell, n_o, n_{nao}, n_{naoc}) = 79.1 - 0.2\ell - 0.5n_o - 3.4n_{nao} - 4.5n_{naoc}
\]

where:

- \( \ell \) is the number of symbols in the formula;
- \( n_o \) is the number of operations in the formula;
- \( n_{nao} \) is the number of non-arithmetical operations;
- \( n_{naoc} \) is the maximum number of consecutive compositions of non-arithmetical operations.

This metric is intended to lie in \([0, 100]\), where 0 means “non-interpretable” and 100 means “interpretable”. However, when applying this formula on large models, there is the possibility that the \( M \)-score exceeds the bounds. For this reason, in [9, 10], the authors rewrote this metric as:

\[
M'(\ell, n_o, n_{nao}, n_{naoc}) = -0.2 + 0.2\ell + 0.5n_o + 3.4n_{nao} + 4.5n_{naoc}
\]

This version of the \( M \) metric works essentially as a complexity metric, and is defined in \([0, \infty)\). In this case, a constant model has \( M' = 0 \), which is the best possible value. As the value of the \( M' \) moves away from 0, the interpretability of the system decreases.

The idea of using complexity as a proxy for interpretability was also proposed in [4], where the authors stated that the computational complexity of a model can be used as a metric of interpretability as it directly resembles the number of operations that must be interpreted by humans.

3 METHOD

To evolve interpretable pipelines for image-based RL tasks, we build on some of the aforementioned previous works from the literature [9, 10, 37]. In detail, our proposed system is an interpretable pipelines composed of two parts:

- a vision module, that is meant to process the input to extract a pre-defined number of features;
- a decision module, whose purpose is to decide which action to take, based on the features extracted by the vision module.

It is important to note that in this work, features represent high-level visual information (position of relevant objects), while decisions are actions taken by a decision tree (playing one of the Atari games considered in our experimentation). However, the proposed pipelines can be extended to tackle classification problems.

A graphical representation of this kind of pipelines is shown in Figure 1. In the following subsections, we will first explain the details of the two kinds of modules, and then we will describe our co-evolutionary approach.

3.1 Vision module

In [37], the authors used a simplified self-attention module that ranks the patches of the image by importance. Then, the coordinates of the \( k \) most important patches are given in input to an LSTM network [19] that computes the decision to take, where \( k \) is a predetermined parameter. Similarly, in this work we use a vision module, whose purpose is to find the \( k \) most important patches in the image, returning their coordinates.

However, in order to have better interpretability, rather than a self-attention module, in our vision module we employ \( k \) convolutional kernels, each of which is supposed to detect a single entity of interest in the image.

Moreover, using \( k \) distinct kernels instead of a single self-attention module allows us to have a fixed-order constraint on the input: e.g.,
The goal of the decision module is to perform “reasoning” on the vision module, thus it does not use the raw data of the whole image. Specifically, it takes as input the list of coordinates computed by the automatically synthesized decision tree as decision module. This module then returns a discrete action that can be directly sent to the environment, see the examples shown in Figure 6 and the corresponding analysis reported in Section 5.1.

### 3.3 Co-evolutionary process

To optimize the vision and decision modules adopted within the proposed pipelines, we employ a co-evolutionary approach [28]. In particular, we combine Covariance Matrix Adaptation Evolution Strategies (CMA-ES) [17] with Genetic Programming (GP) [23], as shown in Figure 2. We use CMA-ES to evolve the parameters of the vision module, i.e., the weights of each kernel module. CMA-ES has been chosen for being one of the most robust algorithms for derivative-free optimization. On the other hand, by using Genetic Programming (more specifically, strongly typed Genetic Programming [27]), we evolve decision trees, as described below.

#### 3.3.1 Genetic Programming for evolving decision trees

To evolve decision trees, we use two types of nodes: condition nodes and leaf nodes. A condition is represented as a node with three child nodes: a comparison node and two conditions. A comparison node is composed of a node representing an operator (e.g., “less than”, “equal to”, and “greater than”) that has two child nodes, encoding two expressions.

An expression node can be either a constant, a variable, or an arithmetical operation between expression nodes.

Since the interpretability of a decision tree crucially depends on the complexity of the conditions (i.e., more complex hyperplanes are hard to interpret, e.g., the ones presented in [11]), not only we use a constant to limit the size of the tree, but we also employ a different constant to limit the depth of the conditions.

By doing so, we can control better the interpretability of the tree by allowing, for instance, deeper trees with simple conditions.

#### 3.3.2 Fitness evaluation

In order to evaluate the quality of the individuals from both populations, we pair each individual with all the individuals of the other population.

We evaluate the pair on $c$ episodes, and we compute the average score (across episodes) for the pair $(s_{i,j})$, where $i$ is the index of the
Kernels

\[ s_i \] that individual obtained across all the pairings, i.e. \( f_i = \max(s_{i,j}) \) for vision modules and \( f_j = \max(s_{i,j}) \) for decision modules.

While using a different operator such as the mean (across pairings) seems more meaningful, from preliminary experiments we observed that using the mean leads to a stagnation of the co-evolutionary process. We hypothesize that this is due to the fact that, by using the mean, the fitnesses are affected by the randomness in the evaluation phase, so that an individual that obtains medium-low scores in all the pairings may have a higher fitness than an individual that works very well combined with a specific individual of the other population but works poorly with all the other individuals.

\[ \bar{f}_i = \frac{1}{\epsilon} \sum_{e=0}^{\epsilon-1} \bar{f}_{i,e} \]

\[ c = \frac{p_c \cdot p_g \cdot g \cdot \epsilon}{\gamma} \]

3.3.3 Reducing the number of evaluations. The computational cost (in terms of number of evaluations, where a single evaluation relates to an instance of the proposed pipelines) for the co-evolutionary process is \( \mathcal{O}(p_c \cdot p_g \cdot g \cdot \epsilon) \), where \( p_c \) is the population size for CMA-ES, \( p_g \) is the population size for the Genetic Programming (assuming that each individual of one population is paired with all the individuals of the other population), \( g \) is the number of generations, and \( \epsilon \) is the number of episodes.

To reduce such cost, we propose a mechanism that evaluates the behavior (i.e., the output) of each individual in the two populations, and avoids the evaluation of individuals whose behavior is too similar. At each generation, for both the vision and decision modules in the current populations of the co-evolutionary process, we give them in input a set of samples and we store their outputs. Then, for both populations (separately), we cluster these outputs by means of the DBSCAN algorithm [34]. More specifically, the samples used for clustering are calculated as follows.

- **Vision modules** – For these modules, we evaluate their behavior by giving them in input a set of \( n_{vm} \) images sampled randomly from an episode used at the beginning of the evolutionary process. Then, we use the concatenation of all the outputs of each module as input samples for the clustering process.

- **Decision modules** – For these modules, we evaluate their behavior by giving them in input \( n_{dm} \) randomly sampled coordinates from the vision modules’ output space. Then, we perform a one-hot encoding of the decision modules’ outputs and, by concatenating all the one-hot encoded outputs for each module, we obtain the input samples for the clustering process. The reason underlying the one-hot encoding is related to the “meaning” of the input samples for the clustering process. In fact, if we used the raw outputs of the decision modules as input samples for clustering, we would give an implicit “proximity” meaning such that an action \( a_i \in A = \{a_0, \ldots, a_i, a_{i+1}, a_{i+2}, \ldots, a_A \} \) would be considered “closer” to action \( a_{i+1} \) than \( a_{i+2} \), while the meaning of such actions may not have such a “similarity” principle. Instead, by performing a one-hot encoding, the distance between two different actions (performed by different decision modules given the same input) is always constant, thus removing the bias associated to the raw outputs.

Once clustering has been performed, we evaluate only the centroids of each cluster (and the individuals not assigned to any cluster) and we assign, for each individual of the cluster, the same fitness.

The reason underlying the choice of the DBSCAN algorithm is due to the fact that this algorithm is based on the concept of *density*, so that we can intuitively set the thresholds for considering two points as “close”. Moreover, this algorithm does not require to specify a pre-defined number of clusters, so this means that if we find a cluster, its points are close enough to be considered similar. Finally, we do not use constant parameters for the distance threshold \( \epsilon \). Instead, we use an initial \( \epsilon_0 \), which, at each step, is multiplied by a scaling constant \( \gamma \in [0, 1] \). This scaling mechanism gives us the following properties. At the beginning of the evolutionary process, since the diversity is high, we perform a coarse-grained clustering of the individuals so that we can significantly speed up the initial generations. On later generations, the thresholds become increasingly smaller, so that even not-so-diverse individuals are evaluated separately: this, in turn, leads to a greater number of simulations, which allows us to discriminate individuals in a more fine-grained fashion. Finally, when the \( \epsilon_i \) parameter (for the \( i \)-th generation) tends to zero, we avoid evaluating only those individuals that are behaviorally identical. However, since in the final generation the evolutionary process is expected to converge, the number of evaluations decreases again.

4 EXPERIMENTAL SETUP

We test our approach in three environments from the Atari Learning Environment implemented in OpenAI Gym [7, 8, 26]. In particular, we use the Pong-v4, Bowling-v4, and Boxing-v4 environments, hereinafter simply referred to as Pong, Bowling and Boxing, respectively. Table 1 shows the parameters used for the evolution. The parameters have been determined empirically, based on the knowledge gained in preliminary experiments. The parameter setting is the same for the three environments. The only difference is in the number of available actions for the decision module, that is 4, 4 and 6 respectively for Pong, Bowling and Boxing. It is important to note that the number of episodes is quite low. This is due to the fact that in the fitness evaluation phase there is no learning involved, and thus the episodes are only used to evaluate the pipelines on average. While a higher number of episodes would certainly increase the precision of the estimate of the performance of the pipeline in unseen episodes, it would significantly increase the computational complexity.

![Figure 2: Scheme of the proposed co-evolutionary process.](image-url)
cost of the search process. All the runs were performed using an
HPC that allocated 70 CPUs and 4GB of RAM for each run, with
a time limit of 6 hours. For each environment, we consider both
the setting with and without frame-skipping, the first one being
harder than the second one. For each environment and setting, we
perform 5 runs. This number of runs is enough to ensure statistical
significance since, given the results shown in Table 2, the confi-
dence interval (95%) is low enough to validate our conclusions (see
the next section for the results). The confidence interval has been
computed as $C_{95}\% = \frac{t_{(1-0.05)}\sigma}{\sqrt{n}}$, where $t$ is the critical value from
the Student’s t distribution, and $\sigma$ is the standard deviation.

## 4.1 Image pre-processing

Before feeding an image to the vision module, we perform the
following pre-processing steps:

1. We remove the topmost 35 pixels: these pixels correspond to
   the part of the image that describes the “status” of the game,
   thus we remove it to avoid that the evolved pipelines use this
   information to take decisions.

2. We resize the image to 96×96: this operation speeds up the vi-
   sion modules’ computations, without losing information about
   the entities present in the game.

3. We normalize the input in [0, 1] by performing a min-max
   normalization.

## 4.2 Environments

All the environments share the same type of observations, which
consist in 210 × 160 RGB images, where each pixel is encoded with
three 8-bit integers (one for each channel).

### 4.2.1 Pong

The Pong environment (and its counterpart without
frame-skipping, PongNoFrameskip) is a game in which there are
two agents (here intended as “players”, not to be confused with the
agents that compose our pipelines), who play the Pong game. Each
agent controls a racket (which can move on the $y$ axis), and the
goal of each agent is to send the ball farther than the opponent’s $x$
position.

**Reward**: Each point scored by the playing agent (i.e., the green
racket) gives a reward of 1 point. On the other hand, each point
scored by the opponent agent gives a reward of -1 points. In all the
other cases, the reward given to the agent is 0.

**Actions**: The action space consists of 4 actions: two of them are
NOP actions (i.e., they do not move the racket), one moves the
racket upwards, and another moves the racket downwards.

**Termination criterion**: The simulation ends when either the agent
or the opponent score 21 points or after $10^5$ elapsed timesteps.

### 4.2.2 Bowling

The Bowling environment (and its counterpart BowlingNoFrameskip) consists in a bowling game, where the agent
has to throw a ball to hit some pins. Thus, differently from Pong
this is a single-player game.

**Reward**: After each round (i.e., throwing twice the balls, or one in
case of a strike), the agent receives a reward equal to the number of
pins that have been hit. Moreover, strikes and spares provide extra
points (also given at the end of the round). In all the other cases,
the reward given to the agent is 0.

**Actions**: The agent can perform 4 actions: a NOP action, an action
that moves the ball upwards, one that moves the ball downwards,
and an action that allows the agent to throw the ball.

**Termination criterion**: The simulation ends after 10 rounds or
after $10^5$ timesteps. However, since an agent that does not know
how to throw the ball will make the simulation become extremely
time consuming, we reduced the amount of maximum timesteps to
$5 \times 10^3$ during the evolutionary process. Then, to obtain the results
shown in Section 5, we test the best pipelines evolved on the full
task.

### 4.2.3 Boxing

In the Boxing (and its counterpart BoxingNoFrameskip) environment, there are two agents that compete: the white boxer
(played by the agent) and the black boxer (which is the opponent).

**Reward**: When the agent hits the opponent, it can receive either 1
or 2 points, depending on the distance between the agents (hitting
the opponent from a closer position gives more points). On the
other hand, when the agent is hit by the opponent it receives a
reward that is the opposite of the one previously described. In all
the other timesteps, the reward given to the agent is 0.

**Actions**: The environment provides 18 actions, composed of: NOP,
movements in the 4 cardinal directions, punching, and combina-
tions of movements and punching (including diagonals). However,
here for simplicity we reduce the set of actions to the non-composite
actions, i.e.: NOP, movements in the 4 directions and punching.

**Termination criterion**: The simulation ends when either the agent
or the opponent score 100 points or after $10^5$ elapsed timesteps.

### Table 1: Parameters used in the experimentation.

| Parameter                                   | Value     |
|----------------------------------------------|-----------|
| CMA-ES Population size ($p_c$)               | 50        |
| CMA-ES Initial mean                          | 0         |
| CMA-ES Initial $\sigma$                      | 0.1       |
| GP Population size ($p_\theta$)              | 50        |
| GP Crossover probability                     | 0         |
| GP Mutation probability                      | 1         |
| GP Tournament size                           | 10        |
| GP Elitism                                   | Yes (1 elite) |
| DBSCAN Number of samples (vision) ($n_{en}$)  | 100       |
| DBSCAN Number of samples (decision) ($n_{dm}$)| 100      |
| Number of generations ($g$)                  | 100       |
| Maximum depth of decision tree               | 4         |
| Maximum depth of condition                   | 2         |
| Number of convolutional kernels ($k$)        | 2         |
| Size of convolutional kernels                | 5×3×3     |
| Size of image                                | 96×96     |
| Number of episodes ($e$)                     | 3         |
| Time limit                                   | 6 hours   |
| Number of runs                               | 5         |

## 5 EXPERIMENTAL RESULTS

The results obtained from the 5 available runs for each environment
and setting are shown in Table 2. The results shown in the table
have been obtained by testing the best evolved pipelines on 100
unseen episodes. We observe that our approach performs well in
the environments without frame-skipping, but it performs poorly in
settings with frame-skipping. While state-of-the-art approaches are able to achieve very good performance even in cases with frame-skipping, it is important to point out that these approaches are not interpretable, thus they do not provide any information about their inner processes. On the other hand, while our approaches do not perform well when trained in setups with frame-skipping, they are completely transparent, potentially allowing an adaptation to domains with frame-skipping. In fact, the non-interpretable approaches have $M'$ scores (see Eq. 1) in the order of $10^7$, while ours are in the order of $10^2$. This difference can be further appreciated in Figure 4. These results encourage future research in IRL, as adding more complexity to the pipelines would still yield a significant gain in interpretability w.r.t. the current non-interpretable state-of-the-art.

Figure 3 shows a comparison of the distribution of the scores of the best pipelines evolved in 5 runs of each environment and setting (normalized w.r.t. the minimum and maximum possible scores of the environments). Once again we observe that, in all the cases, the settings without frame-skipping achieve very good performance (i.e., they are closer to one), while in the cases where frame-skipping is applied our algorithm is not able to achieve good performance.

### Table 2: Summary of the results of the best pipelines evolved in 5 runs of each setting for each environment.

| Env. | Setup       | Mean | Std. | Best | Reference |
|------|-------------|------|------|------|-----------|
| Pong | FS (ours)   | -8.97| 8.49 | 7.42 | [31, 32, 40] |
|      | NoFS (ours)  | 21.00| 0.00 | 21.00|           |
|      | FS (SoTA)   | -    | -    | 21.00|           |
| Bowling | FS (ours)   | 189.68| 5.06 | 196.53|           |
|      | NoFS (ours)  | 220.20| 18.00| 240.00|           |
|      | FS (SoTA)   | -    | -    | 260.00|           |
| Boxing | FS (ours)   | 48.59| 19.05| 75.37| [3, 7, 14, 15, 20, 32, 33] |
|      | NoFS (ours)  | 92.78| 3.09 | 98.00|           |
|      | FS (SoTA)   | -    | -    | 100.00|           |

The fitness trends, as well the cumulative number of evaluations across generations (mean, represented as solid line, ± std. dev., represented as shaded area, across 5 runs) are shown for each environment and setting in Figure 5. We observe that the clustering mechanisms allows us to save a significant amount of evaluations, increasing the efficiency of the co-evolutionary process (approximately saving 41%±12% evaluations).

### 5.1 Analysis of the best evolved pipelines

We conclude our presentation of the results with an analysis of the best evolved pipelines obtained by means of the proposed method. Note that, since the settings without frame-skipping produced better results, we will limit our analysis to these settings.

![Figure 3: Distribution of the normalized scores of the best pipelines evolved in 5 runs of each setting for each environment, normalized w.r.t. the minimum and maximum scores allowed by each environment.](image)

![Figure 4: Comparison in terms of average normalized score across 100 episodes and $M'$ (see Eq. 1) between the best evolved pipelines and the state-of-the-art (SoTA) methods from the references reported in Table 2.](image)

The best pipelines obtained are shown graphically in Figure 6. In order to improve the readability, the decision modules have been manually simplified, deleting the conditions that always evaluate to the same truth value.

#### 5.1.1 Pong.

As shown in Figure 6a, the vision module of the best pipeline evolved for Pong detects the two most important entities: The racket and the ball, giving in output their coordinates: $x_r, y_r, x_b, y_b$, where the subscript $r$ refers to the player’s racket and the subscript $b$ refers to the ball.

The policy of the decision-making module for this environment, as shown in Figure 6d, works as follows. First of all, it checks whether the racket is on the upper part of the screen. If so, it checks whether the y-coordinate of the ball is less than the x-coordinate of the racket. This condition can be simplified: the horizontal position
of the racket is constant \( x_r = 82 \). So, this condition is equivalent to \( y_b < 82 \), which means that the ball is not near to the bottom wall (white part in Figure 6a). Then, if this condition evaluates to True, the decision module decides to go downwards, otherwise if does not perform any action. On the other hand, when the racket is on the lower part of the screen, the decision module checks \( 87.4 > y_b \), i.e., if the ball is not near to the bottom wall. If so, it decides to go upwards, otherwise it does not perform any action.

5.1.3 Boxing. In Figure 6c, we show the two entities recognized by the vision module: the punch of the player, and the position of the right arm of the opponent. Thus, the vision module returns their coordinates: \( x_p, y_p, x_o, y_o \), where the subscript \( p \) refers to the player, and the subscript \( o \) refers to the ball.

The decision module (Figure 6f) works as follows. If the player is positioned upper than the opponent (again, note that the top-left corner has coordinates \((0, 0)\)), it tries to punch the opponent, otherwise it goes upwards, to reach the opponent.

Interestingly, such a simple policy (encoded by the decision module) allows us to understand some properties about the pipeline and the game itself. First, the game can be played with very good results by using only two actions (note that the environment provides 18 actions for the player). Moreover, the other actions are not needed because the opponent chases the player. For this reason, the decision module finds more advantageous to use a semi-defensive strategy, i.e., always punch if the opponent is reachable by the punches, and chase it only when it is upwards.

Figure 5: Fitness and number of evaluations (mean ± std. dev. across 5 runs) over time. For the fitness, the reference is the best possible fitness allowed by the environment. For the number of evaluations, the reference is the number of evaluations needed if the clustering mechanism described in Section 3.3.3 is not applied.
Interpretable pipelines with evolutionarily optimized modules for RL tasks with visual inputs,

Custode and Iacca

![Vision module - Pong](image1.png)

![Vision module - Bowling](image2.png)

![Vision module - Boxing](image3.png)

![Decision module - Pong](image4.png)

![Decision module - Bowling](image5.png)

![Decision module - Boxing](image6.png)

**Figure 6:** Best evolved pipelines for the three environments (without frame-skipping). On the top, we show the entities discovered by the vision modules, while on the bottom we show the corresponding decision modules.

6 CONCLUSIONS AND FUTURE WORKS

Reinforcement learning (RL) has made significant progresses in recent years. However, mainstream RL methodologies, typically based on deep learning, are very hard to understand. In this paper, we proposed a novel methodology (based on a kind of divide-et-impera paradigm) for evolving interpretable systems for RL tasks with visual inputs. In particular, our approach is based on pipelines characterized by a separation of concerns between a vision module (which uses convolutional kernels) and a decision module (based on a decision tree). Our results show that our approach is able to learn how to effectively play three Atari games in simplified settings (i.e., without frame-skipping). However, when applying frame-skipping to the environments, our approach is not able to achieve satisfactory performance.

Future work should introduce ways to address the uncertainty in non-deterministic settings (i.e., with frame-skipping), in order to make this approach more robust to noise in the environment and achieve performances comparable to those of the state-of-the-art algorithms developed for these settings. In this sense, two possibilities would be to incorporate in our approach some mechanisms used in evolutionary optimization in the presence of noise [2], or using fuzzy [21] or probabilistic [24] decision trees.
