CROWDPLAY: CROWDSOURCING HUMAN DEMONSTRATIONS FOR OFFLINE LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Crowdsourcing has been instrumental for driving AI advances that rely on large-scale data. At the same time reinforcement learning has seen rapid progress through benchmark environments that strike a balance between tractability and real-world complexity, such as ALE and OpenAI Gym. In this paper we aim to fill a gap at the intersection of these two: The use of crowdsourcing to generate large-scale human demonstration data in the support of advancing research into imitation learning and offline learning. To this end, we present *CrowdPlay*, a complete crowdsourcing pipeline for any standard RL environment including OpenAI Gym (made available under an open-source license); a large-scale publicly available crowdsourced dataset of human gameplay demonstrations in Atari 2600 games, including multimodal behavior and human-human and human-AI multiagent data; offline learning benchmarks with extensive human data evaluation; and a detailed study of incentives, including real-time feedback to drive high quality data. We hope that this will drive the improvement in design of algorithms that account for the complexity of human, behavioral data and thereby enable a step forward in direction of effective learning for real-world settings.

1 INTRODUCTION

Crowdsourcing has been instrumental in many AI advances, especially recent rapid progress in deep neural network models, which often rely on large training sets. For instance ImageNet (Deng et al., 2009), a large database of annotated images, has enabled a number of breakthroughs in image classification (Krizhevsky et al., 2012). At the same time, reinforcement learning (RL) has seen rapid progress in the last few years, fueled in part by the development of standard, easily accessible, benchmark environments like the Arcade Learning Environment (ALE) (Bellemare et al., 2013; Machado et al., 2018) and OpenAI Gym (Brockman et al., 2016). What has been underexplored is the intersection of the two: Using large-scale crowdsourced human data for offline learning, including imitation learning and offline RL.

We present *CrowdPlay*, a framework, methodology, and dataset that we hope will do for offline learning what ALE and Gym what ALE and AI Gym did for online learning. CrowdPlay supports flexible and scalable crowdsourcing that is geared towards multi-channel recruitment, and is able to interface with any OpenAI Gym or Gym-like Markov decision process (MDP) environment. It supports real-time feedback to participants that can be used to boost data quality, as well as both purely human and mixed human-AI multiagent environments.

CrowdPlay is also the first dataset based on Atari 2600 games that features multimodal and multiagent behavior. It includes both data from normal gameplay as well as explicitly multimodal behavioral data, where players are given instructions to follow a specific behavior. In addition to single-agent data, the dataset includes data from two-player, human-AI and human-human games, and with both competitive and cooperative rewards. Participants were recruited through multiple channels (under IRB, details withheld for blind submission) including Amazon Mechanical Turk, Lab in the Wild, undergraduate students, and multiple social media channels. For some platforms we also include data with a range of different incentive structures for participants. The Atari games were run using ALE and a multiagent version of OpenAI Gym, guaranteeing that transitions are identical to what would be seen in standard Atari RL environments.

In this paper we focus on the use of CrowdPlay for Atari 2600 games, but a major advantage of the approach is that it works for any Gym-like MDP environment. We believe that Atari 2600 games are interesting for imitation learning (IL) and offline RL for the same reasons that they were seminal as a challenge problem in the development of RL: they offer a balance between achievable short-term research advances and sufficient complexity. More recent work in psychology has also shown them to be of sufficient richness to support the study of human learning behavior (Tsividis et al., 2017).

Further, and despite the richness of the data, it is easy to collect at scale through the use of crowdsourcing and a web browser, and moreover, it can be used together with an established simulator for evaluation purposes. The CrowdPlay pipeline directly interfaces with standard RL environments. In particular this means that trajectories and transitions are guaranteed to be the same for human data as they are for RL environments; that offline learning methods automatically have access to a simulator; and that crowdsourcing with human players need not develop environments and tasks from scratch, but can make use of AI agents that can interact with human agents in a benchmark environment.

1.1 Related Work

Crowdsourcing Previous platforms for crowdsourcing such as *TurkServer* (Mao et al., 2012) have focused on supporting synchronous participation through Amazon Mechanical Turk participants and used to study economic behavior in simple experimental environments (and not for the generation of human behavioral data). Tylkin et al. (2021) use a combination of ALE and Javatari for crowdsourcing Atari data in the context of evaluating the performance of an AI agent in human-AI collaboration. They propose modifying two-player Space Invaders to make it cooperative, and to train AI agents using randomized starting position, both of which we adopt in (some of) our multiagent environments. Their crowdsourcing approach is Atari-specific and not publicly available. Much work has been done on the study of incentives for participants in paid crowdsourcing studies, and this is also part of the focus of our work. Prior work (Mao et al., 2013; Mason & Watts, 2009; Yin et al., 2013; Harris, 2011; Shaw et al., 2011) has largely found that quality-dependent payments may increase quantity of work more than quality of work, and has not looked at real-time feedback on work quality.

Offline Learning. Much work has been done on offline learning (Rashidinejad et al., 2021), including Behavior Cloning (BC) (Pomerleau, 1989), Batch Constrained Q-Learning (BCQ) (Fujimoto et al., 2019), Conservative Q-Learning (CQL) (Kumar et al., 2020), Implicit Quantile Network (IQN) (Dabney et al., 2018), DQN (Mnih, 2015) and Off-policy version of Soft Actor-Critic (Haarnoja et al., 2018)

Specific to Atari, Kanervisto et al. (2020) benchmark behavioral cloning algorithms on existing data from several video games, including Atari 2600 games. They do not use memory in the policy nor framestacking in the observation, which we believe limits the potential performance of their approach on Atari games. Laurens & Kazmi (2021) clone River Raid agents using existing datasets, and develop evaluation metrics based on action distribution and playstyle. They note that their approach struggles to clone a human agent.

Atari 2600 datasets *Atari Grand Challenge* (AGC) (Kurin et al., 2017) is a dataset consisting of 45 hours of standard gameplay from five Atari 2600 games. They use a browser app running the Atari emulator in the browser based on Javatari. It is unclear to us if this can be guaranteed to always have identical execution to the ALE emulator.

Atari-Head (Zhang et al., 2020) features 117 hours of gameplay data and includes eye-tracking data. This data was collected using ALE. However, the emulator was run in a semi-frame-by-frame mode, advancing the emulator state only when a key was pressed, and at a maximum of 20 frames per second. The focus of the study was on attention tracking, and is not intended to be representative of natural human gameplay behavior.

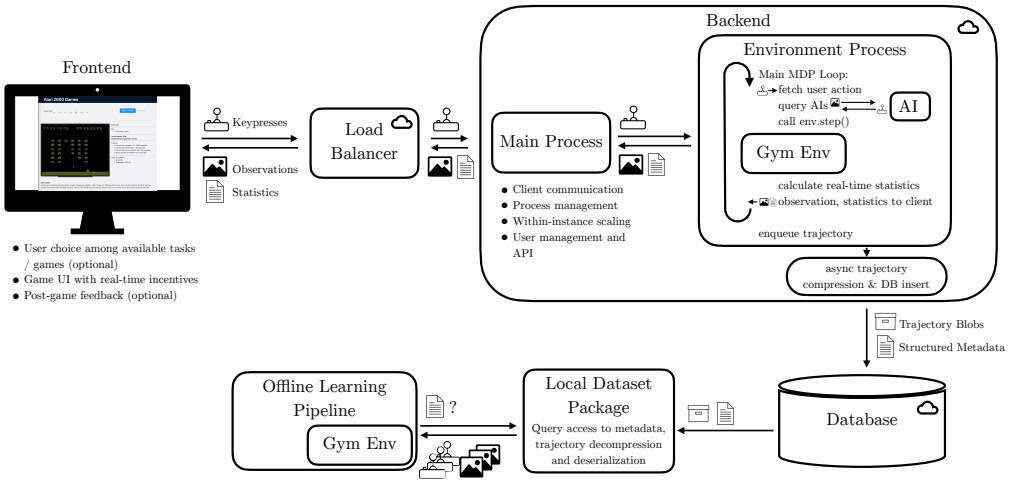

Figure 1: Key parts of the CrowdPlay software architecture. Arrows show the flow of keypresses, observations and metadata between browser client, MDP environment, AI policies, and the database, and the eventual flow to an offline learning pipeline. The backend, load balancer, and database are hosted on cloud infrastructure.

2 CROWDPLAY: THE PIPELINE

2.1 OVERVIEW

The heart of our pipeline are the CrowdPlay backend and frontend, which is a client-server architecture that streams OpenAI Gym environments and similar MDP environments to web browser clients. It is highly extensible, scalable to hundreds or thousands of concurrent users, and allows the real-time capture of both trajectories as well as related statistics. It is geared toward multi-channel recruitment of participants and strong incentives. As its most important feature, it interfaces directly with OpenAI Gym and similar environments, thus opening up the entire array of standard RL environments to rapid crowdsourcing of behavioral data into the support of research into IL and offline RL. It also supports multi-agent environments, including mixed human-AI environments.

Complementing this is an engine to support the local download of the generated dataset, including storage of metadata in a relational database for fast access, and compressed trajectories for storage efficiency. The data download can be real-time and is incremental.

We give a short overview of the CrowdPlay architecture, and refer the reader to Appendix A.1 for more information.

2.2 SOFTWARE ARCHITECTURE

CrowdPlay provides a highly extensible, high performance client-server architecture for streaming MDP environments to remote web browser clients. The backend interfaces with OpenAI Gym and similar MDP environments. Actions are collected as keypresses in the browser client, and sent to the backend where they are fed into the MDP's "step" function. The returned observation is sent back to the browser client for display. This is repeated to generate an episode trajectory. The remainder of the CrowdPlay software infrastructure is built to make this basic loop into a structure that is robust, performant, extensible, user-friendly and scalable. Figure 1 shows the key parts of the CrowdPlay architecture.

Communication between the browser client and backend is through high-performance socket connections. The backend is built to be scalable both within-instance, using multiple processes, as well as across-instance using a load balancer and autoscaling instance groups. Trajectories are stored di-

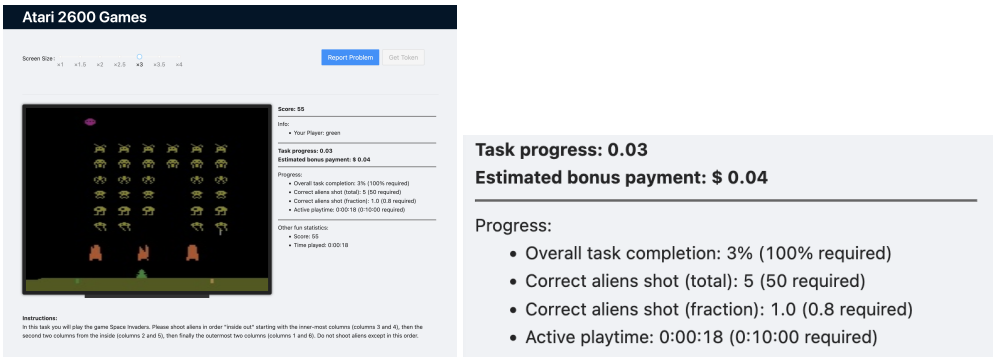

Figure 2: Screenshots of the main screen of CrowdPlay (left) and an enlarged detail of the realtime incentives (right).

rectly as compressed, serialized Python objects, allowing both very easy modification of data capture as well as immediate decoding for use in existing Python-based learning pipelines.

CrowdPlay also supports multi-agent environments. It allows multiple human participants by routing multiple browser clients to a single MDP environment. Mixed human-AI environments are supported through pre-trained neural network policies. For robustness, AI agents can also take over control of a human agent on the fly, in the case that a human player disconnects from a multiagent environment, allowing uninterrupted gameplay for the remaining human players.

2.3 Incentives and Recruitment

CrowdPlay provides an extensible and easy-to-use framework for collecting structured metadata and real-time statistics per user, session, episode, and individual steps. This is used for capturing data quality information, as well as for driving real-time incentives for participants. On platforms where users are paid or otherwise compensated (e.g. through entry into a raffle), we use CrowdPlay's real-time statistics engine to create strong incentives, including minimum effort requirements for compensation, and bonus payments that scale with performance. On platforms where users are not compensated, we use the same real-time statistics to reinforce intrinsic motivation through live "high score" pages and by reframing non-standard gameplay tasks as "challenges." We discuss the platforms that we target in more detail in Appendix A.3, and the various incentive designs and their effect on data in Section 4.

2.4 Data Post-processing for ML

A major focus in the design of CrowdPlay is providing the ease of access of the generated data in downstream ML pipelines. We believe it is crucial for these pipelines to have access not only to the same simulator as the crowdsourcing pipeline, but also to the same observation pre-processing tools that are used in state-of-the-art RL methods on these simulators. Addressing these design goals, CrowdPlay includes a custom, Deepmind-style observation processing function. This is implemented through a custom frame buffer that replaces the usual "Deepmind" wrappers, is verified to give the same output frame-by-frame and pixel-by-pixel as standard implementations of wrappers such as those in OpenAI Gym, and can be applied post-hoc to already recorded trajectories. This same observation processing is also used for the AI agents that we bring into mixed human-AI environments.

3 CrowdPlay Atari: The Dataset

3.1 Scope and Motivation

Our main focus in creating the first CrowdPlay dataset has been on Atari 2600 games, as we believe that human data for these environments can enable advances in IL and offline RL (just as they have

been in driving advances in online RL, and for the same reasons). In curating the dataset, we have been especially motivated by diversity—we have used multiple recruitment channels, each with different extrinsic and intrinsic user motivation and incentive models, and have reached over 800 users in 1300 sessions over a three week period in September 2021. The CrowdPlay Atari dataset currently holds over 250 hours, or 54 million transitions, and was collected across six different games. We have targeted not only implicitly multimodal behavior through recruiting different users and through different channels, but also explicit multimodal behavior through explicit instructions that are reinforced through tight incentives and real-time feedback. We include both single agent as well as multi-agent data, with the latter reported for both human-human and human-AI gameplay, and with both competitive as well as cooperative rewards. We believe this is the first multi-agent human behavioral dataset, at the very least for Atari 2600 games.

3.2 Task Design

We used the following Atari 2600 games for our dataset: Space Invaders, RiverRaid, Montezuma's Revenge, Q*bert, Breakout and Beamrider. Space Invaders makes up the largest part of our dataset. We chose this game as a focus for several reasons: it is well-known and popular both among Atari players as well as in the RL community, it was easy to come up with several specific and distinct behaviors that are still compatible with progressing in the game, and it has a native two-player mode. RiverRaid provides the second largest amount of data and was chosen for similar reasons, in that it is accessible and well understood from a learning perspective, and has obvious opportunities to promote multimodal behavior. The remaining games were chosen for diversity of game types and their popularity in previous RL and IL work.

Multimodal Behavior For Space Invaders and RiverRaid, in addition to standard gameplay, we asked some participants to follow specific behavior in the game. We believe that this data will be useful for multiple research strands. In imitation learning, this type of data can provide a testbed to understand whether algorithms are robust to divergent expert behavior. In offline RL, this can allow for controlled experiments with different reward functions in an otherwise identical environment. For most of this data, we recruited participants via Mechanical Turk, and gave incentives via both a minimum level of task adherence required to complete the HIT (unit of task), as well as an explicit reward function tied to a bonus payment, and with real-time feedback on the same. The behavior types we instructed participants to follow were to either stay on either half of the game screen (in both Space Invaders and Riverraid, or to shoot aliens in a particular order (row by row from the bottom, column by column from the outside in, or column by column from the inside out; in Space Invaders only.) We discuss these tasks in more detail in Appendix A.2.1

Multiplayer Games CrowdPlay also contains a set of trajectories from multi-agent environments. For these, we used the two-player mode of Space Invaders. We include data from two variants: (1) the standard two-player Space Invaders mode, and (2) a custom cooperative mode. In the latter, we modify the original game in two ways. First, in the standard two-player mode there is a score bonus if the other player loses a live. We removed this score bonus. Second, we gave both players the sum of their individual scores. These modifications were done entirely in Python, in the case of the score bonus detecting these events from emulator state. Participants were instructed to ignore the score shown on the game screen, and were instead shown their "cooperative" score next to the game screen. Most of our multiplayer data is from mTurk, and people were incentivized through bonus payments to maximize this score.

Further, we include data from games with two human players, as well as games with one human and one AI player. AI agents were trained in two-agent environments with two separate A2C policies. We used RLLib's implementation of A2C (Mnih et al., 2016; Liang et al., 2018) for 40 million timesteps, using standard Deepmind-style observation processing and hyperparameters. We trained separate agents for the standard and cooperative variants. Following findings in (Tylkin et al., 2021) we trained AI agents using randomized starting positions each episode for better exploration of the states likely encountered in human-AI environments.[1]

[1] AI agents were also used in nominally human-human environments with MTurk participants, in two circumstances. One, if one human player disconnected, an AI agent took over control. This is to allow the remaining MTurk worker to complete their task. Similarly, if an MTurk worker was waiting for 15 minutes

Table 1: Dataset by game and recruitment channel

Task	**Data Collected (hours)**			
	MTurk	**Social Media**	**Email Raffle**	**Total**
Beamrider	**7.90**	**-**	**-**	**7.90**
Breakout	**11.45**	**-**	**-**	**11.45**
Montezuma's Revenge	**9.94**	**3.28**	**2.48**	**15.70**
Q*Bert	**6.60**	**1.98**	**-**	**8.58**
Riverraid	**17.60**	**3.64**	**2.41**	**23.65**
- of which multimodal	12.28	0.44	0.58	13.30
Space Invaders	**128.56**	**8.87**	**5.68**	**143.11**
- of which multimodal	94.89	0.20	1.12	96.21
- of which incentive experiment	23.25	-	-	23.25
Space Invaders (2P)	**6.35**	**0.38**	**0.38**	**7.12**
Space Invaders (2P w/AI)	**54.98**	**0.77**	**0.71**	**56.46**
Total	**243.37**	**18.91**	**11.66**	**273.95**

Table 1 shows a list of available data by game and recruitment channel.

3.3 Challenges and Limitations

We discuss here some challenges that we experienced in collecting this data and the corresponding limitations in the dataset. One, our incentive design has continued to evolve, while a significant amount of the data has been collected with suboptimal incentives. This is part of why we discuss incentive design in the next section in detail, to allow other researchers to build on what we have learned. Two, some of our dataset is unbalanced, especially for the different behavior types in the multimodal Space Invaders data, where we have 2-3 times as much data for some behaviors as for others. This is partly due to suboptimal incentives and some tasks being harder than we had intended, and partly because we improved the way the pipeline assigned MTurk workers to tasks over time. We have since then improved the automatic task assignment logic to more carefully take into account the amount of data already collected. Three, we have found that collecting human-human multiagent data is a difficult challenge; e.g., we expected that the ability to play with friends would be a major draw in social media recruitment, but found that we had virtually zero uptake on this. On the other hand, we have made significant advances in making multiplayer data collection more feasible on MTurk. In particular, the use of fallback AI agents has solved many practical problems with workers needing to depend on one another for successful task completion. Still, we were able to collect much more single-player and human-AI data than human-human data.

4 Incentive Design

4.1 Incentive Design and Recruitment

CrowdPlay has been designed with multichannel recruitment and strong participant incentives in mind. It supports configuring multiple environments and "tasks" for participants to complete, including different user instructions, incentives, and AI agent configuration. At the heart of its capabilities is its ability to capture an extensible list of metadata live during gameplay, including data such as playtime, score, and various in-game behavioral characteristics. In platforms where users are paid for their participation, we use this data to define both a minimum acceptable effort by users to get paid at all, as well as a dynamic bonus payment that can depend on a fine-grained analysis

without a second human player connecting, a game would start with an AI agent controlling the other player. The dataset includes a record of what type of agent was controlling each player at every frame.

of user effort. Both progress toward minimum required effort as well as bonus payments can be displayed to users during live gameplay. For instance, in one Space Invaders task we asked participants to shoot aliens in a specific order. Our architecture is able to determine adherence to these instructions by evaluating emulator state at every frame, detecting when an alien has been shot and which one, and keeping a running count of aliens shot in and out of the prescribed order. Users were rewarded based on both the total number of aliens they shot in the correct order, as well as the fraction of aliens that they shot correctly. Using this framework, we can mimic the reward structure of an MDP through monetary incentives and also make use of realtime feedback; and we can additionally shape incentives to achieve particular modalities. Figure 2 (right) shows an example of this realtime feedback.

4.2 INCENTIVE MODELS

We provide various incentive models, as well as real-time feedback to participants. For social media participants, this was feedback-only. For students we made use of a raffle, and required a minimum time and effort, e.g. 80% of aliens shot in the correct order. For participants from Mechanical Turk, we experimented with multiple models of incentives.

Experimental design Specifically, we report the results of an extensive experiment with the Space Invaders "Inside-Out" task. In this, we gave participants identical instructions for the behavior they were asked to follow, but adopted five different ways of remunerating them:

1. No incentives: In this treatment, we paid participants a lump sum after 5 minutes of time of running the game, irrespective of their actions or performance in the game.
2. Active Time: As above, however we only count time if there was at least one user keypress in the last 30 seconds, and at least one non-zero game reward in the last 60 seconds.
3. Minimum Requirement: In addition to 5 minutes of play time, we require that participants follow instructions to at least some minimal degree. We require they shoot a minimum of 50 aliens in the correct order, and that at least 80% of aliens shot were in the correct order (i.e. from the innermost columns that still had any aliens present).[2]
4. Sliding Bonus: We pay participants a much smaller lump sum "base payment" after 5 minutes of active playtime and a bonus payment that reaches a maximum if participants shot at least 200 aliens in the correct order, and 100% of aliens short were in the correct order. The base and full bonus payment were chosen to equal the lump sum payment of the previous incentive regimes. The bonus payment p scaled linearly in both the number of aliens shot in the correct order n_{correct}, as well as the fraction of aliens shot in the correct order above one-half, i.e. $n_{\text{correct}}/n_{\text{total}} - 0.5$:

$$p = p_{\max} \times \max(1, \frac{n_{\text{correct}}}{500}) \times \min(0, 2(\frac{n_{\text{correct}}}{n_{\text{total}}} - 0.5)). \tag{1}$$

5. All Incentives: We pay a small lump sum subject to the conditions in the minimum requirements regime and a bonus as in the sliding bonus regime.

For both task requirements and bonus payments, participants were given real-time feedback. For minimum requirements, this took the form of an itemized list of requirements and their performance, e.g. "Correct aliens shot: 40 (required: 50)". For bonus payments, this took the form of a live estimate of the expected payment, but did not include details on how the payment was computed. Task instructions stated that the bonus payment would depend on both the number of aliens shot as well as how many were shot in the correct order.

[2]We chose to not provide a base payment for participation, having in mind a very low requirement would only act to filter out no-effort submissions, i.e., spam. We later noticed a number of workers dropping the task, and leaving without payment. We do not know for sure why this is—it is possible workers got bored, but it is also possible that some workers struggled to meet the minimal requirements, despite our intention of setting a trivially low bar. For future studies we recommend continued care in deciding on minimum requirements, and the use of a small base payment that is contingent only on active playtime. On the other hand, with one exception (where a task was harder than we expected, and we later lowered the requirements), we did not receive significant negative feedback from MTurkers regarding the task. Of those that submitted a HIT, the feedback was overwhelmingly positive and often described the tasks as "fun" or similar.

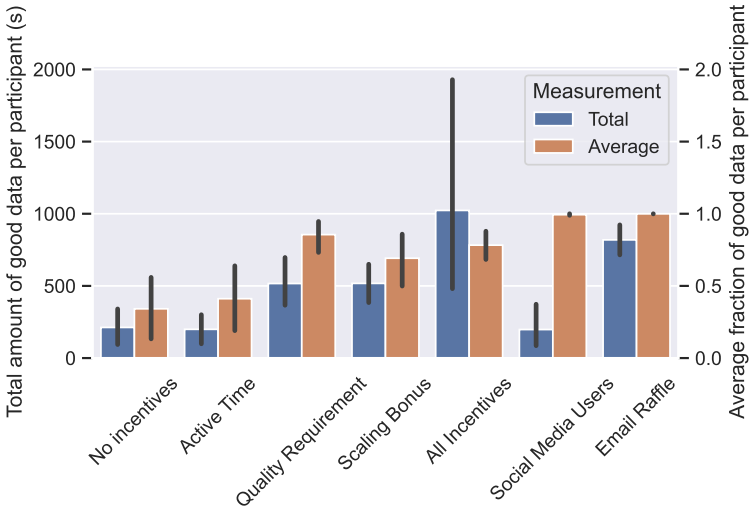

Figure 3: Data quality for different incentive treatments and recruitment channels. Blue bars show the total amount (in seconds) of "good data" collected per user, where this is defined as episodes with at least 80% task adherence. Orange bars show the fraction of good data compared to the total data collected per user.

We also compare data from these treatments with data collected from social media users, as well as students reached on campus via email. For email participants, we enforced a similar minimum requirement as for the minimum requirement treatment on MTurk, with successful completion of the requirement earning an entry into a raffle.

4.3 Results

In regard to the structure of incentives and effect on data, we report several main findings. One, data quality was significantly higher when using any kind of quality-based incentives, be it a minimum requirement tied to a fixed-amount payment, a sliding bonus payment, or a combination of both. Non-incentivized data was low quality even when enforcing that participants actively play the game, thus filtering out any users that put in no effort at all (e.g. by starting the game and then switching to a different window). The orange bars in Figure 3 show this, being higher for the middle three treatment groups than for the left two. Compared to the previous literature on incentives on MTurk, our model differs in that bonus payments are on a sliding scale rather than an all-or-nothing fixed payment, and that participants are given real-time feedback. This might explain why we see a larger effect from bonus payments than previous studies.

Two, the incentive treatments led to a larger amount of good data being collected per participant. Further, the analysis suggests that the combination of minimum requirement and sliding bonus payments can lead to a larger amount of good data collected per participant than either treatment by itself. In other words, while the quality was already high and did not improve much further, the combination of both treatments leads people to play much longer. We caution however that this effect is not shown with statistical confidence, and that more investigation is needed. The blue bars in Figure 3 show this.

Three, participants recruited via social media and email had the highest data quality, but we found these to also be the least scalable recruitment channels (specifics on mechanisms of social media recruitment omitted for blind review). For social media, the amount of data collected per participant was the lowest of these channels, and the total number of users from social media and email was also lower than those from MTurk, and it would easily be possible to recruit more MTurk users.

5 BENCHMARK RESULTS

Our framework provides a variety of data compositions that exhibit real-world intricacies and diversities. We envision that this would help to facilitate rapid progress for offline learning methods that rely on learning from diverse data compositions without access to online exploration (Rashidinejad et al., 2021). We therefore evaluate our datasets on recently proposed offline learning algorithms and hope that it will inspire potential future directions in this area. As the Atari is a discrete action domain, we focus on algorithms that can handle discrete actions effectively. Specifically, our baseline algorithms include Behavior Cloning (BC) (Pomerleau, 1989), Batch Constrained Q-Learning (BCQ) (Fujimoto et al., 2019), Conservative Q-Learning (CQL) (Kumar et al., 2020), Implicit Quantile Network (IQN) (Dabney et al., 2018), DQN (Mnih, 2015) and Off-policy version of Soft Actor-Critic (Haarnoja et al., 2018). As evaluate the algorithms on different task, in Figure 4, we provide the normalized version of the performance across tasks (Agarwal et al., 2020). We provide individual normalized and unnormalized score for each task and experiment details in Appendix B. From the results, we observe that offline version of reinforcement learning algorithms outperforms behavioral regularized and constrained algorithms. Further, while DQN does not perform consistently well across all tasks, it still fares better than both the distributed methods and comparable to the behavior regularized versions. Behavior cloning shows surprisingly well performance which may be due to artifact of those tasks where the collected demonstrations are of high quality for that task but does not cover the state space well, making it difficult for the offline reinforcement learning algorithms. Overall, the results for all the algorithms demonstrate the open challenges on learning from human demonstrations where the data is both limited and noisy which raises the need for more robust and sample efficient methods.

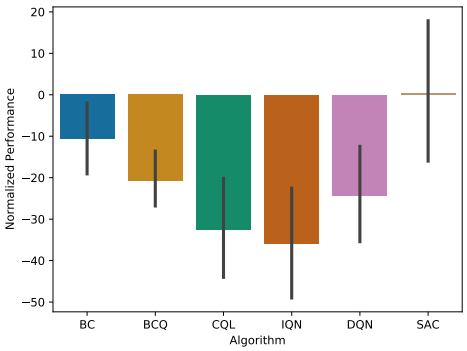

Figure 4: Evaluation Performance of offline RL algorithms across different tasks. The bars indicate the normalized score across tasks, and the error bars show the 95% confidence interval. The score normalization is computed using the best score achieved by humans across each task.

6 DISCUSSION

We have presented CrowdPlay, a pipeline for crowdsourcing human demonstration data in standard RL environments, and a first dataset along with benchmarks on Atari 2600 games. A key advantage of our approach is that it allows easily accessible crowdsourcing for any existing RL environment, and it can provide realtime feedback and incentives that mirror arbitrary RL reward functions or other kinds of desired, complex behavior types. Further, by using standard RL environments we can streamline the end-to-end crowdsourcing and learning pipeline: CrowdPlay can use the same environment simulator that is used by an existing RL pipeline, enabling for instance mixed human-AI environments that use the same algorithms already used to train AI agents. We hope that this new framework and dataset will enable rapid progress in offline learning on large-scale human demonstrations. Atari 2600 games were seminal in driving progress in online RL, and we believe this dataset and pipeline can advance IL and offline RL research for the same reasons that Atari 2600 is interesting for online RL. Moreover, this pipeline is not constrained to Atari games, and we hope that it will be utilized for other environments. Beyond supporting progress in IL and offline RL, we also believe that CrowdPlay can enable other novel applications, both in computer science (such as task classification), but also in other fields such as psychology or behavioral economics.

7 REPRODUCIBILITY STATEMENT

Our entire framework and current version of the dataset will be available publicly under an open-source license and we have provided the links to the implementations used to produce the benchmark results in Appendix B.

REFERENCES

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning, 2020.

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.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning, 2018.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Christopher Harris. You're hired! an examination of crowdsourcing incentive models in human resource tasks. In *Proceedings of the Workshop on Crowdsourcing for Search and Data Mining (CSDM) at the Fourth ACM International Conference on Web Search and Data Mining (WSDM)*, pp. 15–18. Hong Kong, China, 2011.

Anssi Kanervisto, Joonas Pussinen, and Ville Hautamäki. Benchmarking end-to-end behavioural cloning on video games. In *2020 IEEE Conference on Games (CoG)*, pp. 558–565. IEEE, 2020.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, 2020.

Vitaly Kurin, Sebastian Nowozin, Katja Hofmann, Lucas Beyer, and Bastian Leibe. The atari grand challenge dataset. *arXiv preprint arXiv:1705.10998*, 2017.

Diels Laurens and Hussain Syed Kazmi. Behaviourally cloning river raid agents. In *AAAI-21 Workshop on Reinforcement Learning in Games, Location: Virtual conference*, 2021.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pp. 3053–3062. PMLR, 2018.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

Andrew Mao, Yiling Chen, Krzysztof Z Gajos, David C Parkes, Ariel D Procaccia, and Haoqi Zhang. Turkserver: Enabling synchronous and longitudinal online experiments. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

Andrew Mao, Ece Kamar, Yiling Chen, Eric Horvitz, Megan E Schwamb, Chris J Lintott, and Arfon M Smith. Volunteering versus work for pay: Incentives and tradeoffs in crowdsourcing. In *First AAAI conference on human computation and crowdsourcing*, 2013.

Winter Mason and Duncan J Watts. Financial incentives and the" performance of crowds". In *Proceedings of the ACM SIGKDD workshop on human computation*, pp. 77–85, 2009.

Kavukcuoglu K. Silver D. et al. Mnih, V. Human-level control through deep reinforcement learning, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Dean A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*, 1989.

Paria Rashidinejad, Banghua Zhu, Cong Ma, Jiantao Jiao, and Stuart Russell. Bridging offline reinforcement learning and imitation learning: A tale of pessimism, 2021.

Katharina Reinecke and Krzysztof Z Gajos. Labinthewild: Conducting large-scale online experiments with uncompensated samples. In *Proceedings of the 18th ACM conference on computer supported cooperative work & social computing*, pp. 1364–1378, 2015.

Aaron D Shaw, John J Horton, and Daniel L Chen. Designing incentives for inexpert human raters. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pp. 275–284, 2011.

Pedro A Tsividis, Thomas Pouncy, Jaqueline L Xu, Joshua B Tenenbaum, and Samuel J Gershman. Human learning in atari. In *2017 AAAI Spring Symposium Series*, 2017.

Paul Tylkin, Goran Radanovic, and David C Parkes. Learning robust helpful behaviors in two-player cooperative atari environments. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 1686–1688, 2021.

Ming Yin, Yiling Chen, and Yu-An Sun. The effects of performance-contingent financial incentives in online labor markets. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

Ruohan Zhang, Calen Walshe, Zhuode Liu, Lin Guan, Karl Muller, Jake Whritner, Luxin Zhang, Mary Hayhoe, and Dana Ballard. Atari-head: Atari human eye-tracking and demonstration dataset. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 6811–6820, 2020.

A Appendix

A.1 Technical Details

Architecture Overview CrowdPlay uses a client-server architecture, where an MDP environment is run in a simulator and its output streamed to a web browser client. The browser client collects user keypresses which are streamed to the server, where they are decoded into actions which are fed into the MDP simulator. The backend hosts the MDP simulator, records trajectories into a database, and provides additional functionality. Communication uses a http API for management tasks such as establishing a connection, and a websockets for streaming keypresses and observations for better performance. The backend can optionally interface with an external load balancer and MySQL instance for scalability.

Backend Architecture The backend is written in Python for best compatibility with existing Python-based RL simulators and easy data interchange with offline learning pipelines. It is highly parallelized across multiple processes: A main process handles http requests to the API, routes websockets communications, and manages the multiple concurrent MDP environments. Each environment simulator is run in its own process dedicated in essence to just the main "fetch action - environment step - get observation" MDP loop. Once an episode is complete, the entire trajectory is asynchronously serialized using Pickle and compressed using Bzip2 in a separate process and inserted into a database.

MDP simulator interface CrowdPlay interfaces with standard MDP simulators such as OpenAI Gym environments, but also similar interfaces. For our current set of experiments, we use a Gym-like interface implemented using RLLib's multiagent environment definition. We use this even in single-agent games, although this is for convenience and not due to an architectural limitation. The main requirements for the MDP simulator are that it provides a reset () and a step (action) function.

Frontend The CrowdPlay frontend is written in React for a modern UI experience. Its main task is to receive image observations and associated metadata (such as real-time task progress, estimated bonus payment and other statistics) and display them to the user. Optionally users can choose the task / game they wish to play in a library-style user interface; users can be shown a post-game "high score" page which shows performance and other game statistics relative to other participants; and users can report technical problems and leave feedback. For multiplayer games, the frontend supports waiting rooms and can notify players who joined early once enough other players have joined to start the game.

Environment and task definitions CrowdPlay supports multiple environments and environment variations concurrently. Each variation or "task" can include a different base environment (e.g. a different Atari game, or a different MDP simulator altogether), different instructions for the user, different incentives and statistics (see below), different configurations of human and AI players in multiplayer games (see below) etc. For instance, one task could be a two-player Space Invaders game with one human player and one AI agent, with the user instructed to follow a particular behavior in game, and a minimum effort requirement and bonus payment tied to their adherence to that behavior (for deployment to paid participants); while another task could be a two-player Space Invaders game with two human players and no incentives (for deployment to unpaid participants). Users can be assigned to a task through a URL parameter. Tasks can also be grouped into task groups. Users can then either select a task of their choice among a task group; or the backend can automatically assign a task. This automatic assignment can take into account the amount of high-quality data already collected for each task, and can either aim to collected a minimum amount for each task before assigning incoming users to the next task, or to collect an equal amount of data for each task. Quality is determined by episode and session-level realtime statistics (see below). If any tasks in a task group are multiplayer tasks, the scheduler will first try to assign users to any already-instantiated environments that are waiting for additional human players, before considering creating a new environment.

AI agents Because CrowdPlay uses standard MDP interfaces also used in established RL methods, it can support AI agents trained with those methods. We currently have an implementation for policies trained with RLLib, which would be easy to extend to other existing libraries like OpenAI Baselines. CrowdPlay includes a custom image preprocessor that replicates the behavior of standard "DeepMind" wrappers such as those included in OpenAI Gym, but without having to wrap around the MDP environment. This allows us to provide AI agents with a standard "four frames stacked, each four frames apart" observation, while providing human players with an unprocessed observation every frame.

Multi-player Multi-agent environments are supported with both multiple human players as well as mixed human-AI games. For human players, the frontend routes multiple users' websocket connections to the same environment process. AI agents are implemented as above. As a key robustness feature, AI agent can take over on-the-fly if a human player disconnects, which enables the remaining human players to continue playing. This is important on platforms such as Amazon Mechanical Turk, so that one participant disconnecting does not leave another participant unable to complete

their task through no fault of their own. In such environments, we create fallback AI policies for all agents when the environment is started, to avoid an interruption during gameplay (instantiating the policy takes a few seconds). However we only start observation preprocessing and querying the AI policy for actions for each agent only if and once the human player controlling that player disconnects.

Real-time statistics engine CrowdPlay can calculate and record gameplay statistics and other metadata using an extensible real-time statistics engine. The information to be recorded can be configured for each task. CrowdPlay ships with several generic statistics collectors such as for score and active playtime, and several game-specific collectors such as the player's x-position in Space Invaders and Riverraid, and the order in which aliens have been hit in Space Invaders. These can be used to determine how closely the player's gameplay behavior matches a given target behavior. In turn, CrowdPlay supports minimum effort requirements to consider a task "complete" as well as sliding bonus payments based on the statistics. For each statistic defined for a task, per-step information is saved as part of the trajectory in a compressed binary blob, while episode-level and session-level aggregate statistics are saved as structured metadata in a relational database to allow for fast filtering of individual trajectories in downstream offline learning pipelines. Statistic collectors are implemented as functions or Python callables that take as input the trajectory information at each step, including actions, observations and emulator state. Because all of this information is saved, statistic collectors can also be run offline against already recorded trajectories should requirements change later.

Deployment and Scalability The CrowdPlay backend is fully containerized using Docker and can be deployed on any compatible platform. Within-instance scaling to multiple processor cores is handled within the backend. CrowdPlay also includes configuration options for deployment using Amazon Elastic Beanstalk (EB). When deploying on EB, a load-balancer can be used, which enables auto-scaling across instances (additional server instances will be started automatically depending on load). This enables CrowdPlay to scale to hundreds or thousands of simultaneous users. EB can also be configured to use AWS Spot Instances for significantly reduced operating cost.

Dataset Engine Recorded trajectories and metadata can be downloaded into a locally installable Python package providing user-friendly metadata search and trajectory loading functionality. Download from the backend MySQL database can be online and incremental. Metadata is saved locally in a SQLite database, and is mapped into Python classes by the dataset package. This allows filtering episodes by metadata using either SQL queries, or native Python techniques (e.g. list comprehension) according on user preference. Trajectories can be stored in the bzip2-compressed format used by the backend, or expanded into uncompressed pickle files for much faster loading at the expense of storage efficiency.

A.2 Dataset Details

A.2.1 Game Variants

We instructed participants to follow these five behavior types:

Left-side-only In both Space Invaders and RiverRaid, we instructed participants to score as high as possible in the game, while staying only on the left half of the game screen.

Right-side-only As above, but staying on the right side of the game screen.

Row-by-row In Space Invaders we instructed participants to maximize game score while aiming to shoot aliens row by row from the bottom; i.e., participants were asked to completely clear the bottom-most row of aliens, then the second-from-the-bottom row, etc.

Outside-in In Space Invaders, we instructed participants to maximize game score while aiming to shoot aliens column by column from the outside in; i.e., participants should completely clear the two outermost columns, before moving on to the next two columns, etc

Table 2: List of all tasks in the dataset

Task	Data Collected (hours)			
	MTurk	**Social Media**	**Email Raffle**	**Total**
Beamrider	**7.90**	**-**	**-**	**7.90**
Breakout	**11.45**	**-**	**-**	**11.45**
Montezuma's Revenge	**9.94**	**3.28**	**2.48**	**15.70**
Q*Bert	**6.60**	**1.98**	**-**	**8.58**
Riverraid	**17.60**	**3.64**	**2.41**	**23.65**
- left	6.77	0.23	0.35	7.35
- plain	5.33	3.20	1.83	10.35
- right	5.51	0.21	0.23	5.95
Space Invaders	**128.56**	**8.87**	**5.68**	**143.11**
- insideout	11.96	0.20	1.12	13.28
- insideout incentives	23.25	-	-	23.25
- left	18.79	-	-	18.79
- outsidein	11.37	-	-	11.37
- plain	10.41	8.53	4.57	23.50
- right	33.95	-	-	33.95
- rowbyrow	18.82	0.14	-	18.97
Space Invaders (2P)	**6.35**	**0.38**	**0.38**	**7.12**
- competitive	2.60	-	0.19	2.79
- cooperative	3.75	0.38	0.19	4.33
Space Invaders (2P w/AI)	**54.98**	**0.77**	**0.71**	**56.46**
- competitive	45.10	0.49	0.54	46.13
- cooperative	9.88	0.28	0.17	10.33
Total	**243.37**	**18.91**	**11.66**	**273.95**

Inside-out In Space Invaders, we asked participants to shoot aliens column by column, from the inside out; i.e., participants should completely clear the center two columns (column 3 and 4), then the next two columns from the center (columns 2 and 5), and finally the two outermost columns (1 and 6).

A.2.2 Full List of Included Data

Table 2 shows a detailed list of available data by game, game variant, and recruitment channel.

A.2.3 Data Evaluation

The dataset contains rich metadata accessible through both SQL queries as well as Python classes. Metadata includes universal attributes like total episode score and length, but also task-specific attributes like total and fraction of aliens shot in the correct order in the above Space Invaders tasks. This allows fine-grained filtering of available trajectories according to specified criteria. We propose that "data quality" should be thought of as multidimensional. For instance, we saw in our multimodal task data both trajectories with moderately high task adherence (as fraction of aliens shot in correct order) and very high overall performance and length, and vice versa trajectories of moderately high performance and very high task adherence. Our dataset allows filtering for minimum levels of both attributes, as well as others. Further, any metadata not saved explicitly as such could be recovered from the trajectories themselves, as the full emulator RAM is stored in addition to observations.

A.2.4 EXAMPLE PARTICIPANT INSTRUCTIONS

Standard Gameplay For standard gameplay, we provided participants with instructions on how to play the game, technical information and platform-dependent information on payment. For instance, for Space Invaders, our instructions to participants recruited via social media read as follows:

> In this experiment you will play the game Space Invaders. Your goal is to shoot as many of the alien invaders shown on the screen as possible. You control the small cannon at the bottom of the screen. Use your keyboard's arrow keys to move your cannon left and right, and use the spacebar key to fire. The aliens will fire at you too. If you are hit by alien fire three times, the game ends. The game will also end if the alien invaders reach the bottom of the screen. A new game will then start automatically, but you can end the experiment clicking the Finish Experiment button on the top right. You can end the experiment at any time, but we encourage you to play at least a few games.

For participants recruited via MTurk, the instructions read:

> In this experiment you will play the game Space Invaders. Your goal is to shoot as many of the alien invaders shown on the screen as possible. You control the small cannon at the bottom of the screen. Use your keyboard's arrow keys to move your cannon left and right, and use the spacebar key to fire. The aliens will fire at you too. If you are hit by alien fire three times, the game ends. The game will also end if the alien invaders reach the bottom of the screen. A new game will then start automatically. You must play for at least 10 minutes and show some effort in the form of a minimum score to complete this task. Your bonus payment will depend entirely on your score.

Multimodal Behavior For specific multimodal behavior we showed specific instructions:

> In this task you will play the game Space Invaders. Please shoot aliens in order "inside out" starting with the inner-most columns (columns 3 and 4), then the second two columns from the inside (columns 2 and 5), then finally the outermost two columns (columns 1 and 6). Do not shoot aliens except in this order.
>
> User your keyboard's arrow keys and space bar to interact with the game. It is up to you to follow the instructions above, please follow them carefully.

Multiplayer Games For multiplayer games, we provided these instructions:

> In this task you will play the game Space Invaders. This game is cooperative - you and the other player both get points if either of you hits an alien. Please IGNORE THE SCORE SHOWN ON THE GAME SCREEN, and try to maximise the score shown in the text box to the right of the game screen. This score is shared between you and the other player. You may want to cooperate with the other player.

In all cases, specific instructions were followed with these generic instructions:

> User your keyboard's arrow keys and space bar to interact with the game. It is up to you to follow the instructions above, please follow them carefully.
>
> Please do not open multiple tabs or windows of this link at the same time. In a two-player task, if you are not being connected to another human player, or there appear to be issues (e.g. other player is not moving at all), please click 'Reload' in your browser window and tell the other player to do the same.

Instructions for other games and tasks were similar.

A.2.5 DATASET ERRATA

In a number of space_invaders_right trajectories, we had initially set a requirement of at least 90% time spent on the right side of the screen. This proved too hard for some MTurkers, and we thus

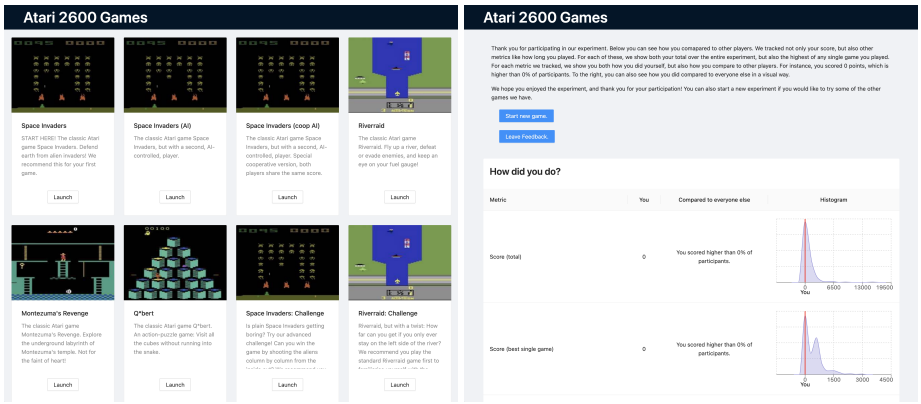

Figure 5: Screenshots of the "game choice" (left) and "high score" (right) screens of CrowdPlay.

lowered the requirement to 80%. This is not distinguished through a separate task ID. MTurkers were compensated for their time in these instances.

In a small number of SI insideout email, RR left right email tasks due to an error a task adherence of 80 instead of 0.8 was formally required, and the task thus not completeable. We later corrected the effort requirement after the fact in the metadata database, both to be able to automatically enter those who achieved the intended requirements into a raffle, and to allow for filtering on that metadata as it was intended. This is the only instance where metadata in the database was altered post-hoc.

A.3 Recruitment and Platform Integrations

We currently target and explicitly support the following platforms.

Amazon Mechanical Turk CrowdPlay can capture and store worker IDs and other metadata for payment processing. Participants are paid a fixed sum for their participation subject to minimum requirements, as well as a variable bonus payment depending on their performance. As above, these are shown to users in real-time, and stored for simple payment processing. In multi-player environments, AI agents can take over if one user disconnects, to allow remaining users to successfully complete their tasks. Participants can be assigned to tasks automatically based on previous participant's performance in a way that aims to balance the amount of high-quality data recorded for each task against defined targets.

Direct User Payments and Raffles Our architecture is able to capture user data like email addresses to contact them for payment or to enter them into a raffle. Task-specific minimum time and effort requirements can be enforced similarly to MTurk.

Lab in the Wild (Reinecke & Gajos, 2015) is a platform for recruiting diverse participants for online experiments. Participants are not monetarily compensated, but are instead motivated through gaining insights about themselves through the experiments. Experiments are thus required to give users feedback with interesting conclusions about the user's performance and behavior. CrowdPlay supports this through its real-time statistics engine, which is used to show users a feedback page after they finish the experiment. Users are shown a variety of statistics about their gameplay behavior beyond score, e.g. shot accuracy, including a comparison with the experiment cohort. Figure 5 (right) shows a screenshot of this.

Social Media Similarly to Lab in the Wild, participants recruited from social media platforms are not compensated for their time but intrinsically motivated. CrowdPlay supports user choice of individual games and tasks through a modern UI resembling a "game library", and uses real-time statistics to restate task instructions as "challenges". Figure 5 (left) shows a screenshot of the "game library" feature.

Table 3: Un-Normalized performance measures for baselines. SI - SpaceInvaders, RR - RiverRaid, L-left play, R - right play, IO - inside out , row - row by row. Beamrider had not converged at the time of this submission and hence not included in the average normalized performance computation.

	Baselines					
Tasks	**BC**	**BCQ**	**CQL**	**IQN**	**DQN**	**SAC**
SI-L	113.11	123.3	5	0	56.3	285
SI-R	98	5.6	0	0	161.4	168.3
SI-IO	202	68.6	45.9	2.5	21.4	177.5
SI-OI	109.7	35.5	0	0	0	285
SI-row	8.8	22.6	0	0	116.3	0
SI-regular	124.45	17.5	0	18.7	1.5	216.4
RR-L	1096.7	724.5	360	0	336.81	295
RR-regular	599	539	0	0	39.8	1130
Q*Bert	0	85.1	0	0	0	0.25
BeamRider*	25.6	158.75	0	0	103.8	25.61

Table 4: Normalized performance measures for baselines. SI - SpaceInvaders, RR - RiverRaid, L-left play, R - right play, IO - inside out , row - row by row. Beamrider had not converged at the time of this submission and hence not included in the average normalized performance computation.

	Baselines					
Tasks	**BC**	**BCQ**	**CQL**	**IQN**	**DQN**	**SAC**
SI-L	-22.83	-18.35	-70.42	-72.62	-47.84	52.81
SI-R	-14.14	-33.64	-34.82	-34.82	-0.75	0.69
SI-IO	11.47	-30.60	-37.80	-51.58	-45.58	3.96
SI-OI	-9.82	-23.0	-29.30	-29.30	-29.30	21.31
SI-row	-33.8	-30.82	-35.71	-35.71	-10.54	-35.71
SI-regular	-2.5	-9.09	-10.17	-9.02	-10.08	3.17
RR-L	-5.31	-21.33	-37.03	-52.54	-38.03	-39.83
RR-regular	-18.87	-20.69	-37.08	-37.08	-35.87	-2.73
Q*Bert	-0.22	0.32	-0.22	-0.22	-0.22	0.59
BeamRider*	-33.57	-23.52	-35.51	-35.51	-27.67	-33.57

B BENCHMARK EXPERIMENT DETAILS

We performed all the experiments using the open source implementations of the algorithms that are provided as part of d3rlpy[3] library. We follow the default training parameters for all the algorithms and run each algorithm for 1M iterations before reporting the score. For the normalized score, we consider best score as the one obtained by a human on the particular task. Random score is the average over 5 runs of random policy for each task and we comput the normalized score as: Normalized Score $= 100 * \frac{AlgoScore - RandomScore}{BestHumanScore - RandomScore}$

[3] https://github.com/takuseno/d3rlpy