Extending Environments to Measure Self-reflection in Reinforcement Learning

Samuel A. Alexander*
Michael Castaneda†
Kevin Compher‡
Oscar Martinez§

Abstract

We consider an extended notion of reinforcement learning in which the environment can simulate the agent and base its outputs on the agent's hypothetical behavior. Since good performance usually requires paying attention to whatever things the environment's outputs are based on, we argue that for an agent to achieve on-average good performance across many such extended environments, it is necessary for the agent to self-reflect. Thus, an agent's self-reflection ability can be numerically estimated by running the agent through a battery of extended environments. We are simultaneously releasing an open-source library of extended environments to serve as proof-of-concept of this technique. As the library is first-of-kind, we have avoided the difficult problem of optimizing it. Instead we have chosen environments with interesting properties. We give examples and introduce a simple transformation which experimentally seems to increase self-reflection.

1 Introduction

An obstacle course might react to what you do: for example, if you step on a certain button, then spikes might appear. If you spend enough time in such an obstacle course, you should eventually figure out such patterns. But imagine an "oracular" obstacle course which reacts to what you would hypothetically do in counterfactual scenarios: for example, there is no button, but spikes appear if you *would* hypothetically step on the button if there was one. Without self-reflecting about what you would hypothetically do in counterfactual scenarios, it would be difficult to figure out such patterns. This suggests that in order to perform well (on average) across many such obstacle courses, some sort of self-reflection is necessary.

This is a paper about empirically estimating the degree to which a Reinforcement Learning (RL) agent is self-reflective. By a self-reflective agent, we mean an agent

*The U.S. Securities and Exchange Commission, samuelallenalexander@gmail.com
†Brooklyn College
‡In-Q-Tel
§The U.S. Securities and Exchange Commission

which acts not just based on environmental rewards and observations, but also based on considerations of its own hypothetical behavior. We propose that an RL agent's degree of self-reflection can be estimated by running the agent through a battery of environments which we call *extended environments*, environments which react not only to what the agent does but to what the agent would hypothetically do. For good performance averaged over many such environments, an agent would need to self-reflect about itself, because otherwise, environment responses which depend on the agent's own hypothetical actions would often seem random and unpredictable. The extended environments which we consider are a departure from standard RL environments, but this does not interfere with their usage for judging standard RL agents: one can run a standard agent in an extended environment in spite of the latter's non-standardness.

To understand why extended environments (where the environment reacts to what the agent would hypothetically do) incentivize self-reflection, consider a game involving a box. The box's contents change from playthrough to playthrough, and the game's mechanics depend upon those contents. The player may optionally choose to look inside the box, at no cost: the game does not change its behavior based on whether the player looks inside the box. Clearly, players who look inside the box have an advantage over those who do not. The extended environments we consider are similar to this example. Instead of a box's contents, the game's mechanics depend upon the player. Rather than looking into a box, the environment "looks into" the player (by simulating a copy of the player) and adjusts its mechanics accordingly. Just as the player in the example gains advantage by looking in the box, an agent designed for extended environments could gain an advantage by examining itself, that is, by self-reflecting.

One might try to imitate an extended environment with a non-extended environment by backtracking—rewinding the environment itself to a prior state after seeing how the agent performs along one path, and then sending the agent along a second path. But the agent itself would retain memory of the first path, and the agent's decisions along the second path might be altered by said memories. Thus the result would not be the same as immediately sending the agent along the second path while secretly simulating the agent to determine what it would do if sent along the first path.

Alongside this paper, we are publishing an MIT-licensed open-source library [1] of extended environments to "ease adoption by other machine-learning researchers" [23]. We are inspired by similar (but non-extended) libraries and other benchmark collections [3] [4] [5] [7] [8] [10] [19] [24]. Our library is intended to show that it is possible to numerically estimate the self-reflectiveness of RL agents. Aside from measuring self-reflectiveness of individual agents, such a benchmark can also be used to experimentally test agent transformations intended to make agents more self-reflective (we introduce one such transformation in Section 5).

Our benchmark is based on Legg and Hutter's theory of universal intelligence measurement [15]. Legg and Hutter argue that to perfectly measure RL agent performance, one should aggregate the agent's performance across the whole space of all sufficiently well-behaved environments, weighted using an appropriate distribution. Rather than a uniform distribution (susceptible to no-free-lunch theorems), Legg and Hutter suggest assigning more weight to simpler environments and less weight to more complex environments. Thus, a high-order approximation of Legg and Hutter's idealized measure would need to involve representative environments of several complexities, weighted

accordingly. But measuring these complexities is hard and subjective [17], and we make no attempt to do so. Our library is intended as a rough first-order approximation, using only $n = 25$ simple extended environments, each with weight $1/n$ (all other environments are considered to have weight $0 \neq 1/n$, so the distribution is non-uniform and no-free-lunch does not apply: the same reason why no-free-lunch does not apply to the ubiquitous Atari benchmark [3]). In choosing those n simple extended environments, we have sought environments interesting in their own right, exhibiting paradoxes, interesting thought experiments, or counter-intuitive winning strategies. We will discuss examples in Section 3.

2 Preliminaries

We take a formal approach to RL to make the mathematics clear. This formality differs from how RL is implemented in practice. In Section 4 we will discuss a more practical formalization (which allows non-determinism).

Our formal treatment of RL is based on Section 4.1.3 of [14], except that we assume the agent receives an initial percept before taking its initial action (whereas in [14], the agent acts first). We will write $x_1y_1 \dots x_ny_n$ for the length-$2n$ sequence $\langle x_1, y_1, \dots, x_n, y_n\rangle$ and $x_1y_1 \dots x_n$ for the length-$(2n-1)$ sequence $\langle x_1, y_1, \dots, x_n\rangle$. In particular when $n = 1$, we will write $x_1y_1 \dots x_n$ for $\langle x_1\rangle$, even if y_1 is not defined. We assume fixed finite sets of actions and observations. By a *percept* we mean a pair $x = (r, o)$ where o is an observation and $r \in \mathbb{Q}$ is a reward.

Definition 1. *(RL agents and environments)*

1. *A (non-extended)* environment *is a function μ which outputs an initial percept $\mu(\langle\rangle) = x_1$ when given the empty sequence $\langle\rangle$ as input and which, when given a sequence $x_1y_1 \dots x_ny_n$ as input (where each x_i is a percept and each y_i is an action), outputs a percept $\mu(x_1y_1 \dots x_ny_n) = x_{n+1}$.*

2. *An* agent *is a function π which, given a sequence $x_1y_1 \dots x_n$ as input (each x_i a percept, each y_i an action), outputs an action $\pi(x_1y_1 \dots x_n) = y_n$.*

3. *If π is an agent and μ is an environment, the* result of π interacting with μ *is the infinite sequence $x_1y_1x_2y_2 \dots$ defined by:*

$$\begin{aligned} x_1 &= \mu(\langle\rangle) & y_1 &= \pi(\langle x_1\rangle) \\ x_2 &= \mu(\langle x_1, y_1\rangle) & y_2 &= \pi(\langle x_1, y_1, x_2\rangle) \\ &\dots & &\dots \\ x_n &= \mu(x_1y_1 \dots x_{n-1}y_{n-1}) & y_n &= \pi(x_1y_1 \dots x_n) \\ &\dots & &\dots \end{aligned}$$

In the following definition, we extend environments by allowing their outputs to depend also on π. Intuitively, extended environments can simulate the agent. This can be considered a dual version of AIs which simulate their environment, as in Monte Carlo Tree Search [6].

Definition 2. *(Extended environments)*

1. *An* extended environment *is a function μ which outputs initial percept $\mu(\pi, \langle\rangle) = x_1$ in response to input $(\pi, \langle\rangle)$ where π is an agent; and which, when given input $(\pi, x_1 y_1 \dots x_n y_n)$ (where π is an agent, each x_i is a percept and each y_i is an action), outputs a percept $\mu(\pi, x_1 y_1 \dots x_n y_n) = x_{n+1}$.*

2. *If π is an agent and μ is an extended environment, the* result of π interacting with μ *is the infinite sequence $x_1 y_1 x_2 y_2 \dots$ defined by:*

$$\begin{aligned} x_1 &= \mu(\pi, \langle\rangle) & y_1 &= \pi(\langle x_1 \rangle) \\ x_2 &= \mu(\pi, \langle x_1, y_1 \rangle) & y_2 &= \pi(\langle x_1, y_1, x_2 \rangle) \\ &\dots & &\dots \\ x_n &= \mu(\pi, x_1 y_1 \dots x_{n-1} y_{n-1}) & y_n &= \pi(x_1 y_1 \dots x_n) \\ &\dots & &\dots \end{aligned}$$

For the sake of simpler mathematics, we have not included non-determinism in our formal definition, but in practice, agents and environments are often non-deterministic, so that π and n do not determine $x_1 y_1 \dots x_n y_n$ (our practical treatment, discussed in Section 4, does allow non-determinism).

The fact that classical agents can interact with extended environments (Definition 2 part 2) implies that various universal RL intelligence measures [15] [11] [9] [16], which measure performance in (non-extended) environments, easily generalize to measure self-reflective intelligence (performance in extended environments). For example, Legg and Hutter's universal intelligence measure $\Upsilon(\pi)$ is defined to be agent π's average reward-per-environment, aggregated over all (non-extended) environments with suitably bounded rewards, each environment being weighted using the algorithmic prior distribution [18]. Simply by including suitably reward-bounded extended environments, we would immediately obtain a variation $\Upsilon'(\pi)$ which measures the performance of π across extended environments—possibly a version of Legg and Hutter's universal intelligence which measures both intelligence and self-reflection ability.

3 Some interesting extended environments

In this section, we give some examples.

3.1 A quasi-paradoxical extended environment

Example 3. *(Rewarding the Agent for Ignoring Rewards) For every percept $x = (r, o)$, let $x' = (0, o)$ be the result of zeroing the reward component of x. Fix some observation O. Define an extended environment μ as follows:*

$$\mu(\pi, \langle\rangle) = (0, O),$$
$$\mu(\pi, x_1 y_1 \dots x_n y_n) = \begin{cases} (1, O) & \text{if } y_n = \pi(x_1' y_1 \dots x_n'), \\ (-1, O) & \text{otherwise.} \end{cases}$$

In Example 3, when the agent takes an action y_n, μ simulates the agent in order to determine: would the agent have taken the same action if the history so far were identical except all rewards were 0? If so, μ gives the agent $+1$ reward, otherwise, μ gives the agent -1 reward. Thus, the agent is rewarded for ignoring rewards. This seems paradoxical. Suppose an agent guesses the pattern and begins deliberately ignoring rewards, as long as the rewards it receives for doing so are consistent with that guess. In that case, does the agent ignore rewards, or not? The paradox, summarized: "I ignore rewards because I'm rewarded for doing so."

We implement Example 3 as IgnoreRewards.py in our library [1].

3.2 A counterintuitive winning strategy

Example 4. *(Tempting Button) Fix an observation B ("there is a button") and an action A ("push the button"). For each percept-action sequence $h = x_1y_1 \dots x_n$, if the observation in x_n is not B, then let h' be the sequence equal to h except that the observation in x_n is replaced by B. Let $o_0, o_1, o_2, \dots$ be observations generated pseudo-randomly such that for each i, $o_i = B$ with 25% probability and $o_i \neq B$ with 75% probability. Let $\mu(\pi, \langle\rangle) = (0, o_0)$, and for each percept-action sequence $h = x_1y_1 \dots x_n$ and action y_n, define $\mu(\pi, h \frown y_n)$ as follows (where O is the observation in x_n and $\frown$ denotes concatenation):*

$$\mu(\pi, h \frown y_n) = \begin{cases} (1, o_n) & \text{if } O = B \text{ and } y_n = A; \\ (-1, o_n) & \text{if } O = B \text{ and } y_n \neq A; \\ (-1, o_n) & \text{if } O \neq B \text{ and } \pi(h') = A; \\ (1, o_n) & \text{if } O \neq B \text{ and } \pi(h') \neq A. \end{cases}$$

Every turn in Example 4, either there is a button (25% probability) or there is not (75% probability).

- If there is a button, the agent gets $+1$ reward for pushing it, -1 reward for not pushing it.
- If there is no button, it does not matter what the agent does. The agent is rewarded or punished based on what the agent *would* do if there *was* a button. If the agent *would* push the button (if there was one), then the agent gets reward -1. Otherwise, the agent gets reward $+1$.

Thus, whenever the agent sees a button, the agent can push the button for a free reward with no consequences presently nor in the future. Nevertheless, it is in the agent's best interest to commit to never push the button! Pushing every button yields average reward $1 \cdot (.25) - 1 \cdot (.75) = -.5$ per turn. Never pushing the button yields average reward $+.5$ per turn.

The environment does not alter the true agent when it simulates the agent in order to determine what the agent would do if there was a button. If the agent's actions are based on (say) a neural net, the simulation will include a simulation of that neural net, and that simulated neural net might be altered, but the true agent's neural net is not. Thus, unless the agent itself introspects about its own hypothetical behavior ("What would

Agent	Avg Reward-per-turn $\pm$ StdErr (test repeated with 5 RNG seeds)
Q	-0.44858 ± 0.00044
DQN	-0.46687 ± 0.00137
A2C	-0.49820 ± 0.00045
PPO	-0.24217 ± 0.00793

Table 1: Performance in Example 4 (100k steps)

I do if there was a button here?"), it seems the agent would have no way of realizing that the rewards in buttonless rooms depend on said behavior. In Table 1 we see that industry-standard agents perform poorly in Example 4 (these numbers are extracted from result_table.csv in [1], which contains performance details for these agents in all the environments in our benchmark; see Sections 4 and 6 for more implementation details).

Example 4 is implemented in our open-source library as TemptingButton.py.

3.3 An interesting thought experiment

Example 5. *(Reverse history) Fix some observation O. For every percept-action sequence $h = x_1 y_1 \dots x_n$ (ending with a percept), let h' be the reverse of h. Define μ as follows:*

$$\mu(\pi, \langle\rangle) = (0, O),$$
$$\mu(\pi, h \frown y) = \begin{cases} (1, O) & \text{if } y = \pi(h'), \\ (-1, O) & \text{otherwise.} \end{cases}$$

In Example 5, at every step, μ rewards the agent iff the agent acts as it would act if history were reversed.

What would it be like to interact with the environment in Example 5? To approximate the experiment, a test subject, commanded to speak backwards, might be constantly rewarded or punished for obeying or disobeying. This might teach the test subject to imitate backward speech, but then the test subject would still act as if time were moving forward, only they would do so while performing backward-speech (they would hear their own speech backwards). But if the experimenter could perfectly simulate the test subject in order to determine what the test subject would do if time really was moving backwards, what would happen? Could test subjects learn to behave as if time was reversed[1]? Another possibility is that humans might simply not be capable of performing well in the environment. Our self-reflectiveness measure is not intended to be limited to human self-reflection levels.

We implement Example 5 as ReverseHistory.py in [1].

[1]The difference between behaving as if the incentivized experience were its experience and actually experiencing that as its real experience brings to mind the objective misalignment problem presented in [13].

3.4 Some additional examples in brief

We indicate in parentheses where the following examples are implemented in [1].

- (SelfRecognition.py) Environments which reward the agent for recognizing actions it itself would take. We implement an environment where the agent observes True-False statements like "If this observation were 0, you would take action 1," and is rewarded for deciding whether those statements are true or false.
- (IncentivizeLearningRate.py) Environments which reward the agent for behaving as if the agent were configured with a particular learning rate, suggesting extended environments can incentivize agents to learn about their own internal mechanisms, as in [22].
- (AdversarialSequencePredictor.py) Environments in which the agent competes against a competitor in an adversarial sequence prediction game [12]. This is done by outsourcing the competitor's behavior to the agent's own action-function, thus avoiding the need to hard-code a competitor into the environment.

4 Extended Environments in Practice

Definitions 1 and 2 are computationally impractical if agents are to run on environments for many steps. In this section, we will discuss a more practical implementation. Our reasons for doing this are threefold:

1. The more practical implementation makes it feasible to run industry-standard agents against our library for many steps. This is important because most industry-standard agents require many steps to learn the environments they are placed in.
2. We find it interesting in its own right how certain environments can be implemented in a practical way whereas others apparently cannot.
3. Non-determinism is effortless and natural in the practical implementation.

To practically realize extended environments, rather than passing the environment an agent, we pass the environment an agent-class which can be used to create untrained copies of the agent, called *instances* of the agent-class. Libraries like OpenAI Gym [5] and Stable Baselines3 [21] are similarly class-based: the key difference is that in our library, one must pass an agent-class to the environment-class's initiation function. The instantiated environment can use that agent-class to create copies of the agent in its internal memory. The extended environment classes in our implementation have the following methods:

- An *__init__* method, used to instantiate an individual instance of the extended environment class. This method takes an agent-class as input, which the instantiated environment can store and use to create as many independent clones of the agent as needed.

Listing 2 A practical version of Example 3.

```
class IgnoreRewards:
  def __init__(self, A):
    # Calling A() creates agent-copies. On
    # initiation, this environment stores
    # one such copy in its internal memory.
    self.sim = A()
  def start(self):
    return 0  # Initial observation 0
  def step(self, action):
    # At each step, use the stored copy
    # (self.sim) to determine how the true
    # agent would behave if all history so
    # far were the same except all rewards
    # were 0. Assumes self.sim has been
    # trained the same as the true agent,
    # except with all rewards 0.
    hypothetical_act = self.sim.act(obs=0)
    reward = 1 if action==hypothetical_act\
        else -1
    # To maintain above assumption, train
    # self.sim as if current reward were 0.
    # True agent will automatically train
    # the same way with the true reward.
    self.sim.train(o_prev=0, a=action,
        r=0, o_next=0)
    return (reward, 0)  # Observation=0
```

- A *start* method, which takes no input, and which outputs a default observation to get the agent-environment interaction started (before the agent takes its first action).
- A *step* method, which takes an action as input, and outputs a reward and observation. Class instances can store historical data internally, so there is no need to pass the entire prior history to this step method.

Agent classes are assumed to have the following methods:

- An *__init__* method, used to instantiate instances.
- An *act* method, which takes an observation and outputs an action. Instances can store information about history in internal memory, so there is no need to pass the entire prior history to this method.
- A *train* method, which takes a prior observation, an action, a reward, and a new observation. Environments which have instantiated agent-classes can use this method to train those instances in arbitrary ways, independently of how the true agent is trained, in order to probe how the true agent would hypothetically behave in counterfactual scenarios.

In Listing 2 we give a practical version of Example 3. The reason it is practical is because it maintains just one copy of the true agent, and that copy is trained incrementally. Not all extended environments (as in Definition 2) can be realized practically. Example 5 (Reverse History) apparently cannot be. The reason Example 5 is inherently impractical is because there is no way for the environment to re-use its previous work to speed up its next percept calculation. Even if the environment retained a simulated agent trained on the previous reverse history $h_0 = x_{n-1}y_{n-2}\ldots y_1x_1$, in order to compute the next percept, the environment would need to *insert* a new percept-action pair x_ny_{n-1} at the *beginning* of h_0 to get the new reverse history $h = x_ny_{n-1}\ldots y_1x_1$. There is no guarantee that the agent's actions are independent of the order in which it is trained, so a fresh new agent simulation would need to be created and trained on all of h from scratch.

This formulation generalizes the *Newcomblike environments* (or *NDPs*) of [2] (Definition 2 would too, except for being deterministic). Essentially, NDPs are environments which may base their outputs on the agent's hypothetical behavior in alternate scenarios which differ from the true history only in their most recent observation. Already that is enough to formalize a version of Newcomb's paradox [20]. When this paradox is formalized either with NDPs or extended environments, the optimal strategy becomes clear (namely, the so-called one-box strategy).

4.1 Determinacy and Semi-Determinacy

Unlike mathematical functions, class methods in the computer science sense can be non-deterministic. They can depend on random number generators (RNGs), time-of-day, global variables, etc. Our measurement strategy might not make sense for some non-deterministic agents. For example, if an agent reads and writes files on disk, then a simulation of that agent might influence the true agent (by altering said files). This suggests the following definition.

Definition 6. *An RL agent-class* Π *is* semi-deterministic *if whenever two* Π*-instances* π_1 *and* π_2 *have been instantiated within a single run of a larger computer program, and have been identically trained (within that same run), then they act identically (within that same run).*

For example, rather than invoke the RNG, Π-instances might query a read-only pool of pre-generated random numbers. Then, within the same run of a larger program, identically-trained Π-instances would act identically, even if they would not act the same as identically-trained Π-instances in a different run.

Our measurement technique—measure self-reflectiveness by running the agent through a battery of extended environments—should work well in the practical framework as long as the agents are instances of semi-determinstic agent-classes. Whenever an instance π of a semi-determinstic agent-class Π interacts with an extended environment μ, whenever μ uses a Π-instance π' to investigate the hypothetical behavior of π, the semi-determinacy of Π ensures that the behavior μ sees in π' is indeed π's hypothetical behavior.

5 Making agents more self-reflective

If we can empirically measure the self-reflection of RL agents, then we can experimentally test whether various transformations make various agents more self-reflective. We will define what we call the *reality check* transformation, intended to increase the self-reflection of certain agents. In Section 6, empirical results will suggest the it works as intended, at least for self-reflection as measured by our library.

Definition 7. *Suppose π is an agent. The* reality check *of π is the agent π_{RC} defined recursively by:*

- $\pi_{RC}(x_1y_1 \dots x_n) = \pi(x_1y_1 \dots x_n)$ *if for all* $1 \leq i < n$, $y_i = \pi_{RC}(x_1y_1 \dots x_i)$.
- $\pi_{RC}(x_1y_1 \dots x_n) = \pi(\langle x_1 \rangle)$ *otherwise.*

In response to a percept-action history, π_{RC} first verifies the history's actions are those π_{RC} would have taken. If so, π_{RC} acts as π. But if not, then π_{RC} freezes and thereafter repeats one fixed action. The act of verifying those past actions is an act of self-reflection, so it seems plausible that at least for certain agents π, π_{RC} should be more self-reflective than π.

Definition 8. *(Informal) By a* good classic agent *we mean an agent which was designed to perform well in non-extended environments, but whose designers made no attempt to make it perform well in extended environments.*

Conjecture 9. *(Informal) For most good classic agents π, π_{RC} outperforms π on average across the space of all extended environments (suitably weighted).*

We say "most" in Conjecture 9 because an agent, though designed only for non-extended environments, might accidentally already perform well in extended environments. We do not claim the reality check operation would further increase such agents' performance. We will argue for Conjecture 9's plausibility using the following theorem.

Theorem 10. *Let π be any agent.*

1. *(Alternate definition) An equivalent alternate definition of π_{RC} would be obtained by changing the condition $y_i = \pi_{RC}(x_1y_1 \dots x_i)$ into $y_i = \pi(x_1y_1 \dots x_i)$.*
2. *(Idempotence) $\pi_{RC} = (\pi_{RC})_{RC}$.*
3. *(Equivalence on genuine history) For every extended environment μ and for every odd-length initial segment $x_1y_1 \dots x_n$ of the result of π_{RC} interacting with μ, $\pi_{RC}(x_1y_1 \dots x_n) = \pi(x_1y_1 \dots x_n)$.*
4. *(Equivalence in non-extended RL) For every non-extended environment μ, the result of π_{RC} interacting with μ equals the result of π interacting with μ.*

See the Appendix for a proof. Note that part 3 shows that π_{RC} never freezes in reality (if π does not): π_{RC} merely commits to freeze in certain impossible hypothetical scenarios.

Here is why we find Conjecture 9 plausible. An extended environment chooses outputs based not only the agent's actions, but also on how the agent would hypothetically act in other scenarios. Outputs so determined would be hard to predict if the agent does not self-reflect on said hypothetical actions. But some extended environments depend, in particular, on *impossible* hypothetical scenarios where the agent takes actions the agent would never take. In *those* scenarios, π_{RC}'s actions are trivial: blind repetition of one fixed action. This in turn trivializes those extended environments' dependency on said actions, making those extended environments more predictable. And assuming π is designed to perform well in non-extended environments, presumably π, and thus (by Theorem 10 part 3), π_{RC} can take advantage of increased predictability.

For example, let π be a deterministic Q-learner and let $x_1 y_1 \dots$ be π_{RC}'s interaction with Example 3 ("Reward Agent for Ignoring Rewards"). For any particular n, the environment computes $x_{n+1} = \mu(x_1 y_1 \dots x_n y_n)$ by checking whether or not $y_n = \pi_{\text{RC}}(x'_1 y_1 \dots x'_n)$, where each x'_i is the percept x_i with reward zeroed. If so, x_{n+1}'s reward is $+1$, otherwise it is -1 (π_{RC} is rewarded to act as if all past rewards were 0). For large enough n, since π is a Q-learner, there is almost certainly some $m < n$ such that $\pi(x_1 y_1 \dots x_m) \neq \pi(x'_1 y_1 \dots x'_m)$—i.e., a Q-learner's behavior depends on past rewards. Thus by part 1 of Theorem 10, $\pi_{\text{RC}}(x_1 y_1 \dots x_n) = \pi_{\text{RC}}(\langle x_1 \rangle) = y_1$. Thus eventually the environment becomes trivial: "reward action y_1 and punish all other actions". A Q-learner, and thus (by Theorem 10 part 3) π_{RC}, would thrive in such conditions.

In our library we implement reality-check as a function taking an agent-class Π as input. It outputs an agent-class Σ. A Σ-instance σ computes actions using a Π-instance π which it initializes once and then stores. When trained, σ checks if the training data is consistent with its own action-method. If so, it trains π on that data. Else, σ freezes, thereafter ignoring future training data and repeating its first action blindly. If Π is semi-deterministic (Definition 6), it follows that Σ is too.

6 Example measurements

Based on our conviction that self-reflection is needed for good average performance across many extended environments, self-reflection can be estimated by running an agent against a battery of extended environments. Our open-source library of extended environments [1] provides 25 such extended environments, and infrastructure for measuring an agent's self-reflection by running the agent on all these environments and their opposites (the *opposite* of an environment is obtained by multiplying all rewards by -1). Opposite-environments are included to ensure that agents which ignore the environment receive score 0, at least if they are semi-deterministic (Definition 6). All environments in the library output individual rewards of either 1, -1 or 0 every step.

We used our library to measure the self-reflection of the following agents and their reality-checks:

- Random: An agent who acts randomly.
- Simple: An agent who takes the first available action that never previously resulted in punishment for the observation in question (or action 0 if no such action

Agent	Good Classic Agent?	Measure±StdErr (Original Agent)	Measure±StdErr (Reality Check)
Random		0.0000 ± 0.0000	0.0000 ± 0.0000
Simple		0.7567 ± 0.0000	0.7146 ± 0.0031
Q	✓	0.5395 ± 0.0030	0.5720 ± 0.0038
DQN	✓	0.5174 ± 0.0068	0.6072 ± 0.0057
A2C	✓	0.6145 ± 0.0061	0.6368 ± 0.0045
PPO	✓	0.0696 ± 0.0005	0.3332 ± 0.0019

Table 3: Measuring self-reflection of agents and their reality-checks

exists).

- A standard Q-learner with $\epsilon = .9$, $\alpha = .1$, $\gamma = .9$.
- DQN, A2C, and PPO (with MLP policy) from the open-source Stable Baselines3 library [21]. Default parameter values were used except for random seed and DQN's learning_starts (which we set to 1 to let DQN learn right away). Comments in [1] describe how we made these agents semi-deterministic and enabled them to run in extended environments.

We ran each agent for 100,000 steps on each environment and its opposite (repeated with 5 RNG seeds). See Table 3 for the results. See ExampleMeasurements.py in our library for replication instructions (we used a personal laptop with no GPU). The table provides evidence for Conjecture 9. That Simple performs so well is a reflection of the library's lack of sophistication. In future work, we will compose our library with others such as the Atari benchmark, e.g.: give the agent *two* Atari joysticks with identical gameplay effects, but penalize the agent for (e.g.) using a different joystick than it would have if all rewards had been zero (see Example 3). Since Simple is not a good classic agent, Conjecture 9 is not contradicted by that row of Table 3. Table 3 does *not* prove that, e.g., "A2C is more self-reflective than DQN," because "A2C" actually refers to a whole hyperparemtrized family, of which we only measured the default member.

That π_{RC} improves performance of good classic agents in Table 3 depends, of course, on which environments are tested against. One could deliberately engineer extended environments where π_{RC} underperforms, and a library of such would give π_{RC} a low measurement. For example: an environment which punishes the agent if simulations thereof appear to freeze in response to impossible percept-action sequences. But this example is, in our opinion, contrived. We conjecture that, on average, environments where π_{RC} underperforms are more contrived than those where π_{RC} overperforms. Thus, the former would be given lower weight if extended environments were suitably weighted, as in [15].

7 Conclusion

We introduced *extended environments*. When computing rewards and observations, extended environments can consider not only actions the RL agent has taken, but also actions the agent would hypothetically take in other circumstances.

An agent may find an extended environment hard to predict if the agent only considers what has actually happened, and not its own hypothetical actions in alternate scenarios. We argued that for good performance (on average) across many extended environments, an agent would need to self-reflect to some degree. Thus, we propose that a battery of benchmark extended environments could provide a way of measuring self-reflection in RL agents. We are simultaneously publishing a rudimentary open-source library [1] of extended environments as a proof-of-concept. Further work is needed to obtain a more optimal set of extended environments. For the purposes of our proof-of-concept, we preferred to focus on extended environments of particular theoretical interest. Some examples are given in Section 3.

We introduced (in Section 5) a *reality check* transform $\pi \mapsto \pi_{\mathrm{RC}}$. We conjectured (Conjecture 9) that for most *good classic agents* π (see Definition 8), π_{RC} outperforms π on average across the space of extended environments, suitably weighted. Numerical computations (in Section 6) provide empirical evidence for the conjecture.

References

[1] Anon. Extended environments. Code repository (included in supplemental file), 2022.

[2] James Henry Bell, Linda Linsefors, Caspar Oesterheld, and Joar Skalse. Reinforcement learning in Newcomblike environments. In *NeurIPS*, 2021.

[3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[4] Benjamin Beyret, José Hernández-Orallo, Lucy Cheke, Marta Halina, Murray Shanahan, and Matthew Crosby. The animal-AI environment: Training and testing animal-like artificial cognition. *Preprint*, 2019.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI gym. *Preprint*, 2016.

[6] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-Carlo tree search: A new framework for game AI. *AIIDE*, 8:216–217, 2008.

[7] François Chollet. On the measure of intelligence. *Preprint*, 2019.

[8] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[9] Vaibhav Gavane. A measure of real-time intelligence. *Journal of Artificial General Intelligence*, 4(1):31–48, 2013.

[10] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019.

[11] José Hernández-Orallo and David L Dowe. Measuring universal intelligence: Towards an anytime intelligence test. *Artificial Intelligence*, 174(18):1508–1539, 2010.

[12] Bill Hibbard. Adversarial sequence prediction. In *International Conference on Artificial General Intelligence*, pages 399–403, 2008.

[13] Evan Hubinger, Chris van Merwijk, Vladimir Mikulik, Joar Skalse, and Scott Garrabrant. Risks from learned optimization in advanced machine learning systems. *Preprint*, 2019.

[14] Marcus Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer, 2004.

[15] Shane Legg and Marcus Hutter. Universal intelligence: A definition of machine intelligence. *Minds and machines*, 17(4):391–444, 2007.

[16] Shane Legg and Joel Veness. An approximation of the universal intelligence measure. In *Algorithmic Probability and Friends: Bayesian Prediction and Artificial Intelligence*, pages 236–249. Springer, 2013.

[17] Jan Leike and Marcus Hutter. Bad universal priors and notions of optimality. In *Conference on Learning Theory*, pages 1244–1259. PMLR, 2015.

[18] Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, 2008.

[19] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in RL. *arXiv preprint arXiv:1804.03720*, 2018.

[20] Robert Nozick. Newcomb's problem and two principles of choice. In Nicholas Rescher, editor, *Essays in honor of Carl G. Hempel*, pages 114–146. Springer, 1969.

[21] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable Baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

[22] Craig Sherstan, Adam White, Marlos C Machado, and Patrick M Pilarski. Introspective agents: Confidence measures for general value functions. In *Conference on Artificial General Intelligence*, pages 258–261. Springer, 2016.

[23] Soren Sonnenburg, Mikio L Braun, Cheng Soon Ong, Samy Bengio, Leon Bottou, Geoff Holmes, Yann LeCun, Klaus-Robert Müller, Fernando Pereira, Carl Edward Rasmussen, Gunnar Rätsch, Bernhard Schölkopf, Alexander Smola, Pascal Vincent, Jason Weston, and Robert C Williamson. The need for open source software in machine learning. *Journal of Machine Learning Research*, 8:2443–2466, 2007.

[24] Roman V Yampolskiy. Detecting qualia in natural and artificial agents. *Preprint*, 2017.

A Appendix

Proof of Theorem 10. Let D be the set of all sequences $x_1y_1 \dots x_n$ (each x_i a percept, each y_i an action).

(Part 1) Define ρ on D by

- $\rho(x_1y_1 \dots x_n) = \pi(x_1y_1 \dots x_n)$ if for all $1 \leq i < n$, $y_i = \pi(x_1y_1 \dots x_i)$.
- $\rho(x_1y_1 \dots x_n) = \pi(\langle x_1 \rangle)$ otherwise.

We must show that $\rho = \pi_{\text{RC}}$. We will prove by induction that for each $x_1y_1 \dots x_n \in D$, $\rho(x_1y_1 \dots x_n) = \pi_{\text{RC}}(x_1y_1 \dots x_n)$. The base case $n = 1$ is trivial: $\rho(\langle x_1 \rangle) = \pi(\langle x_1 \rangle) = \pi_{\text{RC}}(\langle x_1 \rangle)$ since there is no i such that $1 \leq i < 1$. For the induction step, assume $n > 1$, and assume the claim holds for all shorter sequences in D.

Case 1: Assume $(*)$ for all $1 \leq i < n$, $y_i = \pi(x_1y_1 \dots x_i)$. We claim that for all $1 \leq i < n$, $y_i = \rho(x_1y_1 \dots x_i)$. To see this, choose any $1 \leq i < n$. Then for all $1 \leq j < i$, $y_j = \pi(x_1y_1 \dots x_j)$ because otherwise j would be a counterexample to $(*)$. Thus

$$\begin{aligned} \rho(x_1y_1 \dots x_i) &= \pi(x_1y_1 \dots x_i) && \text{(By definition of } \rho) \\ &= y_i, && \text{(By } *) \end{aligned}$$

proving the claim. Now, since we have proved that for all $1 \leq i < n$, $y_i = \rho(x_1y_1 \dots x_i)$, and since our induction hypothesis guarantees that each such

$$\rho(x_1y_1 \dots x_i) = \pi_{\text{RC}}(x_1y_1 \dots x_i),$$

we conclude: for all $1 \leq i < n$, we have $y_i = \pi_{\text{RC}}(x_1y_1 \dots x_i)$. Thus

$$\pi_{\text{RC}}(x_1y_1 \dots x_n) = \pi(x_1y_1 \dots x_n) = \rho(x_1y_1 \dots x_n)$$

as desired.

Case 2: Assume there is some $1 \leq i < n$ such that $y_i \neq \pi(x_1y_1 \dots x_i)$. We may choose i as small as possible. Thus, for all $1 \leq j < i$, $y_j = \pi(x_1y_1 \dots x_j)$. By similar logic as in Case 1, it follows that for all $1 \leq j < i$, $y_j = \rho(x_1y_1 \dots x_j)$. Our induction hypothesis says that for each such j, $\rho(x_1y_1 \dots x_j) = \pi_{\text{RC}}(x_1y_1 \dots x_j)$. So for all $1 \leq j < i$, $y_j = \pi_{\text{RC}}(x_1y_1 \dots x_j)$. By definition of π_{RC}, this means $\pi_{\text{RC}}(x_1y_1 \dots x_i) =$

$\pi(x_1y_1 \dots x_i)$. But $y_i \neq \pi(x_1y_1 \dots x_i)$, so therefore $y_i \neq \pi_{\text{RC}}(x_1y_1 \dots x_i)$. Thus, since $1 \leq i < n$, by definition of π_{RC}, $\pi_{\text{RC}}(x_1y_1 \dots x_n) = \pi(\langle x_1 \rangle)$. Likewise, since $1 \leq i < n$, by definition of ρ, $\rho(x_1y_1 \dots x_n) = \pi(\langle x_1 \rangle)$. So $\rho(x_1y_1 \dots x_n) = \pi_{\text{RC}}(x_1y_1 \dots x_n)$ as desired.

(Part 2) To show that each

$$\pi_{\text{RC}}(x_1y_1 \dots x_n) = (\pi_{\text{RC}})_{\text{RC}}(x_1y_1 \dots x_n),$$

we use induction on n. For the base case, this is trivial, both evaluate to $\pi(\langle x_1 \rangle)$. For the induction step, assume $n > 1$ and that the claim holds for all shorter sequences.

Case 1: $y_i = \pi_{\text{RC}}(x_1y_1 \dots x_i)$ for all $1 \leq i < n$. Then by induction, $y_i = (\pi_{\text{RC}})_{\text{RC}}(x_1y_1 \dots x_i)$ for all $1 \leq i < n$. By definition of $(\pi_{\text{RC}})_{\text{RC}}$, this means

$$(\pi_{\text{RC}})_{\text{RC}}(x_1y_1 \dots x_n) = \pi_{\text{RC}}(x_1y_1 \dots x_n),$$

as desired.

Case 2: There is some $1 \leq i < n$ such that $y_i \neq \pi_{\text{RC}}(x_1y_1 \dots x_i)$. By induction, $y_i \neq (\pi_{\text{RC}})_{\text{RC}}(x_1y_1 \dots x_i)$. Thus, $(\pi_{\text{RC}})_{\text{RC}}(x_1y_1 \dots x_n) = \pi_{\text{RC}}(\langle x_1 \rangle) = \pi(\langle x_1 \rangle)$, which equals $\pi_{\text{RC}}(x_1y_1 \dots x_n)$ since $y_i \neq \pi_{\text{RC}}(x_1y_1 \dots x_i)$ and $i < n$.

(Part 3) Let μ be an extended environment and let $x_1y_1 \dots x_n$ be an odd-length initial segment of the result of π_{RC} interacting with μ. By induction, we may assume $\pi_{\text{RC}}(x_1y_1 \dots x_i) = \pi(x_1y_1 \dots x_i)$ for all $i < n$. In other words, $y_i = \pi(x_1y_1 \dots x_i)$ for all $i < n$. By Part 1, $\pi_{\text{RC}}(x_1y_1 \dots x_n) = \pi(x_1y_1 \dots x_n)$ as desired.

(Part 4) Let μ be a non-extended environment, let $x_1y_1x_2y_2 \dots$ be the result of π interacting with μ, and let $x'_1y'_1x'_2y'_2 \dots$ be the result of π_{RC} interacting with μ. We will show by induction that each $x_n = x'_n$ and each $y_n = y'_n$. For the base case, $x_1 = x'_1 = \mu(\langle\rangle)$ (the environment's initial percept does not depend on the agent), and therefore $y_1 = \pi(\langle x_1 \rangle) = \pi(\langle x'_1 \rangle) = y'_1$. For the induction step,

$$\begin{aligned}
x_{n+1} &= \mu(x_1y_1 \dots x_ny_n) && \text{(Definition 1 part 3)} \\
&= \mu(x'_1y'_1 \dots x'_ny'_n) && \text{(By induction)} \\
&= x'_{n+1}, && \text{(Definition 1 part 3)} \\
y_{n+1} &= \pi(x_1y_1 \dots x_{n+1}) && \text{(Definition 1 part 3)} \\
&= \pi(x'_1y'_1 \dots x'_{n+1}), && \text{(Induction plus } x_{n+1} = x'_{n+1})
\end{aligned}$$

and the latter is $\pi_{\text{RC}}(x'_1y'_1 \dots x'_{n+1})$ since for all $1 \leq i < n$, $y'_i = \pi_{\text{RC}}(x'_1y'_1 \dots x'_i)$ since $x'_1y'_1 \dots$ is the result of π_{RC} interacting with μ. Finally, $\pi_{\text{RC}}(x'_1y'_1 \dots x'_{n+1})$ is y'_{n+1}, so $y_{n+1} = y'_{n+1}$. □