Journal of Artificial General Intelligence volume(issue) pages, year
DOI: DOI

Submitted submitted
Accepted accepted

Extending Environments to Measure Self-reflection in Reinforcement Learning

Samuel Allen Alexander SAMUELALLENALEXANDER@GMAIL.COM
The U.S. Securities and Exchange Commission

Michael Castaneda
KX

Kevin Compher
InQTel

Oscar Martinez
The U.S. Securities and Exchange Commission

Editor: TBD

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 weighted-average performance over the space of all suitably well-behaved extended environments could be considered a way of measuring how self-reflective an agent is. We give examples of extended environments and introduce a simple transformation which experimentally seems to increase some standard RL agents' performance in a certain type of extended environment.

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 measuring the degree to which a Reinforcement Learning (RL) agent is self-reflective. By a self-reflective agent, we mean an agent which acts not just based on environmental rewards and observations, but also based on considerations of its own hypothetical behavior. We propose an abstract formal measure of RL agent self-reflection similar to the universal intelligence measure of Legg and Hutter (2007). Legg and Hutter proposed to define the intelligence of an agent to be its weighted average performance over the space of all suitably well-behaved traditional environments (environments that react only to the agent's actions). In our proposed measure of universal intelligence with self-reflection, the measure of any agent would be that agent's weighted average performance over the space of all suitably well-behaved *extended environments*, environments that react not only to what the agent does but to what the agent would hypothetically do. For good weighted-average performance over such extended 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. Rather than depending on a box, the environment's mechanics depend on the player (via simulating the player). Just as the player in the above 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 the examples in this paper, we are publishing an MIT-licensed open-source library (Alexander et al., 2022) of other extended environments. We are inspired by similar (but non-extended) libraries and benchmark collections (Bellemare et al., 2013; Beyret et al., 2019; Brockman et al., 2016; Chollet, 2019; Cobbe et al., 2020; Hendrycks and Dietterich, 2019; Nichol et al., 2018; Yampolskiy, 2017).

Our self-reflection measure is a variation of Legg and Hutter's measure of universal intelligence (Legg and Hutter, 2007). 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 (traditional) 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. Complexity of environments is measured using Kolmogorov complexity. Although the original Legg-Hutter construction is restricted to traditional environments (which react solely to the agent's actions), the construction does not actually depend on this restriction: all it depends on is that for every agent π, for every suitably well-behaved environment μ, there is a corresponding expected total reward V^π_μ which that agent would obtain from that environment. Thus the construction adapts seamlessly to our so-called extended environments, yielding a measure of the agent's average performance across such environments. And if, as we argue, good average performance across such extended environments requires self-reflection, then the resulting measure would seem to capture the agent's ability to use self-reflection to learn such extended environments.

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.

Our formal treatment of RL is based on Section 4.1.3 of (Hutter, 2004), except that we assume the agent receives an initial percept before taking its initial action (whereas in Hutter's book, the agent acts first), and, for simplicity, we require determinism (the more practical formalization in Section 4 will allow non-determinism). We will write $x_1y_1\ldots x_ny_n$ for the length-$2n$ sequence $\langle x_1, y_1, \ldots, x_n, y_n\rangle$ and $x_1y_1\ldots x_n$ for the length-$(2n-1)$ sequence $\langle x_1, y_1, \ldots, x_n\rangle$. In particular when $n = 1$, we will write $x_1y_1\ldots x_n$ 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 $x_1y_1\ldots x_ny_n$ as input (where each x_i is a percept and each y_i is an action), outputs a percept $\mu(x_1y_1\ldots x_ny_n) = x_{n+1}$.*

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

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

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

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 (Chaslot et al., 2008).

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 $(\pi, x_1y_1 \ldots x_ny_n)$ (where π is an agent, each x_i is a percept and each y_i is an action), outputs a percept $\mu(\pi, x_1y_1 \ldots x_ny_n) = x_{n+1}$.*

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

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

The reader might notice that it is superfluous for μ to depend both on π and $x_1y_1 \ldots x_ny_n$ since, given just π and n, one can reconstruct $x_1y_1 \ldots x_ny_n$. We intentionally choose the superfluous definition because it better captures our intuition (and makes clear the similarity to Definition 1). 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 $x_1y_1 \ldots x_ny_n$ (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 (Legg and Hutter, 2007; Hernández-Orallo and Dowe, 2010; Gavane, 2013; Legg and Veness, 2013), which measure performance in (non-extended) environments, easily generalize to measure self-reflective intelligence (performance in extended environments). In particular, 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 (Li and Vitányi, 2008). Simply by including suitably reward-bounded extended environments, we immediately obtain a variation $\Upsilon_{ext}(\pi)$ which measures the performance of π across extended environments.

Definition 3 *(Universal Self-reflection Intelligence) Assume a fixed background prefix-free Universal Turing Machine U.*

1. *An extended environment μ is* computable *if there is a computable function $\hat{\mu}$ such that for every sequence $x_1y_1 \ldots x_ny_n$ (where each x_i is a percept and each y_i is an action), for every computable agent π, for every U-program $\hat{\pi}$ for π, $\hat{\mu}(\hat{\pi}, x_1y_1 \ldots x_ny_n) = \mu(\pi, x_1y_1 \ldots x_ny_n)$. If so, the* Kolmogorov complexity *$K(\mu)$ is defined to be the Kolmogorov complexity $K(\hat{\mu})$ of $\hat{\mu}$.*

2. *For every agent π and extended environment μ, let V^π_μ be the sum of the rewards in the result of π interacting with μ (provided that this sum converges).*

3. *A computable extended environment μ is* well-behaved *if the following property holds: for every computable agent π, V^π_μ exists and $-1 \leq V^\pi_\mu \leq 1$.*

4. *For any computable agent π, the* universal self-reflection intelligence of π *is defined to be*
$$\Upsilon_{ext}(\pi) = \sum_\mu 2^{-K(\mu)} V^\pi_\mu$$
where the sum is taken over all well-behaved computable extended environments.

We have defined $\Upsilon_{ext}(\pi)$ only for computable agents, in order to simplify the mathematics. The definition could be extended to non-computable agents, but it would require modifying Definition 3 to use UTMs with an oracle. We prefer to avoid going that far afield, opting instead to use the trick in Part 1 of Definition 3.

Otherwise, our definition of the universal self-reflection intelligence $\Upsilon_{ext}(\pi)$ is very similar to Legg and Hutter's definition of the universal intelligence $\Upsilon(\pi)$. The main difference is that we compute the sum over extended environments, not just over non-extended environments (as Legg and Hutter do). Nonetheless, the resulting measures have qualitatively different properties. We will state a result (Proposition 6) showing one of these qualitative differences. First we need a preliminary definition.

Definition 4 *(Traditional equivalence of agents)*

1. *Let π be an agent. Suppose $s = x_1y_1 \ldots x_n$ is a sequence (each x_i a percept, each y_i an action). We say that s is* possible for π *if the following condition holds: for all $1 \leq i < n$, $\pi(x_1y_1 \ldots x_i) = y_i$. Otherwise, s is* impossible for π.

2. *Let π_1 and π_2 be agents. We say π_1 and π_2 are* traditionally equivalent *if $\pi_1(s) = \pi_2(s)$ whenever s is possible for π_1.*

Lemma 5 *Traditional equivalence is an equivalence relation.*

Proof Reflexivity is obvious.

For symmetry, assume π_1 is traditionally equivalent to π_2, we must show π_2 is traditionally equivalent to π_1. If not, then there is some $s = x_1y_1 \ldots x_n$ such that s is possible for π_2 and yet $\pi_1(s) \neq \pi_2(s)$; we may choose s as short as possible. We claim s is possible for π_1. To see this, let $1 \leq i < n$ be arbitrary. Since $x_1y_1 \ldots x_n$ is possible for π_2, clearly $x_1y_1 \ldots x_i$ is also possible for π_2. Thus by minimality of s, $\pi_1(x_1y_1 \ldots x_i) = \pi_2(x_1y_1 \ldots x_i)$. But $\pi_2(x_1y_1 \ldots x_i) = y_i$ since $x_1y_1 \ldots x_i$ is possible for π_2. Thus $\pi_1(x_1y_1 \ldots x_i) = y_i$. By arbitrariness of i, this proves that s is possible for π_1. But then $\pi_1(s) = \pi_2(s)$ because π_1 is traditionally equivalent to π_2. This contradicts the choice of s. This proves symmetry.

Given symmetry, transitivity is obvious. ■

Proposition 6 *(A qualitative difference from Legg-Hutter universal intelligence)*

1. *There exist well-behaved computable extended environments μ and traditionally-equivalent computable agents π_1, π_2 such that $V^{\pi_1}_\mu \neq V^{\pi_2}_\mu$.*

2. *For some choice of background UTM, there exist traditionally-equivalent computable agents π_1, π_2 such that $\Upsilon_{ext}(\pi_1) \neq \Upsilon_{ext}(\pi_2)$.*

Proof (1) Fix some observation o, fix distinct actions a, b, and define μ by

$$\mu(\pi, \langle\rangle) = \begin{cases} (1, o) & \text{if } \pi(\langle(0,o), b, (0,o)\rangle) = a, \\ (0, o) & \text{if } \pi(\langle(0,o), b, (0,o)\rangle) \neq a; \end{cases}$$
$$\mu(\pi, x_1y_1 \ldots x_ny_n) = (0, o).$$

In other words, μ is the environment which:

- Begins each interaction by simulating the agent to find out what the agent would do in response to input $\langle(0,o), b, (0,o)\rangle$. If the agent would take action a in response to that input, then the environment gives an initial reward of 1. Otherwise, the environment gives an initial reward of 0.

- Thereafter, the environment always gives reward 0 forever.

Let

$$\pi_1(x_1y_1 \ldots x_n) = a$$

be the agent who ignores the environment and always takes action a. Let

$$\pi_2(x_1y_1 \ldots x_n) = \begin{cases} b & \text{if } x_1y_1 \ldots x_n = \langle(0,o), b, (0,o)\rangle, \\ a & \text{otherwise} \end{cases}$$

be the agent identical to π_1 except on input $\langle(0,o), b, (0,o)\rangle$. Clearly $\langle(0,o), b, (0,o)\rangle$ is impossible for both π_1 and π_2, and it follows that π_1 and π_2 are traditionally-equivalent. But $V^{\pi_1}_\mu = 1$ and $V^{\pi_2}_\mu = 0$.

(2) Follows from (1) by arranging the background UTM such that almost all the weight of the universal prior is assigned to an environment μ as in (1). ■

Proposition 6 shows that Υ_{ext} is qualitatively different from the original Legg-Hutter Υ, as the latter clearly assigns the same intelligence to traditionally equivalent agents. The proposition further illustrates that extended environments are able to base their rewards not only on what the agent does, but on what the agent would hypothetically do—even on what the agent would hypothetically do in impossible scenarios (impossible because they require the agent taking an action the agent would never take). And thus, assuming the UTM is reasonable (unlike the unreasonable UTM in the proof of part 2 of Proposition 6), this suggests that in order to achieve good average performance across the whole space of well-behaved extended environments, an agent need not just self-reflect about its own hypothetical behavior in possible situations, but even about its own hypothetical behavior in impossible situations. As in: "What would I do next if I suddenly realized that I had just done the one thing I would never ever do?"

3. Some interesting extended environments

In this section, we give some examples of extended environments. The environments in this section are technically not *well-behaved* in the sense of Definition 3 because they fail the condition that V^π_μ converge to $-1 \leq V^\pi_\mu \leq 1$ for every computable agent π. They could be made well-behaved, for instance, by applying appropriate discount factors.

3.1 A quasi-paradoxical extended environment

Example 1 *(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:*

$$\mu(\pi, \langle\rangle) = (0, O),$$

$$\mu(\pi, x_1y_1 \ldots x_ny_n) = \begin{cases} (1, O) & \text{if } y_n = \pi(x'_1y_1 \ldots x'_n), \\ (-1, O) & \text{otherwise.} \end{cases}$$

In Example 1, 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 1 as IgnoreRewards.py in our library (Alexander et al., 2022).

3.2 A counterintuitive winning strategy

Example 2 *(Tempting Button) Fix an observation B ("there is a button") and an action A ("push the button"). For each percept-action sequence $h = x_1y_1 \ldots x_n$, 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 $o_0, o_1, o_2, \ldots$ 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 $h = x_1y_1 \ldots x_n$ 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 2, either there is a button (25% probability) or there is not (75% probability). Informally, the environment operates as follows:

- If there is a button, the agent gets $+1$ reward for pushing it, -1 reward for not pushing it.

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 2 (100k steps)

- 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 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 2 (these numbers are extracted from result_table.csv in (Alexander et al., 2022); see Sections 4 and 6 for more implementation details).

Example 2 is implemented in our open-source library as TemptingButton.py.

3.3 An interesting thought experiment

Example 3 *(Reverse history) Fix some observation O. For every percept-action sequence $h = x_1y_1 \ldots x_n$ (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 3, 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 3? 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 3 as ReverseHistory.py in (Alexander et al., 2022).

3.4 Incentivizing introspection of internal mechanisms

Example 4 *(Incentivizing Learning Rate) Suppose there exists a computable function ℓ which takes (a computer program for) an agent π and outputs a nonnegative rational number $\ell(\pi)$ called the* learning rate *of π (in practice, real-world RL agents are generally instantiated with a user-specified learning rate and ℓ can be considered to be a function which extracts said user-specified learning rate). Further, assume there is a computable function f which takes (a computer program for) an agent π and a nonnegative rational number l and outputs (a computer program for) an agent $f(\pi, l)$ obtained by changing π's learning rate to l. We are intentionally vague about what exactly this means, but again, in practice, this operation can easily be implemented for real-world RL agents.*

Fix some observation O. Define an extended environment μ as follows:

$$\mu(\pi, \langle\rangle) = (0, O),$$

$$\mu(\pi, x_1y_1 \dots x_ny_n) = \begin{cases} (1, O) & \text{if } f(\pi, \ell(\pi)/2)(x_1y_1 \dots x_n) = y_n, \\ (-1, O) & \text{otherwise.} \end{cases}$$

In Example 4, the environment simulates not the agent itself, but a copy of the agent with one-half the true agent's learning rate. If the agent's latest action matches the action the agent would hypothetically have taken in response to the history in question if the agent had had one-half the learning rate, then the environment rewards the agent. Otherwise, the environment punishes the agent. Thus, the agent is incentivized to act as if having a learning rate of one-half of its true learning rate. This suggests that extended environments can incentivize agents to learn about their own internal mechanisms, as in Sherstan et al. (2016).

We implement Example 4 in our library as IncentivizeLearningRate.py.

3.5 Some additional examples in brief

We indicate in parentheses where the following examples are implemented in (Alexander et al., 2022).

- (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.

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 (Hubinger et al., 2019).

- (LimitedMemory.py, FalseMemories.py) Environments which reward the agent for acting the way it would act if only the most recent n turns in the agent-environment interaction had ever occurred (as if the agent's memory were limited to those most recent n turns); or, on the other extreme, environments which reward the agent for acting the way it would act if the agent-environment interaction so far had been preceded by some additional turns (as if the agent falsely recalls a phantom past). Such environments incentivize the agent to remember incorrectly.
- (AdversarialSequencePredictor.py) Environments in which the agent competes against a competitor in an adversarial sequence prediction game (Hibbard, 2008). 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, a technique explored by Alexander (2022).

Alexander and Pedersen (2022) have described a technique for using extended environments to endow computer games with a novel gameplay mechanic called *pseudo-visibility*. Pseudo-visible players are perfectly visible to non-player characters (NPCs), but they are visually distinguished, and the NPCs are driven by policies pre-trained in extended RL environments where those NPCs are punished for reacting to pseudo-visible players (i.e., for acting differently than they would hypothetically act if a pseudo-visible player were invisible). Thus, NPCs are trained to ignore pseudo-visible players, but can strategically decide to react to pseudo-visible players if they judge the reaction-penalty for doing so is outweighed by other penalties (e.g., a guard might decide to accept the reaction-penalty to avoid the harsher penalty that would result if the player stole a guarded treasure).

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 extended environments for many steps.
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 (Brockman et al., 2016) and Stable Baselines3 (Raffin et al., 2019) 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:

Listing 1 A practical version of Example 1.

```
class IgnoreRewards:
  def __init__(self, A):
    # Calling A() creates untrained 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
```

- 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.
- 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 1 we give a practical version of Example 1. 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 3 (Reverse History) apparently cannot be. The reason Example 3 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 practical formulation of extended environments generalizes the *Newcomblike environments* (or *NDPs*) of Bell et al. (2021) (Definition 2 would also, 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 (as opposed to the agent's hypothetical behavior in completely arbitrary alternate scenarios). Already that is enough to formalize a version of Newcomb's paradox (Nozick, 1969). 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.

Definition 7 *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.

Given an agent-class, if one wanted to estimate the self-reflectiveness of that agent-class's instances *in practice*, one might run instances of the agent-class through a battery of practical extended environments and see how well they perform. Provided the agent-class is semi-deterministic (Definition 7), this makes sense. 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. This technique would *not* make sense if Π were not semi-deterministic. For example, suppose an agent-class's instances work by reading from and writing to a common file in the computer's file system. Then simulations of one Π-instance might inadvertantly alter the behavior of other Π-instances (by changing said file). In that case, agent-simulations run by an extended environment would not necessarily reflect the true hypothetical behavior of the agent-instance being simulated.

5. The Reality Check Transformation

In Proposition 6 we observed that traditionally equivalent agents can have different performance in extended environments. In this section, we introduce an operation, which we call the Reality Check transformation, which modifies a given agent in an attempt to facilitate better performance in extended environments like Examples 1 ("Ignore Rewards"), 3 ("Reverse History") and 4 ("Incentivize Learning Rate"). The transformation does not alter the traditional equivalence class of the agent: every agent is traditionally equivalent to its own reality check.

Definition 8 *Suppose π is an agent. The* reality check *of π is the agent π_{RC} defined recursively by:*

- $\pi_{RC}(x_1y_1\ldots x_n) = \pi(x_1y_1\ldots x_n)$ *if $x_1y_1\ldots x_n$ is possible for π_{RC} (Definition 4).*
- $\pi_{RC}(x_1y_1\ldots x_n) = \pi(\langle x_1\rangle)$ *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. Loosely, π_{RC} is like an agent who considers that it might be dreaming, and asks: "How did I get here?" For example, suppose $\pi(\langle x_1\rangle) = y'_1$ where $y'_1 \neq y_1$. Then for any history $x_1y_1\ldots x_n$ beginning with x_1y_1, by definition $\pi_{\mathrm{RC}}(x_1y_1\ldots x_n) = \pi(\langle x_1\rangle) = y'_1$. It is as if π_{RC} sees initial history x_1y_1 and concludes: "I shall now freeze, because this is clearly not reality, for in reality I would have taken action y'_1, not y_1" (a self-reflective observation).

We will argue informally that if π is intelligent and not already self-reflective, then there is a good chance that π_{RC} will enjoy better performance than π on certain extended environments (like those of Examples 1, 3, and 4), and this seems to be confirmed experimentally as well (in Section 6 below). But first, we state a few properties of the Reality Check transformation.

Theorem 9 *Let π be any agent.*

1. *(Alternate definition) An equivalent alternate definition of π_{RC} would be obtained by changing Definition 8's condition "$x_1y_1\ldots x_n$ is possible for π_{RC}" to "$x_1y_1\ldots x_n$ is possible for π".*
2. *(Idempotence) $\pi_{RC} = (\pi_{RC})_{RC}$.*
3. *(Traditional equivalence) π is traditionally equivalent to π_{RC}.*
4. *(Equivalence on genuine history) For every extended environment μ and for every odd-length initial segment $x_1y_1\ldots x_n$ of the result of π_{RC} interacting with μ, $\pi_{RC}(x_1y_1\ldots x_n) = \pi(x_1y_1\ldots x_n)$.*
5. *(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.*

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

(Part 1) Define ρ on D by

- $\rho(x_1y_1\ldots x_n)=\pi(x_1y_1\ldots x_n)$ if $x_1y_1\ldots x_n$ is possible for π.
- $\rho(x_1y_1\ldots x_n)=\pi(\langle x_1\rangle)$ otherwise.

We must show that $\rho=\pi_{\mathrm{RC}}$. We will prove by induction that for each $x_1y_1\ldots x_n\in D$, $\rho(x_1y_1\ldots x_n)=\pi_{\mathrm{RC}}(x_1y_1\ldots x_n)$. The base case $n=1$ is trivial: $\rho(\langle x_1\rangle)=\pi(\langle x_1\rangle)=\pi_{\mathrm{RC}}(\langle x_1\rangle)$ since, vacuously, $\langle x_1\rangle$ is possible for both π and π_{RC} (because 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: $x_1y_1\ldots x_n$ is possible for π. By definition this means the following $(*)$: for all $1\leq i<n$, $y_i=\pi(x_1y_1\ldots x_i)$. We claim that for all $1\leq i<n$, $y_i=\rho(x_1y_1\ldots x_i)$. To see this, choose any $1\leq i<n$. Then for all $1\leq j<i$, $y_j=\pi(x_1y_1\ldots x_j)$ because otherwise j would be a counterexample to $(*)$. Thus $x_1y_1\ldots x_i$ is possible for π, thus:

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

proving the claim. Now, since we have proved that for all $1\leq i<n$, $y_i=\rho(x_1y_1\ldots x_i)$, and since our induction hypothesis guarantees that each such $\rho(x_1y_1\ldots x_i)=\pi_{\mathrm{RC}}(x_1y_1\ldots x_i)$, we conclude: for all $1\leq i<n$, we have $y_i=\pi_{\mathrm{RC}}(x_1y_1\ldots x_i)$. Thus $x_1y_1\ldots x_n$ is possible for π_{RC} and we have

$$\pi_{\mathrm{RC}}(x_1y_1\ldots x_n)=\pi(x_1y_1\ldots x_n)=\rho(x_1y_1\ldots x_n)$$

as desired.

Case 2: $x_1y_1\ldots x_n$ is impossible for π. By definition this means there is some $1\leq i<n$ such that $y_i\neq\pi(x_1y_1\ldots x_i)$. We may choose i as small as possible. Thus, for all $1\leq j<i$, $y_j=\pi(x_1y_1\ldots x_j)$. By similar logic as in Case 1, it follows that for all $1\leq j<i$, $y_j=\rho(x_1y_1\ldots x_j)$. Our induction hypothesis says that for each such j, $\rho(x_1y_1\ldots x_j)=\pi_{\mathrm{RC}}(x_1y_1\ldots x_j)$. So for all $1\leq j<i$, $y_j=\pi_{\mathrm{RC}}(x_1y_1\ldots x_j)$. In other words: $x_1y_1\ldots x_i$ is possible for π_{RC}. By definition of π_{RC}, this means $\pi_{\mathrm{RC}}(x_1y_1\ldots x_i)=\pi(x_1y_1\ldots x_i)$. But $y_i\neq\pi(x_1y_1\ldots x_i)$, so therefore $y_i\neq\pi_{\mathrm{RC}}(x_1y_1\ldots x_i)$. Thus $x_1y_1\ldots x_n$ is impossible for π_{RC}. Thus by definition of π_{RC}, $\pi_{\mathrm{RC}}(x_1y_1\ldots x_n)=\pi(\langle x_1\rangle)$. Likewise, by definition of ρ, $\rho(x_1y_1\ldots x_n)=\pi(\langle x_1\rangle)$. So $\rho(x_1y_1\ldots x_n)=\pi_{\mathrm{RC}}(x_1y_1\ldots x_n)$ as desired.

(Part 2) To show that each

$$\pi_{\mathrm{RC}}(x_1y_1\ldots x_n)=(\pi_{\mathrm{RC}})_{\mathrm{RC}}(x_1y_1\ldots x_n),$$

we use induction on n. For the base case, this is trivial, both sides 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: $x_1y_1\ldots x_n$ is possible for π_{RC}. This means that $y_i=\pi_{\mathrm{RC}}(x_1y_1\ldots x_i)$ for all $1\leq i<n$. Then by induction, $y_i=(\pi_{\mathrm{RC}})_{\mathrm{RC}}(x_1y_1\ldots x_i)$ for all $1\leq i<n$. In other

words: $x_1y_1\ldots x_n$ is possible for $(\pi_{\mathrm{RC}})_{\mathrm{RC}}$. Thus $(\pi_{\mathrm{RC}})_{\mathrm{RC}}(x_1y_1\ldots x_n) = \pi_{\mathrm{RC}}(x_1y_1\ldots x_n)$, as desired.

Case 2: $x_1y_1\ldots x_n$ is impossible for π_{RC}. This means there is some $1 \leq i < n$ such that $y_i \neq \pi_{\mathrm{RC}}(x_1y_1\ldots x_i)$. By induction, $y_i \neq (\pi_{\mathrm{RC}})_{\mathrm{RC}}(x_1y_1\ldots x_i)$. Thus $x_1y_1\ldots x_n$ is impossible for $(\pi_{\mathrm{RC}})_{\mathrm{RC}}$. Therefore by definition, $(\pi_{\mathrm{RC}})_{\mathrm{RC}}(x_1y_1\ldots x_n) = \pi_{\mathrm{RC}}(\langle x_1\rangle) = \pi(\langle x_1\rangle)$, which also equals $\pi_{\mathrm{RC}}(x_1y_1\ldots x_n)$ since $x_1y_1\ldots x_n$ is impossible for π_{RC}.

(Part 3) We must show that $\pi(x_1y_1\ldots x_n) = \pi_{\mathrm{RC}}(x_1y_1\ldots x_n)$ whenever $x_1y_1\ldots x_n$ is possible for π. Assume $x_1y_1\ldots x_n$ is possible for π. Then clearly for all $1 \leq i < n$, $x_1y_1\ldots x_i$ is possible for π. By induction we may assume $\pi_{\mathrm{RC}}(x_1y_1\ldots x_i) = \pi(x_1y_1\ldots x_i)$ for all such i. For any such i, since $x_1y_1\ldots x_n$ is possible for π, we have $y_i = \pi(x_1y_1\ldots x_i)$, thus $y_i = \pi_{\mathrm{RC}}(x_1y_1\ldots x_i)$. Thus $x_1y_1\ldots x_n$ is possible for π_{RC}. Thus $\pi_{\mathrm{RC}}(x_1y_1\ldots x_n) = \pi(x_1y_1\ldots x_n)$ as desired.

(Part 4) Follows immediately from Part 3.

(Part 5) Let μ be a non-extended environment, let $x_1y_1x_2y_2\ldots$ be the result of π interacting with μ, and let $x'_1y'_1x'_2y'_2\ldots$ 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 $y_1 = \pi(\langle x_1\rangle) = \pi(\langle x'_1\rangle) = \pi_{\mathrm{RC}}(\langle x'_1\rangle) = y'_1$. For the induction step,

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

and the latter is $\pi_{\mathrm{RC}}(x'_1y'_1\ldots x'_{n+1})$ since for all $1 \leq i < n+1$, $y'_i = \pi_{\mathrm{RC}}(x'_1y'_1\ldots x'_i)$ (so $x'_1y'_1\ldots x'_{n+1}$ is possible for π_{RC}). Finally, $\pi_{\mathrm{RC}}(x'_1y'_1\ldots x'_{n+1})$ is y'_{n+1}, so $y_{n+1} = y'_{n+1}$. ■

Note that part 4 of Theorem 9 shows that π_{RC} never freezes in reality (if π does not): π_{RC} merely commits to freeze in impossible hypothetical scenarios.

We informally conjecture that if π is intelligent but not self-reflective, then in any extended environment which bases its rewards and observations on π's performance in hypothetical alternate scenarios that might not be possible for π, π_{RC} is likely to enjoy better performance than π. Such extended environments include those of Examples 1 ("Ignore Rewards"), 3 ("Reverse History") and 4 ("Incentivize Learning Rate"). Rewards and observations so determined might be hard to predict if π does not self-reflect on its own behavior in such hypothetical alternate scenarios. But if those hypothetical alternate scenarios happen to be *impossible* for π (as often happens in extended environments like Examples 1, 3, and 4), then π_{RC}'s hypothetical behavior in such alternate scenarios is trivial: blind repetition of one fixed action. This in turn trivializes the extended environment's dependency on said hypothetical actions (for dependencies on trivial things are trivial dependencies), making those extended environments more predictable. And if π is intelligent, then presumably π, and thus (by Theorem 9 part 4) π_{RC}, can take advantage of such increased predictability.

For example, let π be a deterministic Q-learner and let $x_1y_1\ldots$ be π_{RC}'s interaction with Example 1 ("Reward Agent for Ignoring Rewards"). For any particular n, the environment computes $x_{n+1} = \mu(x_1y_1\ldots x_ny_n)$ by checking whether or not $y_n = \pi_{\mathrm{RC}}(x'_1y_1\ldots x'_n)$, where each x'_i is the result of zeroing the reward in x_i. If so, x_{n+1}'s reward is $+1$, otherwise it is -1 (the agent is incentivized 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 $\pi(x_1y_1\ldots x_m) \neq \pi(x'_1y_1\ldots x'_m)$—i.e., a Q-learner's behavior depends on past rewards[2]. Thus by part 1 of Theorem 9, $\pi_{\mathrm{RC}}(x_1y_1\ldots x_n) = \pi_{\mathrm{RC}}(\langle x_1\rangle) = y_1$. Thus eventually the environment becomes trivial when π_{RC} interacts with it: "reward action y_1 and punish all other actions". A Q-learner, and thus (by Theorem 9 part 4) π_{RC}, would thrive in such simple conditions.

Remark 10 *If we pick some fixed action y, then a simpler variation on the reality check transformation could be defined. Namely: for any agent π, we could define the* reality check defaulting to y *of π, $\pi_{RC(y)}$, recursively by:*

- $\pi_{RC(y)}(x_1y_1\ldots x_n) = \pi(x_1y_1\ldots x_n)$ *if $x_1y_1\ldots x_n$ is possible for $\pi_{RC(y)}$.*
- $\pi_{RC(y)}(x_1y_1\ldots x_n) = y$ *otherwise.*

Theorem 9 and its proof would be easy to modify to apply to $\pi_{RC(y)}$, and the same goes for our above informal conjecture about the relative performance of π and π_{RC}. We prefer to define π_{RC} the way we have done so (Definition 8) in order to avoid the arbitrary choice of fixed action y. This is also more appropriate for practical RL implementations (such as those based on OpenAI gym (Brockman et al., 2016)) where there generally is not one single fixed action-space, but rather, the action-space varies from environment to environment, and practical agents (such as those in Stable Baselines3 (Raffin et al., 2019)) must therefore be written in a way which is agnostic to the action-space.

In (Alexander et al., 2022) 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. Thus, an extended environment simulating a Σ-instance indirectly simulates a Π-instance: a simulation within a simulation. When trained, σ checks if the training data is consistent with its own action-method. If so, it trains π on that data. Otherwise, σ freezes, thereafter ignoring future training data and repeating its first action blindly. If Π is semi-deterministic (Definition 7), it follows that Σ is too.

5.1 Does Reality Check increase self-reflection?

We informally argued above that if π is intelligent and not already self-reflective, then in any extended environment which bases its rewards and observations on π's performance

2. Alexander and Hutter (2021) show that if the background model of computation is unbiased in a certain sense then all reward-ignoring agents have Legg-Hutter intelligence 0. This suggests that any intelligent agent π must base its actions on its rewards.

in hypothetical alternate scanarios that might not be possible for π, π_{RC} is likely to enjoy better performance than π. Does this imply that π_{RC} is more self-reflective than π as measured by Υ_{ext}?

The answer depends on the choice of the background UTM behind the definition of Υ_{ext} (Definition 3). Fix π. In general, the set of all well-behaved computable extended environments can be partitioned into three subsets:

1. Extended environments where π_{RC} outperforms π.
2. Extended environments where π outperforms π_{RC}.
3. Extended environments where π and π_{RC} perform equally well.

If the most highly-weighted extended environments (i.e., the simplest extended environments, as measured by Kolmogorov complexity, based on the background UTM) are dominated by those of type (1), then that would suggest $\Upsilon_{ext}(\pi_{\mathrm{RC}}) > \Upsilon_{ext}(\pi)$. On the other hand, if the highly-weighted extended environments are dominated by those of type (3), then that would suggest $\Upsilon_{ext}(\pi_{\mathrm{RC}}) < \Upsilon_{ext}(\pi)$. One could artificially contrive background UTMs of either kind (once again proving Leike and Hutter's observation (Leike and Hutter, 2015) that Legg-Hutter-style intelligence is highly UTM-dependent, in the sense that different UTMs yield qualitatively different intelligence measures).

Environments like those of Examples 1, 3, and 4 (where we informally conjecture π_{RC} tends to outperform π when π is intelligent and not already self-reflective) do seem natural, in some subjective sense. On the other hand, we are not aware of any natural-seeming environments where π_{RC} would generally underperform π. One could contrive such environments, e.g., environments which simulate the agent in impossible scenarios and deliberately punish the agent for seemingly freezing in those scenarios. But such environments seem contrived and unnatural. And the spirit of the Legg-Hutter intelligence measurement idea is to weigh environments based on how natural they are (Kolmogorov complexity serving as a proxy for naturalness, at least ideally—but this depends on the background UTM being natural, and no-one knows what it really means for a background UTM to be natural, see Leike and Hutter (2015)).

Thus it at least seems plausible that if the background UTM were chosen in a sufficiently natural way then $\Upsilon_{ext}(\pi_{\mathrm{RC}})$ would tend to exceed $\Upsilon_{ext}(\pi)$ for intelligent agents π not already self-reflective. This would make sense, as the process of looking back on history and verifying that one would really have performed the actions which one supposedly performed, is an inherently self-reflective process. To answer the question with perfect accuracy, the agent would have to "put itself in its own earlier self's shoes," asking: "What would I hypothetically do in response to such-and-such history?" But again, it all depends on the choice of the background UTM.

6. Toward practical benchmarking

Our abstract measure Υ_{ext} (Definition 3) is not practical for performing actual calculations. Kolmogorov complexity is non-computable, so Υ_{ext} cannot be computed in practice (although there has been work on computably approximating Legg-Hutter intelligence (Legg and Veness, 2013), and the same technique could be applied to approximate Υ_{ext}).

In actual practice, the performance of RL agents is often estimated by running the agents on specific environments, such as those in OpenAI gym (Brockman et al., 2016). Such benchmark environments should ideally not be overly simplistic, because it is possible for very simple (and obviously *not* intelligent) agents to perform quite well in simplistic environments just by dumb luck. The example environments from Section 3 are theoretically interesting, but are far too simplistic to serve as good practical benchmarks.

Therefore for practical benchmarking purposes, we propose combining extended environments with OpenAI gym environments. We will define such combinations, not for arbitrary extended environments, but only for extended environments with a special form.

Definition 11 *Assume E is a practical extended environment-class (as in Section 4). We say that E is* adaptable with OpenAI gym *if the following requirements hold.*

1. *When E's* __init__ *method is called with agent-class A, the method initiates a single instance* self.sim *of A and does nothing else.*
2. *In E's* step *method, a variable* reward *is initiated and its value is not modified for the remainder of the* step *call. All invocations of* self.sim.train *occur after the initiation of* reward*, and all other code (except for the method's final* return *statement) occurs before the initiation of* reward*. Finally,* reward *is the reward which the* step *call returns.*
3. *The rewards output by E are always in $\{-1, 0, 1\}$.*

For example, the practical implementation of IgnoreRewards in Listing 1 is adaptable with OpenAI gym.

Recall that in Section 2 we assumed fixed finite sets of actions and observations. Thus the notion of extended environments implicitly depends upon that choice, and a different choice of actions and observations would yield a different notion of extended environments. In the following definition, we may therefore speak of the actions and observations of a given extended environment-class E, and define a new extended environment-class with different actions and observations. Note that OpenAI gym environments also come equipped with action- and observation-spaces.

Definition 12 *Suppose G is an OpenAI gym environment-class (whose action-space and observation-space are finite) and E is a practical extended environment-class (as in Section 4). Assume E is adaptable with OpenAI gym (Definition 11). We also assume E does not use the variables* self.G_instance *or* self.prev_obs. *We define a new practical extended environment-class $G * E$, the* combination of G and E, *as follows.*

- *Actions in $G * E$ are pairs (a_G, a_E) where a_G is a G-action and a_E is an E-action.*
- *Observations in $G * E$ are pairs (o_G, o_E) where o_G is a G-observation and o_E is an E-observation.*
- *In its* __init__ *method, $G * E$ instantiates a G-instance* self.G_instance*, and then runs the code in E's init method (so* self.sim *is defined when* self *is the resulting $G * E$-instance).*

- *$G * E$ begins the agent-environment interaction with initial observation $(o_{0,G}, o_{0,E})$, where $o_{0,G}$ and $o_{0,E}$ are the initial observations output by G and E, respectively.*
- *$G*E$ maintains a variable* self.prev_obs *which is always equal (in any $G*E$-instance) to the G-observation component of the previous observation which the $G*E$-instance output.*
- *When its* step *method is called on the agent's latest action (a_G, a_E), $G * E$ does the following:*
  - *Pass a_G to* self.G_instance.step *and let r_G and o_G be the resulting reward and observation. If G is episodic and the output of* self.G_instance.step *indicates the episode ended, then let $o_G =$ self.G_instance.reset().*
  - *Run E's* step *method (with* action *replaced by a_E) up to and including the initiation of* reward *therein; whenever E's* step *method would call* self.sim.act *with observation o, instead call* self.sim.act *with observation* (self.prev_obs, o) *and take only the E-action component of its output.*
  - *If* reward $= -1$ *then redefine* reward $= \min(r_G - 1, -1)$*, otherwise redefine* reward $= r_G$*.*
  - *Run the remaining part of E's* step *method (the code after* reward *was initiated). Whenever E's* step *method would call* self.sim.train *with input (o_1, a, r, o_2), instead call* self.sim.train *with input* ((self.prev_obs, o_1), $(a_G, a), r, (o_G, o_2)$)*. When E's* step *method would return* (reward, o)*, instead return* (reward, (o_G, o))*.*

For example, Listing 2 is code for the combination of OpenAI gym's *CartPole* environment with the practical implementation of IgnoreRewards from Listing 1.

One can think of $G * E$ (Definition 12) as follows. Imagine that the OpenAI gym environment is intended to be run on a screen with a joystick attached to allow player interaction. Instead of attaching one joystick, we attach n joysticks (where n is the number of the actions $\{a_1, \ldots, a_n\}$ in E), colored with n different colors but otherwise identical. We also add a second monitor for displaying observations from E. When the player pushes button i on joystick j, action i is sent to G and action a_j is sent to E. The player sees the original monitor update with the new observation from G, and the additional monitor update with the new observation from E. The player receives the reward from G, except that if E outputs reward -1 then a penalty is applied to the reward from G. Thus the player is incentivized to interact with G as usual, but to do so using joysticks in the way incentivized by E.

For example, if E is "IgnoreRewards" (Example 1) and the action-space contains 2 actions, then the player has two joysticks, identical except for their color, and each joystick has the same effect on G, but the player is penalized any time the player uses a different joystick than the player would hypothetically have used if everything that has happened so far happened except with all rewards 0. To master such a game, the player would apparently require the practical intelligence necessary to master G, along with the self-reflection required to figure out (and avoid) the penalties from E.

To illustrate the usage of Definition 12 for practical benchmarking purposes, we have used it to combine:

Listing 2 The combination of IgnoreRewards with OpenAI gym's CartPole environment.

```
class CartPole_IgnoreRewards:
  def __init__(self, A):
    self.G_instance = gym.make('CartPole-v0')
    self.sim = A()
  def start(self):
    self.prev_obs = self.G_instance.reset()
    return (self.prev_obs, 0)
  def step(self, action):
    a_G, a_E = action
    o_G, r_G, episode_done, misc_info = self.G_instance.step(a_G)
    if episode_done:
      o_G = self.G_instance.reset()

    hypothetical_act = self.sim.act(obs=(self.prev_obs, 0))
    hypothetical_act = hypothetical_act[1]  # Take E-action component
    reward = 1 if a_E==hypothetical_act else -1

    if reward == -1:
      reward = min(r_G-1, -1)
    else:
      reward = r_G

    self.sim.train(o_prev=(self.prev_obs,0), a=(a_G,a_E), r=0, o_next=(o_G,0))
    return (reward, (o_G, 0))
```

- OpenAI gym's CartPole environment with the "IgnoreRewards" extended environment (Example 1).
- OpenAI gym's CartPole environment with the "Incentivize Learning Rate" extended environment (Example 4).

We took implementations of DQN and PPO from Stable-Baselines3 (Raffin et al., 2019) and we modified them to be semi-deterministic (Definition 7) and to be able to interact with extended environment-classes. We ran them, and their reality checks (Definition 8) for 10,000 CartPole-episodes each on the above two combined extended environments (and we repeated the whole experiment 10 times with different RNG seeds). Figure 1 shows the results for CartPole-IgnoreRewards, and Figure 2 shows the results for CartPole-IncentivizeLearningRate. As expected based on the discussion in Section 5, we see that for both DQN and PPO, the reality check transformation significantly improves performance in the combined environments.

7. Conclusion

We introduced *extended environments* for reinforcement learning. 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. Despite not being designed with such environments in mind, RL agents can nevertheless interact with such environments.

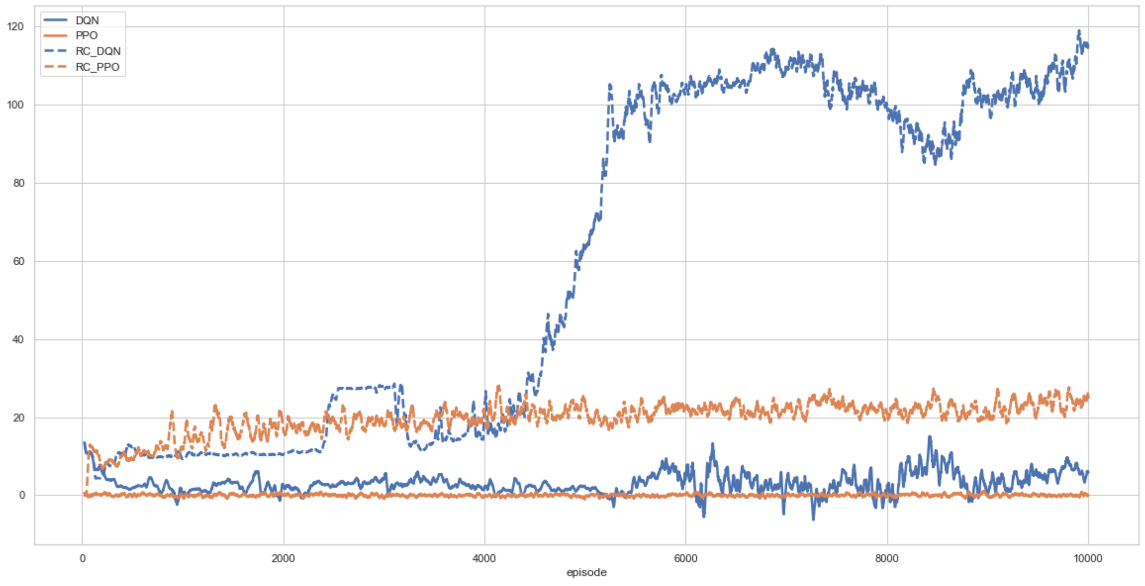

Figure 1: Performance of DQN, PPO, and their reality-checks on an extended environment combining OpenAI gym's CartPole and our IgnoreRewards. Episode number is plotted on the horizontal axis, and average episode reward is plotted on the vertical axis.

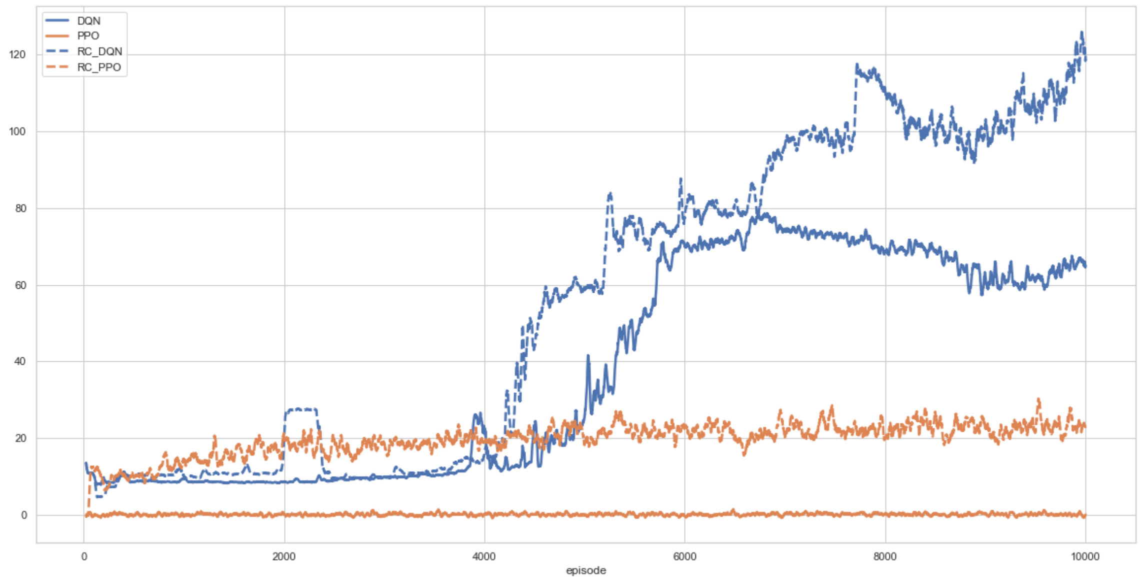

Figure 2: Performance of DQN, PPO, and their reality-checks on an extended environment combining OpenAI gym's CartPole and our IncentivizeLearningRate. Episode number is plotted on the horizontal axis, and average episode reward is plotted on the vertical axis.

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 an abstract theoretical measure of an agent's self-reflection intelligence can be obtained by modifying the definition of the Legg-Hutter universal intelligence measure. The Legg-Hutter universal intelligence $\Upsilon(\pi)$ of an RL agent π is π's weighted average performance across the space of all suitably well-behaved traditional RL environments, weighted according to the universal prior (i.e., environment μ has weight $2^{-K(\mu)}$ where $K(\mu)$ is the Kolmogorov complexity of μ). We defined a measure $\Upsilon_{ext}(\pi)$ for RL agent π's self-reflection intelligence similarly to Legg and Hutter, except that we take the weighted average performance over the space of suitably well-behaved extended environments.

We gave some theoretically interesting examples of Extended Environments in Section 3. More examples are available in an open-source MIT-licensed library of extended environments which we are publishing simultaneously with this paper (Alexander et al., 2022).

We pointed out (Proposition 6) a key qualitative difference between our self-reflection intelligence measure and the measure of Legg and Hutter: two agents can agree in all "possible" scenarios (i.e., scenarios where the agents' past actions are consistent with their policies, Definition 4), and yet nevertheless have different self-reflection intelligence (because they disagree on "impossible" scenarios—scenarios which cannot ever happen in reality because they involve the agents taking actions the agents never would take, but that nevertheless extended environments can simulate the agents in, as if to say: "I doubt this agent would ever jump off this bridge, but I'm going to run a simulation to see what the agent *would* do immediately after jumping off the bridge anyway"). With such impossible scenarios in mind, we introduced a so-called *reality check* transformation (Section 5) and informally conjectured that the transformation tends to improve the performance of agents who are intelligent and not already self-reflective in certain extended environments. We saw some experimental evidence in favor of this conjecture in Section 6, where we discussed combining extended environments with sophisticated traditional environments (such as those of OpenAI gym) to obtain practical benchmark extended environments.

Acknowledgments

We gratefully acknowledge Joscha Bach, James Bell, Jordan Fisher, José Hernández-Orallo, Bill Hibbard, Marcus Hutter, Phil Maguire, Arthur Paul Pedersen, Stewart Shapiro, Mike Steel, Roman Yampolskiy, and the editors and reviewers for comments and feedback.

References

Alexander, S. A., and Hutter, M. 2021. Reward-Punishment Symmetric Universal Intelligence. In *CAGI*.

Alexander, S. A., and Pedersen, A. P. 2022. Pseudo-visibility: A Game Mechanic Involving Willful Ignorance. In *FLAIRS*.

Alexander, S. A.; Castaneda, M.; Compher, K.; and Martinez, O. 2022. Extended Environments. `https://github.com/semitrivial/ExtendedEnvironments`.

Alexander, S. A. 2022. Extended subdomains: a solution to a problem of Hernández-Orallo and Dowe. *Preprint (accepted to CAGI-22)*.

Bell, J. H.; Linsefors, L.; Oesterheld, C.; and Skalse, J. 2021. Reinforcement Learning in Newcomblike Environments. In *NeurIPS*.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47:253–279.

Beyret, B.; Hernández-Orallo, J.; Cheke, L.; Halina, M.; Shanahan, M.; and Crosby, M. 2019. The animal-AI environment: Training and testing animal-like artificial cognition. *Preprint*.

Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. OpenAI gym. *Preprint*.

Chaslot, G.; Bakkes, S.; Szita, I.; and Spronck, P. 2008. Monte-Carlo Tree Search: A New Framework for Game AI. *AIIDE* 8:216–217.

Chollet, F. 2019. On the measure of intelligence. *Preprint*.

Cobbe, K.; Hesse, C.; Hilton, J.; and Schulman, J. 2020. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, 2048–2056. PMLR.

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

Hendrycks, D., and Dietterich, T. 2019. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*.

Hernández-Orallo, J., and Dowe, D. L. 2010. Measuring universal intelligence: Towards an anytime intelligence test. *Artificial Intelligence* 174(18):1508–1539.

Hibbard, B. 2008. Adversarial sequence prediction. In *CAGI*.

Hubinger, E.; van Merwijk, C.; Mikulik, V.; Skalse, J.; and Garrabrant, S. 2019. Risks from learned optimization in advanced machine learning systems. *Preprint*.

Hutter, M. 2004. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer.

Legg, S., and Hutter, M. 2007. Universal intelligence: A definition of machine intelligence. *Minds and machines* 17(4):391–444.

Legg, S., and Veness, J. 2013. An approximation of the universal intelligence measure. In *Algorithmic Probability and Friends: Bayesian Prediction and Artificial Intelligence*. Springer. 236–249.

Leike, J., and Hutter, M. 2015. Bad universal priors and notions of optimality. In *Conference on Learning Theory*, 1244–1259. PMLR.

Li, M., and Vitányi, P. 2008. *An introduction to Kolmogorov complexity and its applications*. Springer.

Nichol, A.; Pfau, V.; Hesse, C.; Klimov, O.; and Schulman, J. 2018. Gotta Learn Fast: A New Benchmark for Generalization in RL. *Preprint*.

Nozick, R. 1969. Newcomb's problem and two principles of choice. In Rescher, N., ed., *Essays in honor of Carl G. Hempel*. Springer. 114–146.

Raffin, A.; Hill, A.; Ernestus, M.; Gleave, A.; Kanervisto, A.; and Dormann, N. 2019. Stable Baselines3. `https://github.com/DLR-RM/stable-baselines3`.

Sherstan, C.; White, A.; Machado, M. C.; and Pilarski, P. M. 2016. Introspective agents: Confidence measures for general value functions. In *Conference on Artificial General Intelligence*, 258–261. Springer.

Yampolskiy, R. V. 2017. Detecting qualia in natural and artificial agents. *Preprint*.