OFFLINE HYPERPARAMETER SELECTION FOR OFFLINE REINFORCEMENT LEARNING

Tom Le Paine*, **Cosmin Paduraru***, **Andrea Michi**, **Caglar Gulcehre**, **Konrad Żołna**, **Alexander Novikov**, **Ziyu Wang**[G], **Nando de Freitas**
DeepMind, Google[G]

ABSTRACT

Offline reinforcement learning (ORL), also known as RL from logged data or batch RL, is an important avenue for deploying RL in real-world scenarios, like robotics. However, existing hyperparameter selection methods for ORL break the offline assumption because they evaluate policies for each hyperparameter setting in the environment. This online execution is often infeasible and hence undermines the main aim of ORL. To make progress, this work advances the study of *offline hyperparameter selection* (OHS) methods, which select the best policy from a set of many policies trained with different hyperparameters, using only logged data. Through large-scale empirical evaluation we show that: (1) ORL algorithms are not robust to hyperparameter choices, (2) factors such as the ORL algorithm and method for estimating Q values can have a big impact on hyperparameter selection, and (3) when we control those factors carefully, we can reliably rank policies across hyperparameter choices, and therefore choose policies which are close to the best policy in the set. Overall, our results present an optimistic view that OHS is within reach, even in challenging control tasks with pixel observations, high dimensional action spaces, and long horizon.

offline reinforcement learning, hyperparameter selection, control

1 INTRODUCTION

The desire to apply reinforcement learning to real-world problems has prompted renewed interest in *offline reinforcement learning* (ORL), i.e. methods that can learn a policy from logged data (Fujimoto et al., 2019; Levine et al., 2020). These methods are useful when it is challenging, dangerous or expensive to execute a policy on the environment, for example in self-driving cars, health-care (Futoma et al., 2020), dialogue (Jaques et al., 2019), and robotics (Cabi et al., 2020).

Despite strong results, the quality of different ORL algorithms depends heavily on hyperparameter choice. The standard way of performing hyperparameter selection in RL is to evaluate policies online by interacting with the environment. In contrast, hyperparameter selection for ORL should only involve logged data, where direct interaction with the environment is not allowed, as pointed out, for instance, by Wu et al. (2019). Therefore, we must devise offline statistics to rank policies produced by different hyperparameter settings.

This paper presents a thorough empirical study of offline hyperparameter selection for ORL. In particular, this study: (1) presents several methods for ORL hyperparameter selection that only use logged data, (2) introduces simple and scalable evaluation metrics to assess the various ORL hyperparameter selection methods, (3) incorporates challenging domains with high-dimensional action and observation spaces, and long time horizons, and (4) focuses on common important hyperparameters associated with the model architecture, optimizer, and loss function.

Our experiments confirm that ORL algorithms are not robust to hyperparameter choices, strengthening the case for developing reliable offline hyperparameter selection methods. Additionally, through our experiments we identify three important choices that affect hyperparameter selection:

*indicates joint first authorship, both authors equally contributed to this project.

- The choice of ORL algorithm: We find that algorithms that encourage policies to stay close to the behavior policy are easier to evaluate and rank.
- The choice of Q estimator: We find Q values estimated by the OPE algorithm we use, Fitted Q-Evaluation, to be more accurate than ORL estimates.
- The choice of statistic for summarizing the quality of a policy: The average critic value of the initial states works better than alternatives.

2 Offline Hyperparameter Selection

ORL enables RL in real scenarios where only logged data can be used. Thus, hyperparameter selection for ORL should follow the same assumption (see Figure 1).

Offline hyperparameter selection (OHS) is closely related to offline policy evaluation (OPE), which focuses on estimating a value function based on offline data. [1] There are two salient differences between OHS and OPE. First, in OHS there is a known relationship between the data and the policies which may be leveraged to simplify the problem, whereas in OPE this relationship is generally unknown. Second, in OHS we mainly care about picking the best policy (or close to the best) from a set, not precisely assessing the quality of a policy as OPE aims to do. As a result, OHS and OPE may focus on optimizing different performance metrics (see Figure 2).

This paper is concerned with hyperparameter *selection*, which is subtly different from hyperparameter tuning. Hyperparameter selection refers to picking the best out of a set of given policies that were trained with different hyperparameters, whereas tuning includes both selection and a strategy for searching the hyperparameter space.

We use standard RL nomenclature (Sutton & Barto, 1998). In short, the agent policy takes s as input states to generate actions a which yields rewards r that are discounted using γ. They all can be subscripted with a discrete timestep $t \geq 0$ and batched. The policy π_ϕ and critic Q_θ are parametrized by ϕ and θ, respectively.

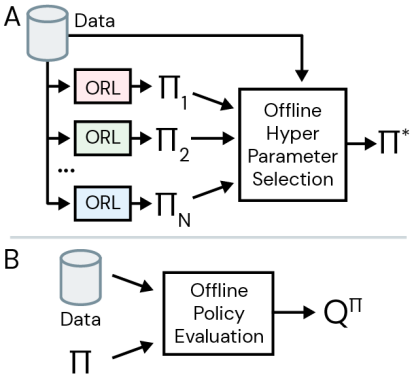

Figure 1: **Offline hyperparameter selection compared to offline policy evaluation.** (A) In offline hyperparameter selection, we learn a set of N policies using ORL with different hyperparameters, and attempt to pick the best policy π^*. For both learning and hyperparameter selection, we have access to the same logged data, and *not* to online interactions with the environment. (B) Offline Policy Evaluation is a closely related problem where the goal is to estimate the value of a policy given data.

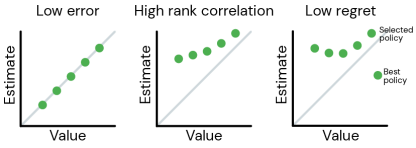

Figure 2: **Evaluating performance of offline hyperparameter selection.** Each point represents a policy in the set, plotted according to its actual and estimated value. OPE aims to minimize the error between the value and the estimate (moving points closer to the diagonal). In contrast, in OHS it is sufficient to (i) rank the policies as measured by spearman rank correlation, or (ii) select a policy whose value is close to the value of the best policy (i.e. low regret).

2.1 Offline Statistics for Policy Ranking

The challenge when performing OHS is to rank several policies using statistics computed solely from offline data. We envision the following workflow to apply OHS in practice: (1) Train ORL policies using several different hyperparameter settings, (2) for each policy, compute scalar statistics summarizing the policy's performance (without interacting with the environment), and (3) pick the top k best policies according to the summary statistics.

The statistics considered in this paper are based on the estimated value functions (also referred to as "critics") for policies trained by ORL with different hyperparameter settings. We obtain these critics in two ways.

[1] Offline policy evaluation is in turn very closely related to off-policy evaluation. Both estimate the value function of an evaluation policy using data produced by a different policy, typically called the behavior policy, but off-policy evaluation can be performed online. In this paper, since we are only concerned with the offline setting, they are equivalent and we use the same acronym (OPE) for both.

ORL We can simply use the critic learned during ORL training for that hyperparameter setting. For methods which do not usually leverage a critic, we additionally train a critic which does not impact the training but can be used to obtain statistics, see Section 3.2.

OPE We use Fitted Q Evaluation (FQE, see Section 3.3) to retrain a critic for the policy generated by the ORL algorithm.

We then compute scalar values for the purpose of ranking policies by calculating a statistic based on the critic Q_θ, the evaluation policy π_e, and the dataset $\mathcal{D}$. We do this in one of two ways:

- $\hat{V}(s_0)$ We use an estimate of the expected value of the evaluation policy for the initial state distribution $\mathbb{E}_{s_0 \sim \mathcal{D}}[Q_\theta(s_0, \pi_e(a)]$. This is an estimate of what we care about, i.e. the value we would achieve by running the policy in the environment from initial states.
- **Soft OPC** We use the soft off-policy classification (OPC) statistic proposed in Irpan et al. (2019). This statistic can be written as $\mathbb{E}_{(s,a)\sim\mathcal{D},success}[Q_\theta(s,a)] - \mathbb{E}_{(s,a)\sim\mathcal{D}}[Q_\theta(s,a)]$. Here success indicates whether (s, a) is part of a successful trajectory, namely one whose return is above a certain threshold; we try different values for this threshold in our experiments and pick the best one.

We also considered two additional statistics: the average Q across all states, which performed similarly but slightly worse than $\hat{V}(s_0)$, as well as the average TD error, which clearly underperformed the main statistics considered. We defer these additional statistics to Appendix C.

2.2 Metrics for Evaluating Offline Hyperparameter Selection

We use evaluation metrics that aim to capture how useful different statistics are for ranking multiple policies and selecting the best one(s). To compute the metrics, we first obtain a reliable estimate of the actual expected discounted return for each policy when starting in the environment's initial state distribution, by running that policy in the actual environment. This estimate will be used as the ground truth, and will be referred to as the **actual value**. We then compute the following metrics:

- **Spearman's rank correlation** First compute the rank values of the different policies according to both the summary statistics and the actual values. Spearman's rank correlation is the Pearson correlation between the two sets of rank values.
- **Regret @ k** First compute the top-k set, i.e. the k policies with the highest summary statistic values. Regret @ k is the difference between the actual value of the best policy in the entire set, and the actual value of the best policy in the top-k set. This metric aims to answer the question "If we were able to run policies corresponding to k hyperparameter settings in the actual environment and get reliable estimates for their values, how far would the best in the set we picked be from the best of all hyperparameter settings considered?".
- **Absolute error** The absolute value of the difference between the statistic $\hat{V}(s_0)$ and the actual values. This does not measure ranking quality directly, but we include it here because zero absolute error would correspond to perfect ranking, and because it is a standard measure in the OPE literature.

3 Experimental Setup

In the previous section, we described offline hyperparameter selection, and the statistics and metrics we consider. In this section, we describe additional elements of the experimental setup including the tasks, ORL algorithms, OPE algorithms, and hyperparameters considered.

3.1 Tasks

Real-world problems are diverse, spanning a range of challenging properties. We want to verify offline hyperparameter selection for high dimensional action space, observation space, long time horizon problems. To achieve this end, we use ten tasks spanning two domains included in the RL Unplugged (Gulcehre et al., 2020) benchmark suite, as well as one robotic manipulation domain

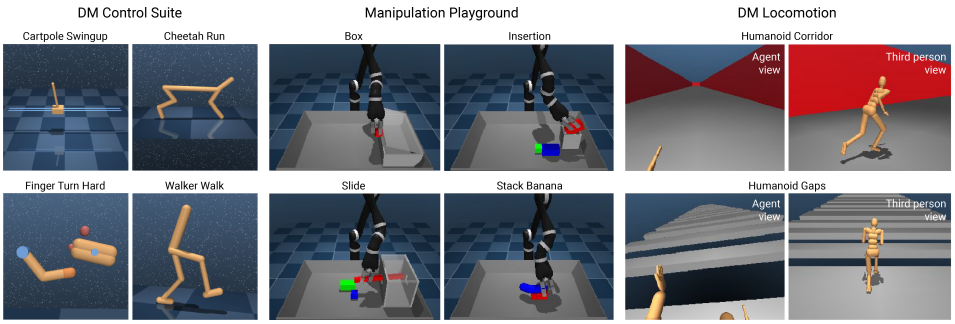

Figure 3: **Tasks considered.** We consider tasks from three continuous control domains: 1) DM Control Suite involves low dimensional action spaces from features of the MDP state, 2) Manipulation playground involves low dimensional action spaces from proprioceptive features of a Jaco arm, as well as visual representation of the scene, 3) DM Locomotion involves control of a high action space humanoid avatar, from visuals provided by an egocentric camera controlled by the policy.

(Wang et al., 2020). In Figure 3, we illustrate some of those tasks and below we describe them in detail. For additional details about the datasets we used see Appendix F.

DM Control Suite A set of continuous control tasks introduced by (Tassa et al., 2018) implemented in MuJoCo (Todorov et al., 2012). We choose four tasks with various levels of difficulty from the subset included in Gulcehre et al. (2020). For all tasks in this domain we use a feature representation of the MDP state, including proprioceptive information such as joint positions and velocities.

Manipulation tasks These tasks require continuous control of a Kinova Jaco robotic arm with 9 degrees of freedom (simulated in MuJoCo (Todorov et al., 2012)). The tasks include manipluation problems such as picking up a block and placing it in a box. We use joint velocity control (at 20HZ) of all 6 arm joints and the 3 joints of the hand. The agent observes the proprioceptive features directly, but can only infer the objects on the table from pixel observations. Two camera views of size 64×64 are provided: one frontal camera covering the whole scene, and an in-hand camera for closeup of the objects. The episodes are of length 400 and the reward function is binary depending on whether the task is successfully executed.

DM Locomotion A set of continuous control tasks which involve controlling a 56 degrees of freedom humanoid avatar, resulting in a large action spaces (Tassa et al., 2020). For all tasks, we learn directly from large observation spaces (i.e. 64x64 RGB images). These images are generated by an egocentric camera under the control of the policy. We focus on humanoid corridor and humanoid gaps, which are difficult tasks but do not require long-term memory.

3.2 ORL Algorithms

Our experiments select among policies produced using different hyperparameter settings for three offline reinforcement learning algorithms listed below.

Behavior Cloning (BC; Pomerleau (1989)) BC's policy objective attempts to match the actions from the behavior data. Standard behavior cloning does not train a value function, but we do train a value function alongside the policy to enable downstream policy evaluation. We note that the training procedure is agnostic to the critic trained.

Critic Regularized Regression (CRR; Wang et al. (2020)) The policy objective of CRR attempts to match the actions from the behavior data, while also preferring actions with high value estimates. This encourages the policy to be close to the behavior policy for some, but not all states.

Table 1 — **Policy updates for considered algorithms.** BC and CRR regress to actions logged in the data while D4PG relies on critic estimates.

Algorithm	Policy Update
BC	$-\nabla_\phi \log \pi_\phi(a_t \vert s_t)$
CRR	$-\nabla_\phi \log \pi_\phi(a_t \vert s_t) \cdot w^*$
D4PG	$\nabla_\phi Q_\theta(s_t, \pi_\phi(s_t))$
FQE	None

$^*w$ *reflects CRR weighting as in (Wang et al., 2020)*

Distributed Distributional Deep Deterministic Policy Gradient (D4PG; Barth-Maron et al. (2018)) D4PG's policy objective directly optimizes the critic. As a result, there is no regularization towards the behavior policy.

By design, all the algorithms share the same value update, which is

$$-\nabla_\theta d(Q_\theta(s_t, a_t), r_t + \gamma \mathbb{E}_{a\sim\pi_\phi(s_{t+1})} Q_\theta(s_{t+1}, a)), \tag{1}$$

where d is a divergence measure. As the value update is the same for all the algorithms, they are defined by their policy updates which are shown in Table 1. This allows us to isolate the effect of the policy update on downstream policy evaluation.

Finally, we want to reiterate that two of the algorithms, BC and CRR, encourage the policy to stay close to the behavior policy, whereas D4PG does not, allowing the policy to freely optimize against the critic. As shown in our experiments (see Section 4), this characteristic strongly influences the OHS accuracy.

3.3 Offline Policy Evaluation Algorithms

We re-evaluate the policies generated by the ORL algorithms described above using FQE.

Fitted Q Evaluation (FQE; Le et al. (2019)) Our FQE algorithm employs the same value function updates as the above ORL methods but keeps the policy fixed. Pseudocode for it can be found in Algorithm 1, and a simplified version of the code we use can be found in Appendix E.

Algorithm 1 Fitted Q Evaluation

Input: Dataset $\mathcal{D}$, policy π_e to evaluate
For $n_{updates}$ do
  Sample $\{s_i, a_i, r_i, s'_i\}_{i=1}^{batch\_size}$ from $\mathcal{D}$
  Update critic according to Eq. 1

FQE was shown to work well in a recent suite of experiments on relatively simple problems (Voloshin et al., 2019). Using it allows us to interrogate how much re-evaluation with the same objective and dataset improves policy evaluation. In our experiments, we assume that the ORL algorithms and FQE have access to the same data.

There are many possible OPE algorithms we could try, as discussed in Section 5. We focus on Fitted Q Evaluation due to its simplicity and scalability. Other OPE methods have to solve difficult estimation problems, such as learning a transition model from pixels or computing importance sampling corrections for continuous actions. Fitted Q Evaluation foregoes estimating these complex quantities by directly estimating the value function of the policy being evaluated.

3.4 Hyperparameters and Other Implementation Details

For each task, we train policies using the hyperparameters described in Table 2, resulting in 256 policies per task (64 BC, 128 CRR, 64 D4PG). We considered hyperparameters that affect the model architecture (hidden size, num blocks[2]), the optimizer (learning rate and learner steps), and loss function (algorithm and loss term beta), since these are known to be important for many machine learning problems. The search space in grid search grows rapidly with respect to the number of hyperparameters considered and unique hyperparameter settings. Thus, we tried to choose a small representative set that is reasonably broad in terms of performance. Our implementations are based on open-sourced D4PG implementation from Acme (Hoffman et al., 2020).

Table 2 — **Hyperparameters considered.** We consider parameters specific to the model architecture, the optimizer, and loss function.

Hyperparameter	Values
Hidden size	64, 1024
Num blocks	1, 5
Learning rate	0.001, 0.00001
Learner steps	range(50k, 250k, 25k)
Algorithms	BC, CRR, D4PG
Beta (for CRR)	0.1, 10

Each policy comes with an associated critic Q_θ, which we use to calculate ORL statistics. In addition, we re-evaluate each policy and hyperparameter choice using FQE for each combination of the hyperparameters in Table 2, meaning that we run the FQE algorithm 256 times per task. We use these resulting critics to generate the OPE statistics. Finally, to obtain the actual value, we run each policy in the environment for 100 episodes. In the experiments we compare the actual values to the ORL and OPE statistics.

[2]The CRR paper (Wang et al., 2020) uses blocks composed of two linear layers followed by layer-norm and residual connections. We used the same architecture.

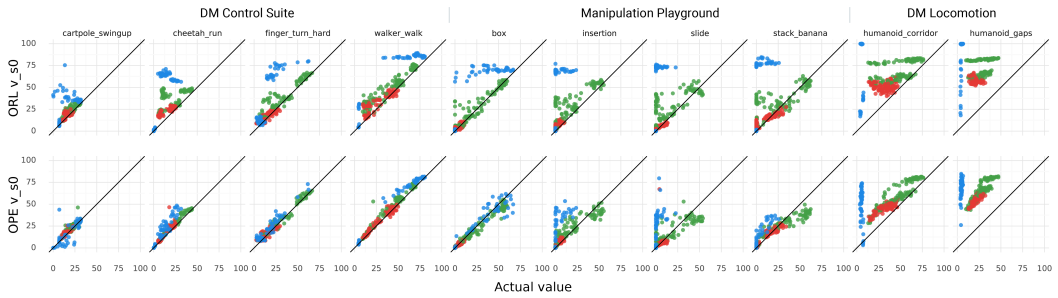

Figure 4: **Value estimates vs actual values.** Each point represents a policy trained using different hyperparameters including different algorithms (■ BC, ■ CRR, ■ D4PG). (top) Value estimates from ORL algorithms. Notice nearly all values are over-estimated (i.e. lay above the diagonal) to some degree. D4PG over-estimates the most, followed by CRR, then BC. (bottom) Value estimates from re-evaluating policies using offline policy evaluation, specifically FQE. Re-evaluation significantly reduces over-estimating in these domains, though D4PG deviates the most from the actual values.

In this paper, we use distributional critics (Bellemare et al., 2017) for all our algorithms, including FQE. This means that the value function Q_θ in Eq. 1 is represented as a discrete distribution, and the discrepancy measure d is the cross-entropy between the two distributions, as in Barth-Maron et al. (2018) and Wang et al. (2020). In Appendix D we compare FQE estimates with and without a distributional critic and find they are similar.

4 Results

Our experiments confirm that hyperparameter choice does play an important role in the performance of the ORL algorithms we use on our set of tasks (see for example the range of actual values (x axis) on the graphs in Figure 4). The results presented in this section aim to shed light on the conditions under which the statistics in Section 2.1 rank these hyperparameter choices well.

4.1 Overestimation

In the top row of Figure 4 and 5 we compare the actual values against the ORL $\hat{V}(s_0)$ statistics. We show that, on all tasks, ORL's $\hat{V}(s_0)$ overestimates the value. In extreme cases (e.g. humanoid environments; see Figure 4 (top)), according to the ORL statistics, D4PG deceptively looks to produce the best policy, while actually being the worst.

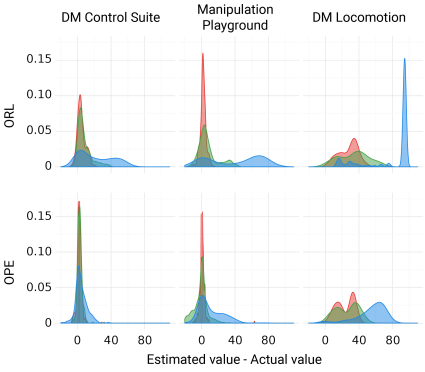

Figure 5: **Distribution of over-estimation.** Summary of the distribution of over-estimation for each algorithm (■ BC, ■ CRR, ■ D4PG) and task domain. Re-evaluation by OPE (bottom plot) significantly reduces over-estimation across all domains and algorithms. Y-axis is probability density.

We find a clear over-estimation trend – statistics tend to over-estimate the most on D4PG, followed by CRR, followed by BC. Again, we note that BC and CRR attempt to produce policies that are similar to the behavior policy, whereas D4PG does not. This may make it easier to estimate the value of the policies they produce, given only the behavior data. In terms of task domains, statistics tend to over-estimate the most on DM Locomotion, followed by Manipulation Playground, followed by DM Control Suite.

In the bottom row of Figure 4 and 5, we compare the actual values against the OPE $\hat{V}(s_0)$ statistics. We show that re-evaluation using FQE significantly reduces over-estimation across all algorithms and tasks. Unfortunately, the over-estimation trends described above remain. Specifically, for D4PG on DM Locomotion, the statistic still over-estimates significantly.

Finally, the plots in Figure 4 suggest better performing policies overestimate less. This finding highlights the importance of reducing overestimation for ORL to learn better policies.

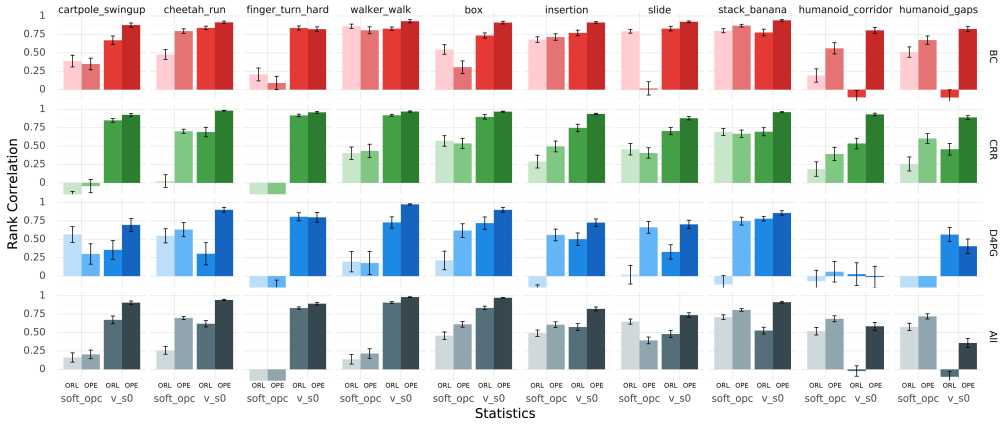

Figure 6: **Rank correlation within algorithm and across all algorithms.** We compare the rank correlation between the actual value and two policy statistics: $\hat{V}(s_0)$ and Soft OPC (see additional statistics in Appendix C). We use both ORL and OPE critics. Note that each set of bars (same color and tone) represents an offline hyperparameter selection procedure with different design choices. Prior work Irpan et al. (2019) is most similar to ORL Soft OPC on D4PG. We find the best set of design choices OPE $\hat{V}(s_0)$ on CRR to work significantly better.

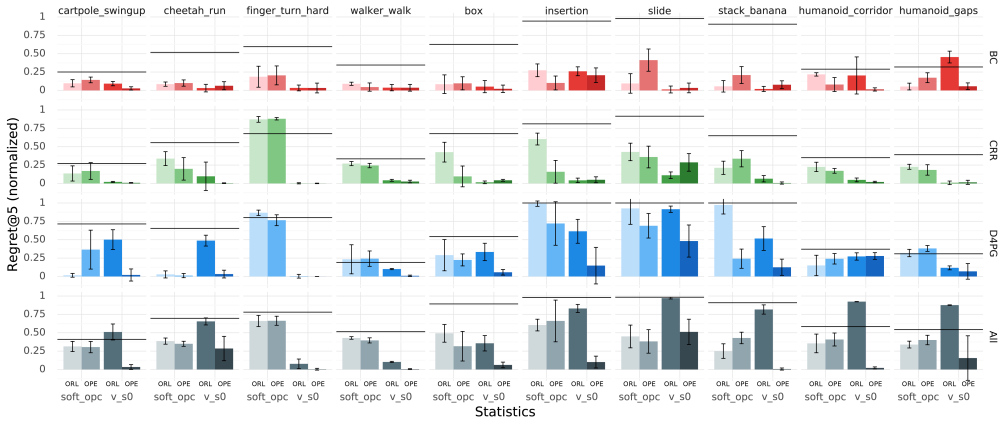

Figure 7: **Regret@5 within algorithm and across all algorithms.** We compare the normalized regret@5 between the actual value and various policy statistics from ORL and OPE critics ($\hat{V}(s_0)$, Soft OPC, additional statistics in Appendix C). We normalize by the value of the best policy. The black horizontal line indicates the regret associated with the policy of median value. The regret follows similar trends to rank correlation. Note that prior work Irpan et al. (2019) is most similar to ORL Soft OPC on D4PG. We find the best set of design choices ORL/OPE $\hat{V}(s_0)$ on CRR to work significantly better, with low regret across all tasks, much lower than the regret associated with picking the policy with median value, and often close to zero.

4.2 Ranking Quality

In Figure 6 we compute the rank correlation between the actual values and various statistics, broken down by ORL algorithm, and across all algorithms. The ORL $\hat{V}(s_0)$ statistic has better rank correlation for BC and CRR (algorithms that encourage the policy to stay close to the behavior policy) than for D4PG. The ORL $\hat{V}(s_0)$ statistic also performs better on environments which showed less overestimation like DM Control Suite, and worse on environments with more overestimation like DM Locomotion. For the hardest combination, D4PG on the DM Locomotion tasks, the rank correlation is quite low. Because D4PG is hard to rank in this setting, it is also hard to rank policies across all algorithms.

The same general trends hold for the OPE $\hat{V}(s_0)$ statistic, but it has higher correlation than ORL $\hat{V}(s_0)$ across the board. Especially for BC and CRR on the DM Locomotion tasks. For BC and CRR, OPE $\hat{V}(s_0)$ rank correlation is above 0.9 for most tasks. But for D4PG and across all algorithms on

DM Locomotion is still quite low. In Appendix B we analyze how OPE $\hat{V}(s_0)$ performance varies with number of FQE learner steps.

Similar trends for ORL and OPE $\hat{V}(s_0)$ statistics can be seen in the regret plots in Figure 7. BC and CRR have generally low regret across all tasks, usually much lower than the regret associated with picking the policy with median value, and often close to zero. Regret for D4PG is often low, and usually better than the median choice. For D4PG the regret for OPE $\hat{V}(s_0)$ tends to be lower than for ORL $\hat{V}(s_0)$.

Figures 6 and 7 indicate that ranking based on $\hat{V}(s_0)$ is preferable to ranking based on the Soft OPC statistic in terms of both ranking correlation and regret. For instance, OPE $\hat{V}(s_0)$ has higher rank correlation than OPE Soft OPC for nearly all tasks. It is not clear why this is the case. One potential explanation is Soft OPC depends on the value estimates along unsuccessful trajectories. It is possible the learned policies deviate from the behavior policy greatly on these states, and the value estimates end up being poor, making the difference in value between successful vs unsuccessful states less meaningful. This may be exacerbated by the challenging properties of these tasks, such as the high action dimension or long time horizon.

5 Related Work

There is a large body of work on OPE including importance sampling methods (Precup, 2000; Precup et al., 2001; Munos et al., 2016) marginal importance sampling methods (Liu et al., 2018; Nachum et al., 2019; Uehara et al., 2020), model-based methods (Mannor et al., 2004), and doubly robust methods which combine importance sampling and model-based methods (Dudík et al., 2011; Jiang & Li, 2016; Thomas & Brunskill, 2016; Farajtabar et al., 2018). Importance sampling based approaches can be difficult to apply in domains with large continuous action spaces, and model based approaches can be difficult to apply in domains with high dimensional observation spaces. This makes pixel-based robotics applications particularly difficult for a broad range of OPE methods. One approach to OPE that avoids both importance sampling corrections and learning transition models is Fitted Q Evaluation (Le et al., 2019). A comprehensive empirical study of OPE methods was carried out in Voloshin et al. (2019). The study considered most of the aforementioned OPE methods and finds FQE to be surprisingly effective despite its simplicity. Compared to this work, Voloshin et al. (2019) does not address model selection, only standard policy evaluation using MSE as the evaluation metric. Additionally, they consider simpler environments and datasets.

Hyperparameter tuning has been an essential tool for improving the performance of algorithms on many tasks and domains (Bergstra & Bengio, 2012; Snoek et al., 2012; Jaderberg et al., 2017; Melis et al., 2018). However, offline hyperparameter tuning for ORL has received relatively little attention. Farahmand & Szepesvári (2011) have proposed BerMin for offline model selection, and they investigated the convergence and other theoretical properties of it. Their method, however, resolves around the Bellman error which does not correlate well with policy performance in complicated domains. Irpan et al. (2019) proposed Off-Policy Classification (OPC) and evaluated it using rank correlation on goal-directed continuous control tasks. Their evaluation scheme is similar to ours, but we 1) focus on more thorough evaluation in more challenging simulated domains, and 2) extend the evaluation to examine several important factors such as the provenance of the policies, provenance of the q estimator, and additional offline statistics. More recently, Gulcehre et al. (2020) have proposed protocols for evaluating ORL methods that involve evaluation on a set of tasks that are not available for online hyperparameter tuning.

6 Conclusions

We provide evidence that by carefully considering the choice of ORL algorithm, Q estimator, and statistic, we can achieve a strong strategy for offline hyperparameter selection across challenging tasks. In particular, we find that using algorithms that encourage policies to stay close to the behavior policy such as CRR, re-estimating the Q value using FQE, and using $\hat{V}(s_0)$ as our ranking statistic is sufficient for performing offline hyperparameter selection in the tasks we considered. This is true even in the DM Locomotion tasks, which require control of a 56 degrees of freedom humanoid avatar from visuals provided by an egocentric camera.

REFERENCES

Gabriel Barth-Maron, Matthew W. Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. In *International Conference on Learning Representations*, 2018.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pp. 449–458, 2017.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(1):281–305, 2012.

Serkan Cabi, Sergio Gómez Colmenarejo, Alexander Novikov, Ksenia Konyushkova, Scott Reed, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerik, Oleg Sushkov, David Barker, Jonathan Scholz, Misha Denil, Nando de Freitas, and Ziyu Wang. Scaling data-driven robotics with reward sketching and batch reinforcement learning. In *Robotics: Science and Systems*, 2020.

Miroslav Dudík, John Langford, and Lihong Li. Doubly robust policy evaluation and learning. *CoRR*, abs/1103.4601, 2011.

Amir-massoud Farahmand and Csaba Szepesvári. Model selection in reinforcement learning. *Machine learning*, 85(3):299–332, 2011.

Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. More robust doubly robust off-policy evaluation. In *International Conference on Machine Learning*, pp. 1447–1456, 2018.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, volume 97, pp. 2052–2062, 2019.

Joseph Futoma, Michael C. Hughes, and Finale Doshi-Velez. POPCORN: partially observed prediction constrained reinforcement learning. In *Artificial Intelligence and Statistics*, pp. 3578–3588, 2020.

Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gómez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel Mankowitz, Cosmin Paduraru, Gabriel Dulac-Arnold, Jerry Li, Mohammad Norouzi, Matt Hoffman, Nachum Ofir, Tucker George, Nicolas Heess, and Nando de Freitas. Rl unplugged: Benchmarks for offline reinforcement learning. *Preprint arXiv:2006.13888*, 2020.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A research framework for distributed reinforcement learning. *Preprint arXiv:2006.00979*, 2020.

Alexander Irpan, Kanishka Rao, Konstantinos Bousmalis, Chris Harris, Julian Ibarz, and Sergey Levine. Off-policy evaluation via off-policy classification. In *Advances in Neural Information Processing Systems*, pp. 5437–5448, 2019.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *Preprint arXiv:1711.09846*, 2017.

Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *Preprint arXiv:1907.00456*, 2019.

Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pp. 652–661, 2016.

Hoang Le, Cameron Voloshin, and Yisong Yue. Batch policy learning under constraints. In *International Conference on Machine Learning*, volume 97, pp. 3703–3712, 2019.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *Preprint arXiv:2005.01643*, 2020.

Qiang Liu, Lihong Li, Ziyang Tang, and Dengyong Zhou. Breaking the curse of horizon: Infinite-horizon off-policy estimation. In *Advances in Neural Information Processing Systems*, 2018.

Shie Mannor, Duncan Simester, Peng Sun, and John N Tsitsiklis. Bias and variance in value function estimation. In *International Conference on Machine Learning*, pp. 72, 2004.

Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*, 2018.

Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. Neural probabilistic motor primitives for humanoid control. In *International Conference on Learning Representations*, 2019.

Remi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 1054–1062. 2016.

Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. In *Advances in Neural Information Processing Systems*, 2019.

Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*, pp. 305–313, 1989.

Doina Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, pp. 80, 2000.

Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. Off-policy temporal-difference learning with function approximation. In *International Conference on Machine Learning*, pp. 417–424, 2001.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pp. 2951–2959, 2012.

Richard S. Sutton and Andrew G. Barto. Reinforcement learning: An introduction. *IEEE Trans. Neural Networks*, 1998.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite. *Preprint arXiv:1801.00690*, 2018.

Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm_control: Software and tasks for continuous control. *Preprint arXiv:2006.12983*, 2020.

Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pp. 2139–2148, 2016.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

Masatoshi Uehara, Jiawei Huang, and Nan Jiang. Minimax weight and q-function learning for off-policy evaluation. In *Proceedings of Machine Learning and Systems 2020*, pp. 1023–1032. 2020.

Cameron Voloshin, Hoang M. Le, Nan Jiang, and Yisong Yue. Empirical study of off-policy policy evaluation for reinforcement learning. *Preprint arXiv:1911.06854*, 2019.

Ziyu Wang, Alexander Novikov, Konrad Żołna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, and Nando de Freitas. Critic regularized regression. *Preprint arXiv:2006.15134*, 2020.

Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *Preprint arXiv:1911.11361*, 2019.

Appendix

A Absolute Error Results

In the main paper we provide over-estimation results, summarized for each task domain. In this section we show the absolute error for each task individually.

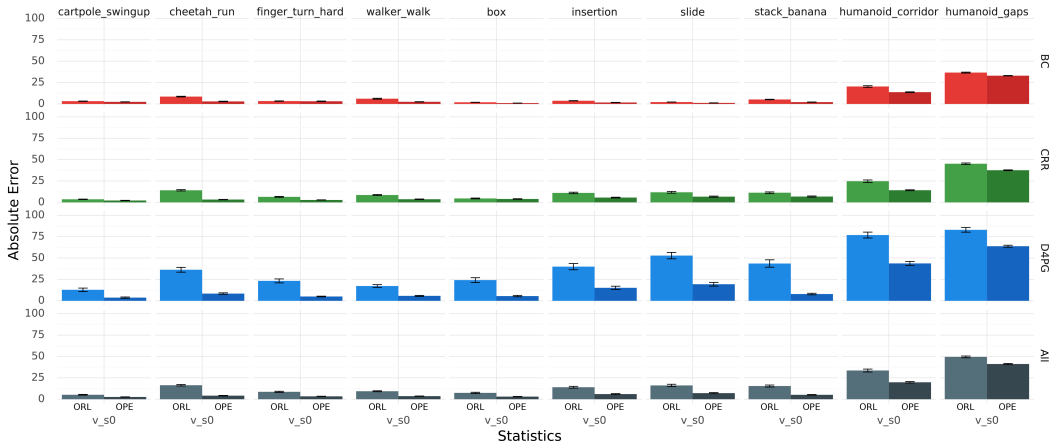

Figure 8: **Absolute Error.** We compute the absolute error between the actual values and V(s0) statistics, using the ORL and OPE critics. We find two main error trends: first in terms of algorithms, statistics tend to have highest error on D4PG, followed by CRR, followed by BC. Second in terms of task domains, statistics tend to have highest error on DM Locomotion, followed by Manipulation Playground, followed by DM Control Suite. And OPE statistics have lower error and ORL statistics.

B FQE Sensitivity to Its Own Hyperparameters

Our FQE implementation was based on an existing CRR implementation and we used its default hyperparameters for all tasks (hidden size = 1024, num blocks = 4, learning rate = 0.0001, learner steps = 250k). We did not perform an exhaustive investigation of FQE's sensitivity to hyperparameters, but we did look into its sensitivity to the number of learner steps on the control suite. Figure 9 suggests that FQE is not very sensitive to the number of learner steps and, in general, does not seem to get worse as it runs for longer. Finding a good way to tune FQE hyperparameters remains an open problem for future research.

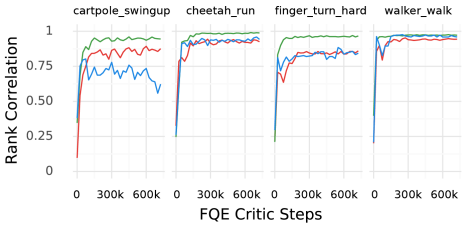

Figure 9: **Rank correlation vs FQE learner steps.** (■ BC, ■ CRR, ■ D4PG) FQE is relatively stable to the number of learner steps. Though D4PG on cartpole_swingup eventually diverges.

C Additional Ranking Results

In addition to $\hat{V}(s_0)$ and Soft OPC, we also considered two additional offline statstics:

- **avg_q** Use the expected value across all states in the dataset $\mathbb{E}_{s\sim\mathcal{D}}[Q_\theta(s,\pi(s))]$. This differs slightly from what we care about, it corresponds to running the policy from any state along a trajectory in the behavior data. Nevertheless this statistic performs similarly to $\hat{V}(s_0)$.
- **td_err** Use the average temporal difference error across all (s,a,r,s') tuples in dataset. $\mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}[r+\gamma Q_\theta(s',\pi(s'))-Q_\theta(s,a)]$. This statistic is more indicative of the quality of the critic than the quality of the policy, which may explain why it performs quite poorly for our purposes.

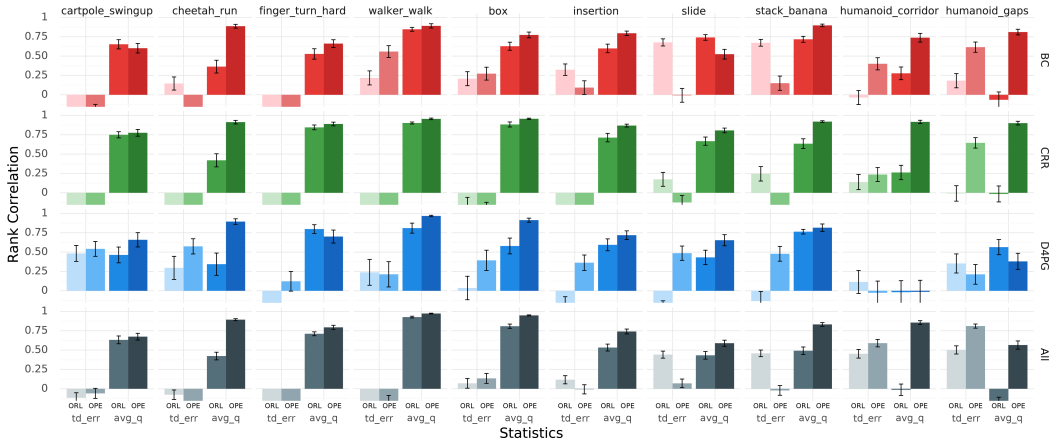

Figure 10: **Rank correlation within algorithm and across all algorithms cont'd.** We compare the rank correlation between the actual value and additional policy statistics from ORL and OPE critics (avg_q, td_err). In general, avg_q follows similar trends to $\hat{V}(s_0)$, but its slightly worse, and td_err performs quite poorly overall.

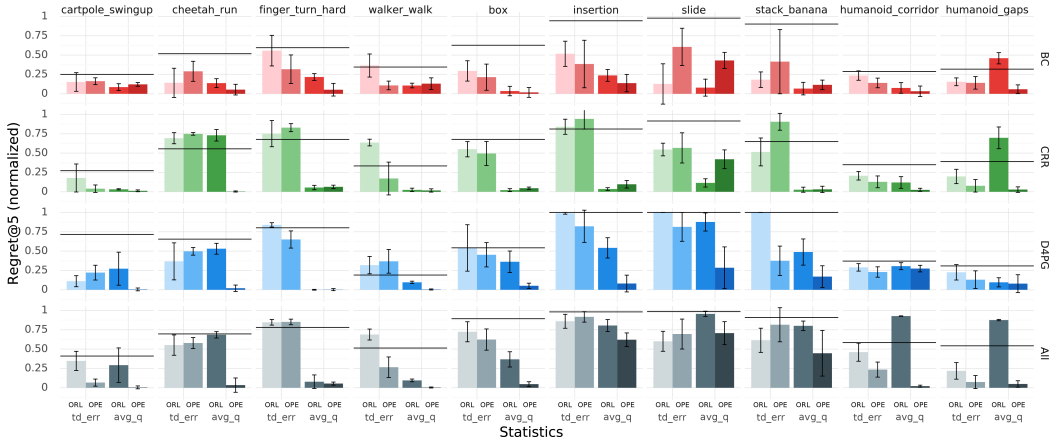

Figure 11: **Regret@5 within algorithm and across all algorithms cont'd.** We compare the normalized regret@5 between the actual value and additional policy statistics from ORL and OPE critics (avg_q, td_err). The regret follows similar trends to rank correlation. In general, avg_q follows similar trends to $\hat{V}(s_0)$, but its slightly worse, and td_err performs quite poorly overall.

D FITTED Q EVALUATION WITHOUT DISTRIBUTIONAL CRITIC

We aimed to keep the critic loss consistent for all experiments in the main paper. But some readers may be curious how FQE would perform without a distributional critic. We re-ran our FQE evaluation with and without a distributional critic on the DM Control Suite tasks.

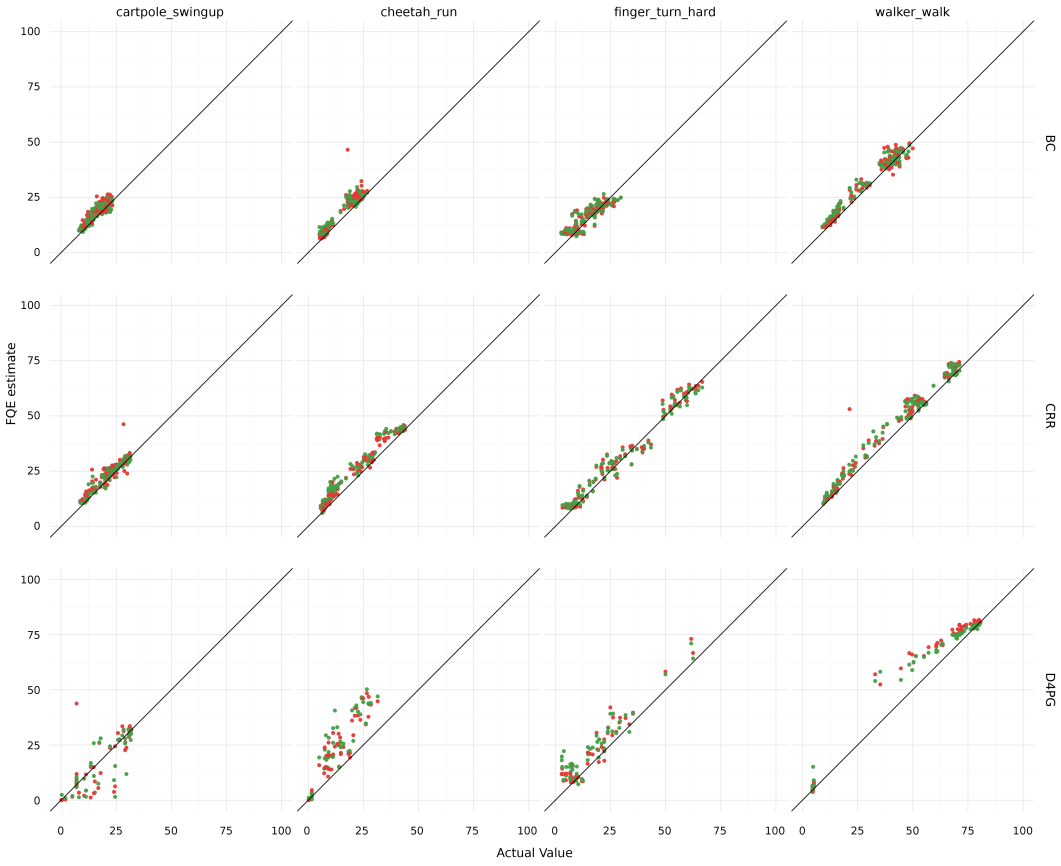

Figure 12: **Comparing FQE estimates with and without distributional critics.** (■ With, ■ Without) Overall estimates fall within a similar range. FQE with a distributional critic has a few outliers with high values that FQE without does not. On closer inspection we found these corresponded to experiments that were terminated early.

E FITTED Q EVALUATION CODE

We wrote our FQE code using TensorFlow 2 and Acme (Hoffman et al., 2020). Listing 1 is a simplified version of the code we used.

Listing 1: **Simplified code for of Fitted Q Evaluation.** The code is functional despite its simplicity.

```
# Copyright 2020 DeepMind Technologies Limited.
# SPDX-License-Identifier: Apache-2.0

discount = 0.99
target_update_period = 100

num_steps = 0

for o_tm1, a_tm1, d_t, is_terminal, o_t in dataset:
  q_t = target_critic_network(o_t, policy_network(o_t))

  # Operations that will be differentiated should be executed in
  # this context.
  with tf.GradientTape() as tape:
    q_tm1 = critic_network(o_tm1, a_tm1)

    # Use 0 discount at terminal states.
    curr_discount = 0.0 if is_terminal else discount
    critic_loss = losses.categorical(q_tm1, r_t, curr_discount, q_t)

  # Get trainable variables.
  variables = critic_network.trainable_variables

  # Compute gradients.
  gradients = tape.gradient(critic_loss, variables)

  # Apply gradients.
  optimizer.apply(gradients, variables)

  # Update online -> target parameters if necessary.
  source_variables = critic_network.variables
  target_variables = target_critic_network.variables
  if num_steps % target_update_period == 0:
    for src, dest in zip(source_variables, target_variables):
      dest.assign(src)
  num_steps += 1
```

F DATASET DETAILS

In this section, we provide details regarding the datasets used in this paper. For the sizes of all datasets used, please refer to Table 3.

Table 3 — **Dataset sizes in number of episodes.**

Task	No. Episodes
Cartpole Swingup	100
Cheetah Run	2000
Finger Turn Hard	2000
Walker Walk	800
Box	8000
Stack Banana	8000
Insertion	8000
Slide	8000
Humanoid Corridor	16000
Humanoid Gaps	16000

DM Control Suite We largely follow the procedures of generating data as described in Gulcehre et al. (2020). All datasets used in the DM control suite domains are generated by 3 independent runs of a D4PG agent. Episodes from the entire training run is saved to increase diversity. Unlike in Gulcehre et al. (2020), we do not further filter out successful episodes and the size of the datasets used in this paper is larger than that in Gulcehre et al. (2020).

Manipulation tasks These datasets are the same as the robotics datasets used in Wang et al. (2020). The dataset for each task is generated from 3 independent runs of a D4PGfD agent[3] where 100 human demonstrations are used for each task to assist with exploration. The dataset contains 8000 episodes from the entire training process and thus consists of both successful and unsuccessful episodes.

DM Locomotion We largely adhere to the procedures of generating data as described in Gulcehre et al. (2020). For each task, three policies are trained following Merel et al. (2019). Episodes from the entire training runs are saved and sub-sampled to include both successful and failed episodes. Unlike in Gulcehre et al. (2020), we do not further filter out successful episodes and the size of the datasets used in this paper is larger than that in Gulcehre et al. (2020).

[3]D4PGfD is similar to DDPGfD but is augmented with distributional critics.