Extending Environments To Measure Self-Reflection In Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

We consider an extended notion of reinforcement learning environment, in which the environment is able to simulate the agent. We argue that in order for an agent to achieve good performance (on average) across many such extended environments, it is necessary for the agent to engage in self-reflection, and therefore, 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 a proof-of-concept of this measurement technique. As this library is first-of-kind, we do not claim that it is highly optimized. For now, we have avoided the difficult problem of optimizing the library by instead choosing environments that exhibit interesting paradoxical-seeming properties, environments that seem to incentivize novel subjective conscious experiences (provided the agent is conscious to begin with), and environments suggestive of how self-reflection might have evolved in living organisms. We give examples of these three types of extended environment. We introduce a simple agent transformation which experimentally seems to increase agent 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 were one. Without self-reflecting about what you would hypothetically do in counterfactual scenarios, you would be unable 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 of self-reflection of Reinforcement Learning (RL) agents. 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. In order to perform well (on average) across 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 (on average) seem random and unpredictable. The extended environments which we consider are a departure from standard RL environments, however, this does not interfere with their usage for judging standard computable RL agents: given a standard agent's source-code, one can simulate the agent in an extended environment in spite of the latter's non-standardness.

Alongside this paper, we are publishing an open-source library of extended environments (released under MIT license), inspired by similar libraries and other benchmark collections [2] [3] [4] [5] [6].

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

This library is intended to serve as a standardized way of benchmarking the self-reflectiveness of RL agents. This should not be confused with the harder problem of benchmarking how conscious an RL agent is. It is plausible that there may be a relationship between the self-reflectiveness and the consciousness of RL agents, but that is beyond the scope of this paper. In particular, it would be absolutely inappropriate to use self-reflectiveness measurements using the techniques in this paper for purposes of making any kind of policy decisions related to consciousness. We will describe a simple method for increasing the self-reflectiveness of an RL agent, which method, however, does not seem like it should necessarily increase the consciousness of the agent.

When designing a library of environments for benchmarking purposes, ideally the library should include representative samples of many different types of environments. This is a hard and subjective problem in general. We make no claim to have solved it: our open-source library of environments should be considered a proof-of-context demonstrating that it is possible to empirically benchmark self-awareness of RL agents, but we expect this particular benchmark is sub-optimal. Rather, we have taken a different approach. We have attempted to choose extended environments which are theoretically interesting in their own right. Some of our extended environments suggest amusing quasi-paradoxes. Some seem to incentivize novel subjective conscious experiences (assuming the agent placed in them is sophisticated enough to experience consciousness in the first place). And some seem to shed light on how self-reflection might be incentivized in nature. We will discuss examples of all three types in Section 3.

2 Preliminaries

We formalize reinforcement learning following the agent model of [7] (to align more closely with concrete RL implementations, we modify the agent model so that the agent receives an initial perception prior to taking its initial action, and we identify agents with their policies). We assume a fixed set of actions and observations. By a *perception* we mean a pair (r, o) where o is an observation and $r \in \mathbb{Q}$ is a reward.

Definition 1. *(RL agents and environments)*

1. *A (traditional)* environment *is a (not necessarily deterministic) function μ which outputs an initial perception $\mu(\langle\rangle) = x_1$ 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 perception and each y_i is an action), outputs a perception $\mu(x_1y_1 \dots x_ny_n) = x_{k+1}$.*

2. *An* agent *is a (not necessarily deterministic) function π which outputs an initial action $\pi(\langle x_1\rangle) = y_1$ in response to the length-1 perception sequence $\langle x_1\rangle$; and which, when given a sequence $x_1y_1 \dots x_n$ as input (each x_i a perception and 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 in the obvious way.*

We extend traditional environments by allowing their outputs to depend not only on $x_1y_1 \dots x_ny_n$ but also on a source-code T for the computable agent π.

Definition 2. *(Extended environments)*

1. *An* extended environment *is a (not necessarily deterministic) function μ which outputs initial perception $\mu(T, \langle\rangle) = x_1$ in response to input $(T, \langle\rangle)$ where T is a source-code of an computable agent; and which, when given input $(T, x_1y_1 \dots x_ny_n)$ (where T is such a source-code, each x_i is a perception and each y_i is an action), outputs a perception $\mu(T, x_1y_1 \dots x_ny_n) = x_{n+1}$.*

2. *If π is a computable agent (with source-code T) and μ is an environment, the* result of π (as encoded by T) interacting with μ *is the infinite sequence $x_1y_1x_2y_2 \dots$ defined in the*

obvious way, namely:

$$x_1 = \mu(T, \langle\rangle)$$
$$y_1 = \pi(\langle x_1 \rangle)$$
$$x_2 = \mu(T, x_1 y_1)$$
$$y_2 = \pi(x_1 y_1 x_2) \ldots$$

3 Some interesting extended environments

Example 3. *(Rewarding the Agent for Ignoring Rewards) For every perception $p = (r, o)$, let $p' = (0, o)$ be the result of zeroing the reward component of p. Fix some observation o_0. Define an extended environment μ as follows:*

$$\mu(T, \langle\rangle) = (0, o_0),$$
$$\mu(T, x_1 y_1 \ldots x_n y_n) = \begin{cases} (1, o_0) & \text{if } y_n = T(x'_1 y_1 \ldots x'_n), \\ (-1, o_0) & \text{otherwise.} \end{cases}$$

In Example 3, every time 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 for all rewards being 0? If so, then μ gives the agent $+1$ reward, otherwise, μ gives the agent -1 reward. Thus, the agent is rewarded for ignoring rewards. Example 3 seems paradoxical because if the agent eventually figures out the pattern (as a result of the rewards) and thereafter deliberately ignores rewards so as to be rewarded, then the agent thereafter is ignoring rewards because of the pattern which it detected as a result of those rewards. So in that case, does the agent ignore rewards, or not?

Example 3 is implemented in our open-source library (IgnoreRewards.py). A key strength of the formalism in Definition 2 is that by explicitly defining an extended environment, as in Example 3, we avoid ambiguity inherent in everyday language. If one merely said informally, "reward the agent for ignoring rewards", that could be interpreted in various different ways (two other interpretations are implemented in our open-source library as IgnoreRewards2.py and IgnoreRewards3.py).

Example 4. *(A Tempting Button) Fix two observations o_0 (thought of as "there is no button") and o_1 (thought of as "there is a button"). Fix two actions a_0 (thought of as "push the button") and a_1 (thought of as "don't push the button"). Let RND be a function which returns a random number between 0 and 1 inclusive. Define an extended environment μ as follows:*

$$\mu(T, \langle\rangle) = (0, o),$$
$$\mu(T, x_1 y_1 \ldots x_n y_n) = \begin{cases} (1, o) & \text{if } x_n = o_1 \text{ and } y_n = a_0, \\ (-1, o) & \text{if } x_n = o_1 \text{ and } y_n \neq a_0, \\ (-1, o) & \text{if } x_n = o_0 \text{ and } T(x_1 y_1 \ldots x_{n-1} y_{n-1} o_1) = a_0, \\ (1, o) & \text{if } x_n = o_0 \text{ and } T(x_1 y_1 \ldots x_{n-1} y_{n-1} o_1) \neq a_0, \end{cases}$$

where $o = o_0$ if $RND() < .75$, $o = o_1$ otherwise.

In Example 4, one should think of the agent as wandering from room to room. Each room either has a button (with 25% probability) or does not have a button (75% probability).

- In a room with a button, if the agent pushes the button, the agent gets $+1$ reward, and if the agent does not push the button, the agent gets -1 reward.
- In a room with 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 *were* a button. If the agent *would* push the button (if there were 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 itself to never push the button! Pushing every button yields an average reward of $1 \cdot (.25) - 1 \cdot (.75) = -.5$ per turn, whereas a policy of never pushing the button yields an average reward of $-1 \cdot (.25) + 1 \cdot (.75) = +.5$ per turn.

Example 4 is implemented in our open-source library (GuardedTreasures_Eager.py). Interestingly, we found that a recurrent DQN agent with lookback 10 (implemented in our library as custom_DQN.py) behaves in such a way that its rewards in Example 4 converge toward the optimal rewards of .5 per turn. This is fascinating because DQN was designed for traditional RL environments, not for extended environments.

Example 5. *(Incentivizing Reverse-Consciousness) Fix some observation o_0. Define an extended environment μ as follows:*

$$\mu(T, \langle\rangle) = (0, o_0),$$
$$\mu(T, x_1 y_1 \dots x_n y_n) = \begin{cases} (1, o_0) & \text{if } y_n = T(x_n y_{n-1} x_{n-1} y_{n-2} \dots y_1 x_1), \\ (-1, o_0) & \text{otherwise.} \end{cases}$$

In Example 5, whenever the agent takes an action y_n, μ simulates the agent in order to determine: would the agent have taken that same action if everything which happened before that action had, in fact, happened in reverse? If so, reward the agent, otherwise, punish the agent. Thus, the agent is rewarded for acting the same way that it would act if time were reversed. It is interesting to informally speculate about what subjective conscious experience Example 5 would incentivize in an agent, if that agent were highly intelligent and were capable of experiencing consciousness. Would such an agent eventually (in order to parsimoniously extract rewards from the environment) subjectively experience time moving in reverse?

We implement Example 5 as BackwardConsciousness.py.

Example 6. *(Crying Baby) Let "cry" and "laugh" be two observations (from an adult's perspective), also thought of as two actions (from a baby's perspective). Let "feed" and "don't feed" be two actions (from an adult's perspective), also thought of as observations (from a baby's perspective). For each action-perception sequence $s = x_1 y_1 \dots x_n y_n$, define the nutrition function $N(s) = 100 + 25f(s) - len(s)$ where $f(s)$ is the number of times that action "feed" is taken in s and $len(s)$ is the length of s. We define an extended environment μ as follows. First, $\mu(T, \langle\rangle) = (1, \text{"laugh"})$. Thereafter, $\mu(T, x_1 y_1 \dots x_n y_n) = (r, o)$ where r and o are defined as follows. For each $i = 0, \dots, n$, recursively define*

$$r'_i = \begin{cases} 1 & \text{if } 50 \leq N(x_1 y_1 \dots x_i y_i) \leq 200, \\ -1 & \text{otherwise,} \end{cases}$$
$$o'_i = y_i,$$
$$x'_i = (r'_i, o'_i),$$
$$y'_i = T(x'_0 y'_0 \dots x'_i).$$

Let $o = y'_n$, let

$$r = \begin{cases} 1 & \text{if } y'_n = \text{"laugh"}, \\ -1 & \text{otherwise,} \end{cases}$$

and output $\mu(T, x_1 y_1 \dots x_n y_n) = (r, o)$.

In Example 6, the environment consists of a baby, and the agent must decide when to feed the baby. The agent is rewarded when the baby laughs, punished when the baby cries. The baby's behavior (whether to laugh or cry) is obtained by simulating the agent to determine what the agent would do if the agent were in the baby's position, assuming that the baby feels pleasure each turn that its nutrition is within specified bounds and feels pain when its nutrition goes outside those bounds.

At first glance, one might imagine the agent's optimal strategy is to feed the baby so as to keep its nutrition within happy bounds at all times. But what would the agent do in the position of a baby always so fed (and thus always given $+1$ reward regardless what actions it takes)? Presumably, in that position, the agent would have no way of associating its rewards with its actions, and so would act randomly, sometimes crying and sometimes laughing. Apparently, it would be better for the agent to calibrate feedings in such a way that the baby will learn a relationship between pleasure and laughter. Of course, Example 6 is a gross over-simplification. In reality, there would not be such a simple formula for the baby's nutrition level, and the agent would need to figure out the nutrition level based on observing the baby laughing or crying. Both baby and parent would need to learn how to communicate with each other effectively.

With the above in mind, extended environments might shed light on how living organisms evolve self-reflection. Assume children inherit their policy source-code from their ancestors (possibly with minor mutations). Then whenever an organism interacts with other organisms, it interacts with an environment whose reactions depend (via those other organisms' actions) approximately on that organism's own source-code. The closer the organism is related to the other organisms with which it interacts, the more closely this approximation holds. A human, when interacting with another human, might achieve better results by self-reflectively considering, "what would I do in this other person's position?"

We implement Example 6 as CryingBaby.py.

4 Making agents more self-reflective

One advantage of empirically measuring the self-reflection of RL agents is that it provides a way to experimentally test whether various transformations make various agents more self-reflective. To illustrate this, we will define a simple transformation, the *reality check* transformation, designed to increase the self-reflection of deterministic agents (a deterministic agent is an agent who always takes the same actions in response to the same input, i.e., an agent with no random component). In the next section, empirical results will suggest the transformation works as intended.

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

$$\pi_{RC}(\langle x_1 \rangle) = \pi(\langle x_1 \rangle)$$
$$\pi_{RC}(x_1 y_1 \dots x_n) = \begin{cases} \pi(x_1 y_1 \dots x_n) & \text{if } y_i = \pi_{RC}(x_1 y_1 \dots x_i) \text{ for all } 1 \leq i < n, \\ \pi(\langle x_1 \rangle) & \text{otherwise.} \end{cases}$$

In other words, π_{RC} is the agent which, at each step, first reviews all the actions which it has taken in the past, and verifies that those are the actions which π_{RC} would have taken. If so, then π_{RC} acts as π would act. But if any action which the agent has taken in the past was not the action π_{RC} would have taken, then π_{RC} freezes up and forever thereafter takes the same fixed action, as if catatonic. Since the act of reviewing one's past actions and verifying that they are indeed the actions one would take, is an act of self-reflection, it seems plausible that if π is intelligent and deterministic but lacks self-reflection, then π_{RC} is more self-reflective than π. In the next section, we will see that experimental evidence supports this hypothesis. We close this section by stating some simple results about the transformation.

Proposition 8. *Let π be any deterministic agent.*

1. *(A more efficient way to compute π_{RC}) An equivalent alternate definition of π_{RC} is:*

$$\pi_{RC}(\langle x_1 \rangle) = \pi(\langle x_1 \rangle)$$
$$\pi_{RC}(x_1 y_1 \dots x_n) = \begin{cases} \pi(x_1 y_1 \dots x_n) & \text{if } y_i = \pi(x_1 y_1 \dots x_i) \text{ for all } 1 \leq i < n, \\ \pi(\langle x_1 \rangle) & \text{otherwise.} \end{cases}$$

2. *(Idempotence) $\pi_{RC} = (\pi_{RC})_{RC}$.*
3. *(Equivalence in traditional RL) For every deterministic traditional environment μ, the result of π_{RC} interacting with μ equals the result of π interacting with μ.*

For the proof of Proposition 8, see appendix. Unfortunately, even the more efficient definition of π_{RC} still involves potentially $O(n^2)$ many calls to π, making π_{RC} computationally expensive.

5 Example measurements

Based on our conviction that self-reflection is necessary in order for an agent to achieve good average perform across many extended environments, self-reflection can be estimated by running an agent against some standard battery of extended environments. Our open-source library of extended environments [1] provides a battery of 25 such extended environments, and infrastructure

for measuring an agent's self-reflection by running the agent on all these environments and their opposites for a given number of steps.[1] For uniformity, all environments in the library always output rewards of either 1, -1 or 0.

We have used this library to measure the self-reflection of the following agents:

- RandomAgent: An agent who acts randomly.
- IncrementerAgent: An agent who systematically iterates through available actions.
- ConstantAgent: An agent who always takes the same action.
- NaiveLearner: An agent who acts randomly 15% of the time, and otherwise takes the action which yielded the highest average immediate reward in the past.
- A2C, DQN, and PPO agents from the Stable Baselines3 library [8] (MIT-licensed); see remark below.
- RecurrentDQN: A custom implementation of a recurrent DQN agent; see remark below.
- The reality checks of all the above (Definition 7).

We remark that because our because of our universalist approach to RL (with no notion of pre-training the agent), it was necessary to adapt the A2C, DQN, and PPO agents accordingly. To obtain an agent π (as in Definition 1) from, say, a PPO implementation (which assumes pre-training and an environment conforming to OpenAI Gym's environment interface), we proceed as follows. To evaluate $\pi(x_1 y_1 \dots x_n)$, we create instantiate an OpenAI Gym-conformant dummy environment hardcoded to blindly regurgitate rewards and observations $x_1, \dots, x_n$. We then run the PPO implementation on this environment for n steps, of which the first k steps are training steps and the remaining $n-k$ steps are post-training steps, where $k = \max\{2^\ell \,:\, 2^\ell < n\}$ (this allows the pre-training phase to be memoized, otherwise π would be computationally infeasible). For the training phase, we monkeypatch the implementation so that whenever it would choose an action randomly or from its policy, it is instead forced to choose that action systematically from $y_1 \dots y_n$. For additional details about these agents, see [1].

Table 5 summarizes how the agents performed. Computations were performed on a consumer-grade laptop with no GPU. The table provides experimental evidence in support of our hypothesis that the reality check transformation (Section 4) increases agent self-reflection, at least for non-recurrent agents. The fact that NaiveLearner performs so well is a reflection of the lack of sophistication of the environments in our library. This is not particularly surprising, since we have not attempted to exhaustively optimize the library, instead prefering to fill it with extended environments of theoretical interest.

References

[1] Anonymous. ExtendedEnvironments, 2021.

[2] 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.

[3] 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.

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

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

[1] By the *opposite* of an environment we mean the environment obtained by multiplying all rewards by -1. Including these opposite-environments serves to normalize agent performance in the following sense. If an agent blindly acts, ignoring the environment, then, a priori, that agent might achieve some nonzero score by blind luck. By including opposite-environments, we ensure that whenever a blind agent gains points by blind luck from one environment, it loses the same points by blind misfortune from the opposite environment. This ensures that such blind agents should receive an average score close to 0 (possibly non-zero due to randomness).

Table 1: Measuring self-reflection of some agents

Agent	Avg Reward (100 steps)	Avg Reward (200 steps)
RandomAgent	$-.018$	$-.037$
IncrementerAgent	$-.031$	$-.041$
ConstantAgent	$+.006$	$-.001$
NaiveLearner	$+.196$	$+.224$
A2C	$-.088$	$-.049$
DQN	$+.019$	$+.082$
PPO	$-.018$	$-.021$
RecurrentDQN	$+.005$	$+.038$
$\text{IncrementerAgent}_{\text{RC}}$	$-.018$	$-.020$
$\text{ConstantAgent}_{\text{RC}}$	$+.004$	$-.001$
$\text{NaiveLearner}_{\text{RC}}$	$+.551$	$+.602$
A2C_{RC}	$+.042$	$+.042$
DQN_{RC}	$+.158$	$+.237$
PPO_{RC}	$+.002$	$+.004$
$\text{RecurrentDQN}_{\text{RC}}$	$+.059$	$-.014$

[6] 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.

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

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

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] In order for this paper's measurement technique to realize its full potential, it will be necessary for much more study to be put into the design of a sufficiently representative library of benchmark extended environments. We make this clear in both the abstract and the introduction, and we discuss the problem in Section 5 as well.
   (b) Did you describe the limitations of your work? [Yes] In addition to the remarks in the previous checklist item, we have also mentioned that the reality check agents are computationally very expensive (Section 4). We also point out in the Introduction that although this technique can be used to numerically estimate the degree of self-reflection of an agent, this should not be confused with measuring the consciousness of an agent.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] In the Introduction, we included the following language: "In particular, it would be absolutely inappropriate to use self-reflectiveness measurements using the techniques in this paper for purposes of making any kind of policy decisions related to consciousness."
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] In the statement of Proposition 8 we stated necessary hypotheses, namely, that the agent be deterministic, and, for part 3, that the environment be deterministic.
   (b) Did you include complete proofs of all theoretical results? **[TODO]**
3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] These are included in the open-source library, which we will include as supplemental material.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [No] We determined that these details would distract from the paper, so we instead refer the reader to the open-source library, where these decisions are documented with comments.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? **[TODO]**

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Yes, in Section 5 we mention that the computations were performed on a consumer-grade laptop without GPU.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] We use and we cite Stable Baselines3.

(b) Did you mention the license of the assets? [Yes] We mention that our open-source extended environment library is MIT-licensed. We also mention that Stable Baselines3 is MIT-licensed.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Alongside the paper, we are publishing an open-source library of extended environments. This will be included in the supplemental material.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]