d3rlpy: An Offline Deep Reinforcement Learning Library

Takuma Seno[1,2], **Michita Imai**[1]
Keio University[1], Sony AI[2]
{seno,michita}@ailab.ics.keio.ac.jp

Abstract

In this paper, we introduce d3rlpy, an open-sourced offline deep reinforcement learning (RL) library for Python. d3rlpy supports a number of offline deep RL algorithms as well as online algorithms via a user-friendly API. To assist deep RL research and development projects, d3rlpy provides practical and unique features such as data collection, exporting policies for deployment, preprocessing and postprocessing, distributional Q-functions, multi-step learning and a convenient command-line interface. Furthermore, d3rlpy additionally provides a novel graphical interface that enables users to train offline RL algorithms without coding programs. Lastly, the implemented algorithms are benchmarked with D4RL datasets to ensure the implementation quality. The d3rlpy source code can be found on GitHub: `https://github.com/takuseno/d3rlpy`.

1 Introduction

Deep reinforcement learning (RL) has been showing significant advancements in numerous domains such as gaming [28, 10], robotics [24] and autonomous driving [20]. While RL algorithms has a potential to solve complex tasks, active data collection is a major challenge especially for the environments where the interaction is expensive. To this problem, offline RL [25], where the algorithms find the good policy within the previously collected static dataset, is recently getting more attentions.

Although the recent deep RL papers open-sourced their experiment codes, the implementations are scattered across different repositories and the repositories are not usually providing user-friendly APIs, which makes it difficult for practitioners and researchers to incoporate the algorithms into their projects. Moreover, the non-standardized implementations makes it difficult for researchers to track the exact implementation difference between algorithms. On the other hand, there are already many libraries that provide a collection of deep RL algorithms [15, 4, 31]. However, they are designed for online RL paradigms and not providing complete supports for offline RL in terms of algorithms and interfaces.

In this paper, we introduce d3rlpy, an offline deep RL library for Python. d3rlpy provides many of online and offline RL algorithms built with PyTorch [32]. The key features of the d3rlpy are as follows:

Offline RL support d3rlpy is designed to support offline RL paradigms as well as the standard online RL. The available algorithms are listed in Table 1. All algorithms support the offline training interface and the online training interface so that users can easily pretrain offline algorithms and fine-tune the policy in online.

User-friendly API To make the state-of-the-art algorithms available for wide variety of users, d3rlpy provides a user-friendly API through the scikit-learn [33] style interface. For deep

Offline Reinforcement Learning Workshop at Neural Information Processing Systems, 2021.

algorithm	discrete action	continuous action	proposed as offline RL
DQN[28]	✓		
Double DQN[35]	✓		
DDPG[26]		✓	
TD3[14]		✓	
SAC[17, 5]	✓	✓	
BCQ[13, 11]	✓	✓	✓
BEAR[22]		✓	✓
CQL[23]	✓	✓	✓
AWAC[30]		✓	✓
CRR[36]		✓	✓
PLAS[37]		✓	✓
PLAS+P[37]		✓	✓
TD3+BC[12]		✓	✓

Table 1: A list of supported deep RL algorithms.

RL researchers, the interface is flexible enough to customize neural network models, change optimizers and train with their own datasets and environments.

Practical features Unlike the most of the existing libraries that focus on reproducing the paper results, d3rlpy also provides practical and unique features that allow users to integrate the powerful trained policies into their research or development projects.

Futhermore, d3rlpy also provides MINERVA, a novel tool with the graphical user interface (GUI) for an out-of-the-box offline RL training. MINERVA enables users to upload datasets, train offline RL algoirthms and export the trained policies for deployment without coding.

This paper is organized as follows: Section 2 presents related works on deep RL libraries, Section 3 presents the library design of the d3rlpy, Section 4 presents the practical and unique features of the d3rlpy, Section 5 introduces the MINERVA, a GUI offline RL training tool, Section 6 presents the benchmark scores in offline RL and Section 7 provides a conclusion.

2 Related work

A number of libraries are proposed in the deep RL field. ChainerRL [15] provides many of deep RL algorithms with the faithful reproduction results, which enables deep RL researchers to experiment their new algorithms and compare with baselines. Dopamine [4] is focusing on DQN variants [28, 6, 19] to make them available for researchers as baseline implementations. Tonic [31] aims to provide many of continuous control algorithms with large-scale benchmark results. There are two design differences between d3rlpy and the existing libraries. First, the d3rlpy is the first library that fully supports offline RL algorithms. Second, d3rlpy provides the simple and easy interface for practitioners such as application engineers as well as researchers while the existing libraries are focusing on research purposes where users need to understand how to compose them.

Regarding the GUI tool in RL, ChainerRL Visualizer [15] is proposed as a debugging tool for the ChainerRL. With ChainerRL Visualizer, the users can introspect RL policies by 1-step environment execution and plotting details of the model outputs. Vizarel [8] provides an interface for visualizing evaluation episodes and data distribution of the replay buffer [27]. Although the existing GUI tools were designed for debugging RL models, there were no existing tools capable of RL training. The MINERVA is the first tool that offers the RL training capability by leveraging the d3rlpy as a backend.

3 Design of d3rlpy

3.1 Library interface

d3rlpy provides scikit-learn's style API to make the use of this library as easy as possible. In terms of the library design, there are two main differences from the existing libraries. First, d3rlpy has an interface for offline RL training, which takes a dedicated RL dataset structure, `MDPDataset`,

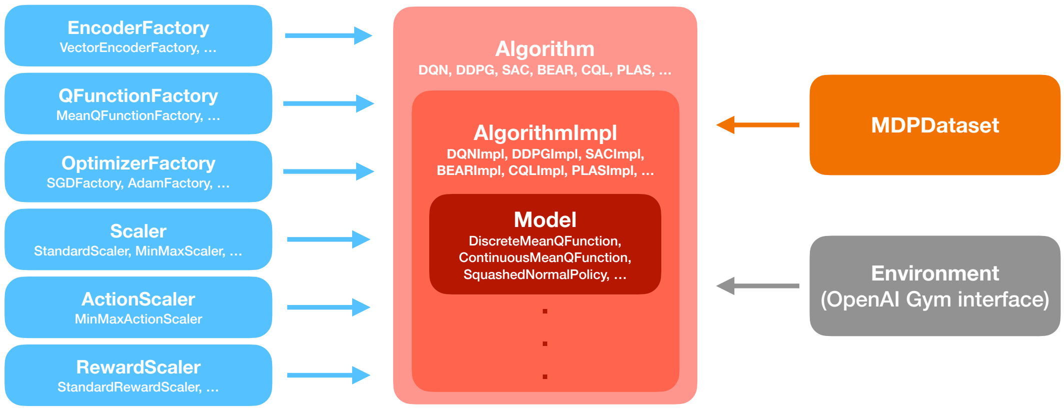


Figure 1: The illustration of module components in d3rlpy. `Algorithm` takes `EncoderFactory`, `QFunctionFactory`, `OptimizerFactory`, `Scaler`, `ActionScaler` and `RewardScaler` to instantiate `AlgorithmImpl` and `Model` modules. `MDPDataset` and OpenAI Gym-styled environment can be used to train policies.

described in Section 3.3. Second, the all methods for training such as `fit` and `fit_online` are implemented in the algorithm modules to make d3rlpy as user-friendly as possible by reducing the number of modules to use in training. Moreover, the neural network architectures are automatically selected from MLP and the convolutional model [28] depending on the observation, which allows the users to start training without composing the neural network models unless using the customized architectures. These design choices are expected to lower the bar to start using this library.

Since d3rlpy supports both offline and online training, the seamless transition from offline training to online fine-tuning is easily realized. As fine-tuning the policies trained in offline is demanded, but is still a challenging problem [30], this seamless transition supports the further resarches by allowing RL researchers to easily conduct fine-tuning experiments.

```python
import d3rlpy
import gym

# prepare MDPDataset object
dataset, env = d3rlpy.datasets.get_dataset("hopper-medium-v0")

# prpare algorithm object
sac = d3rlpy.algos.SAC(actor_learning_rate=3e-4, use_gpu=0, ...)

# start offline training
sac.fit(
    dataset,
    n_steps=500000,
    eval_episodes=dataset.episodes,
    scorers={
        "environment": d3rlpy.metrics.evaluate_on_environment(env),
        "average_value": d3rlpy.metrics.average_value_estimation_scorer,
    }
)

# seamless transition to online training
sac.fit_online(env, n_steps=500000, eval_env=gym.make("Hopper-v2"))
```

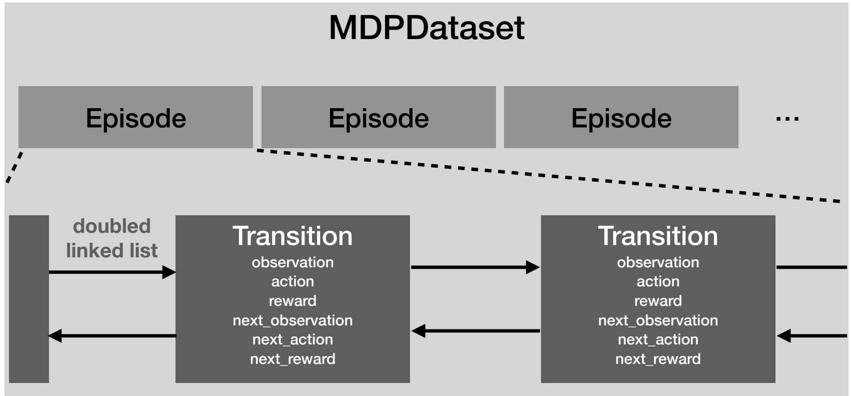


Figure 2: The diagram of the `MDPDataset`. The `MDPDataset` consists of a list of `Episode` object that includes `Transition` objects representing transition tuples. The `Transition` object has pointers to the previous and the next transitions.

3.2 Algorithms

Figure 1 depicts a module components in d3rlpy. The `Algorithm` has a hierarchical design that instantiates `AlgorithmImpl` inside itself. This hierarchy is to give high-level user-friendly APIs such as `fit` method to `Algorithm` and low-level APIs such as `update_actor` and `update_critic` to `AlgorithmImpl`. The main motivation of this hierarchical API system is to increase module resusability of the algoirthms when the new algorithm only requires high-level changes. For example, *delayed policy update* of TD3, which updates policy parameters every two gradient steps, can be implemented by adjusting the frequency of `update_actor` method calls in the high-level module without changing the low-level logics.

The users can customize neural network models through `EncoderFactory` module and change optimizers through `OptimizerFactory` module. `Scaler`, `ActionScaler` and `RewardScaler` are modules for preprocessing and postprocessing, which is described in Section 4.2. `QFunctionFactory` provides the distributional Q-function options, which is described in Section 4.4. `MDPDataset` is a module to represent RL datasets described in Section 3.3.

3.3 MDPDataset

d3rlpy provides `MDPDataset` module internally used to represent an offline RL dataset. Figure 2 depicts a diagram of the `MDPDataset` module. The hierarchy of the `Episode` module and `Transition` module is convenient for episode-wise data division for training and testing. The `Transition` module is implemented with Cython, which allows Python programs to use C++ extentions, to optimize memory copy at mini-batch sampling. By exploiting the pointers to the previous and the next transitions and the low-overhead implementation in Cython, the multi-step return calculation [34] and *frame stacking* [28] are done at batch sampling without slowing the training. Especially, the advantage of the sampling time *frame stacking* is to reduce CPU memory usage by placing single frame images on the memory instead of stacked images.

4 Practical features

4.1 Dataset collection

In offline RL research projects, data collection to create a new dataset plays an important role especially when users need to develop policies for new tasks. d3rlpy supports this data collection by giving OpenAI Gym [3] style environment and the replay buffer module to the algorithm object. For the diverse sets of dataeset creation, the data collection can be done with and without parameter updates, which corresponds to the dataset with a static policy and a non-stationary policy respectively.

```python
env = gym.make("Hopper-v2")
sac = d3rlpy.algos.SAC()
buffer = d3rlpy.online.buffers.ReplayBuffer(1000000, env=env)

# collect data without training
sac.collect(env, buffer, n_steps=1000000)
# collect data with training
# sac.fit_online(env, buffer, n_steps=1000000)

# convert replay buffer to MDPDataset
dataset = buffer.to_mdp_dataset()
```

4.2 Preprocessing and postprocessing

By exploiting the static dataset in offline RL training, d3rlpy provides various preprocessing and postprocessing methods in a handy way. For observation preprocessing, *normalization*, *standardization* and *pixel* (division by 255) are available. Especially, the observation standardization has been shown to improve the policy performance in offline RL setting [12]. Regarding the action preprocessing, *normalization* is available and the action output from the trained policy will be denormalized to the original scale as postprocessing. Lastly, reward preprocessing supports *normalization*, *standardization*, *clip* and *constant multiplication*. The reward preprocessing is still underexplored in RL researches, however, the reward scale is known as an important factor affecting the actor-critic policy performance [18].

```python
cql = d3rlpy.algos.CQL(
    scaler="standard",         # standardize observations
    action_scaler="min_max",   # normalize actions and postprocess outputs
    reward_scaler="standard",  # standardize rewards
)
```

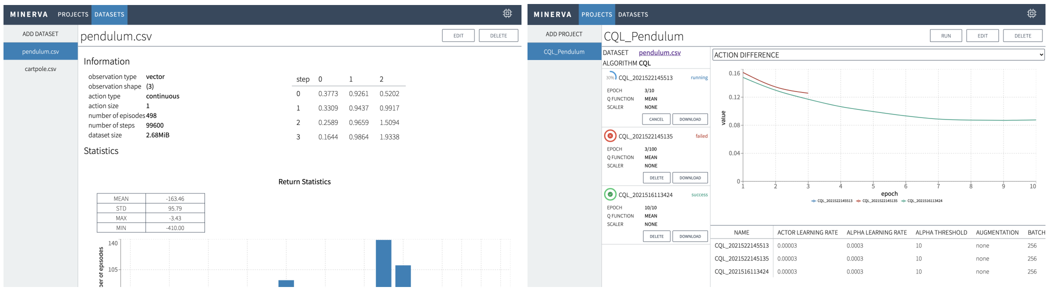
4.3 Export policy for deployment

Once the policy is ready for deployment, d3rlpy provides the policy export functionality. The main advantage of this feature is that users can deploy the trained policy without the d3rlpy dependency, and the exported formats support different programming languages other than Python and are optimized for inference. d3rlpy supports two machine learning model formats, TorchScript [32] and ONNX [2]. The TorchScript is a serialized model format provided by PyTorch, which is optimized and executable in C++ and Python programs only with the PyTorch dependency. The ONNX is an open format built to represent machine learning models with various language supports such as Python, C++ and JavaScript. To reduce the users' implementation efforts, this exported policy includes the tensor operations for preprocessing and postprocessing described above so that the programs can use the exported policy without processing inputs and outputs.

```python
cql = d3rlpy.algos.CQL()
cql.fit(...)
cql.save_policy("policy.pt")    # export as TorchScript
cql.save_policy("policy.onnx")  # export as ONNX
```

4.4 Distributional Q-function

d3rlpy supports distributional Q-functions, Quantile Regression (QR) [7] and Implicit Quantile Network (IQN) [6]. The distributional Q-functions have achieved dramatical performance improvements by capturing the variance of returns instead of learning the expected Q-values. Unlike conventional RL libraries that implement distributional Q-functions as DQN-variants, d3rlpy enables users to use them with all algorithms.



(a) Dataset Management (b) Training Management

Figure 3: The MINERVA. With the use of the MINERVA, the users can easily upload their datasets, train offline RL algorithms and export the trained policies as TorchScript and ONNX format for deployment.

```
# use Quantile Regression Q-function in CQL training
cql = d3rlpy.algos.CQL(q_func_factory="qr")
```

4.5 Multi-step learning

Multi-step learning is known as a simple and powerful approach to improve RL agent performance [34, 19]. In d3rlpy, the multi-step returns are computed when sampling a mini-batch and cut at the end of episodes if the episode length is shorter than the return horizon, which allows the algorithm to use the all transition tuples to train. The computation is implemented in Cython for the minimal overhead.

```
# use N=3 in multi-step learning for Q-function update
sac = d3rlpy.algos.SAC(n_steps=3)
```

4.6 Command-line interface

A convenient command-line interface (CLI) are also available in d3rlpy. This CLI is capable of visualizing the logged metrics data, executing the trained policies to record videos and exporting the policies for deployment described in Section 4.3. When executing or exporting the trained policies, the serialized algorithm metadata saved at the beginning of the training is used to construct the trained models. The metadata format is described in Appendix A.

```
# plot training metrics by matplotlib
$ d3rlpy plot d3rlpy_logs/.../environment.csv
# record videos of evaluation episodes
$ d3rlpy record d3rlpy_logs/.../model_500000.pt --env-id Hopper-v2
# export trained policy as TorchScript
$ d3rlpy export d3rlpy_logs/.../model_500000.pt --format torchscript
```

5 MINERVA: Graphical user interface for offline RL

The MINERVA is a GUI tool powered by the d3rlpy, which enables users to upload datasets, train offline RL algorithms and export the trained policy as TorchScript and ONNX formats. The main difference from the conventional visualizers for RL is that the MINERVA is capable of not only visualizing data, but also training policies by focusing on the offline training paradigm. The MINERVA is built as a Web interface with the d3rlpy as a backend with an ability to manage multiple training jobs in parallel. The dataset format is defined as CSV text file (additionally with image files

Dataset	SAC	TD3	AWAC	BCQ	BEAR	CQL	CRR	PLAS	PLAS+P	TD3+BC
halfcheetah-random-v0	30.8	31.9	18.4	2.2	2.3	29.7	22.0	27.0	28.5	11.0
walker2d-random-v0	2.9	3.6	0.4	4.1	4.9	5.3	3.6	6.7	7.6	2.7
hopper-random-v0	0.8	1.1	10.9	10.6	10.1	10.9	10.6	10.6	11.0	11.0
halfcheetah-medium-v0	25.1	31.0	40.5	40.2	36.9	41.7	41.8	40.2	41.8	42.6
walker2d-medium-v0	7.1	0.1	64.7	45.8	56.8	75.2	53.4	33.4	66.2	74.9
hopper-medium-v0	0.8	0.8	40.4	40.2	33.5	46.2	43.4	68.7	58.4	86.7
halfcheetah-medium-replay-v0	42.0	36.1	42.0	39.1	37.6	40.1	41.3	44.1	45.1	43.0
walker2d-medium-replay-v0	8.1	8.1	24.2	16.0	12.9	18.8	26.7	22.2	3.4	25.3
hopper-medium-replay-v0	0.6	10.5	28.8	16.9	26.2	30.6	37.3	16.2	11.0	31.0
halfcheetah-medium-expert-v0	-0.4	-1.7	18.4	58.1	45.3	9.0	13.1	82.8	66.2	86.9
walker2d-medium-expert-v0	0.0	0.9	55.6	52.3	62.4	69.7	60.0	98.1	84.3	96.0
hopper-medium-expert-v0	11.7	8.7	51.2	112.5	83.3	100.9	19.7	110.4	99.7	112.0

Table 2: Normalized scores collected with the final trained policy in D4RL. The scores are averaged over 3 random seeds.

Dataset	SAC	TD3	AWAC	BCQ	BEAR	CQL	CRR	PLAS	PLAS+P	TD3+BC
halfcheetah-random-v0	33.9	33.9	23.0	2.3	3.3	34.8	23.6	33.0	34.9	12.5
walker2d-random-v0	23.7	18.7	7.9	6.5	11.9	15.5	10.8	21.4	22.7	21.3
hopper-random-v0	14.9	4.4	11.7	11.0	11.1	11.2	15.0	11.0	11.1	11.2
halfcheetah-medium-v0	31.2	33.5	42.3	41.0	37.8	42.6	42.4	40.7	42.4	43.4
walker2d-medium-v0	16.5	15.1	80.1	76.1	76.3	82.4	72.7	67.6	80.8	82.5
hopper-medium-v0	3.5	13.7	102.5	100.5	97.4	100.6	95.0	98.5	101.3	100.8
halfcheetah-medium-replay-v0	47.0	42.0	43.9	42.3	39.7	46.3	42.9	44.6	46.1	44.6
walker2d-medium-replay-v0	31.2	27.3	50.9	34.9	34.5	49.5	44.6	45.3	44.9	53.4
hopper-medium-replay-v0	30.9	50.1	98.5	42.1	58.8	65.2	86.3	53.4	51.9	64.5
halfcheetah-medium-expert-v0	5.7	4.0	27.4	86.5	63.7	24.4	23.6	102.7	93.5	99.9
walker2d-medium-expert-v0	13.9	17.0	112.0	88.9	91.1	110.9	98.1	108.1	111.4	111.2
hopper-medium-expert-v0	28.9	49.5	112.8	114.5	112.8	114.7	87.5	113.6	114.7	114.0

Table 3: Normalized scores collected with the best trained policy in D4RL. The scores are averaged over 3 random seeds.

for image observations) described in Appendix B. Whether the action-space is discrete or continuous is automatically determined based on the dataset and the algorithm options shown to the users depend on the detected action-space.

Figure 3 shows example screenshots of a dataset management page and an experiment management page. In the dataset management page depicted in Figure 3(a), the dataset metadata such as dataset size and observation shapes, contents and data statistics are displayed. The training management page depicted in Figure 3(b) shows a list of training jobs and collected metrics to compare multiple trials.

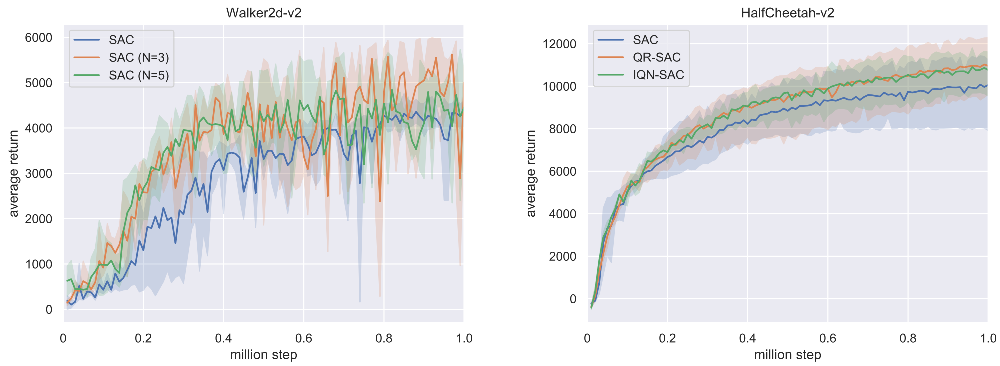
6 Benchmark

6.1 Offline RL

We evaluate the implemented algorithms on the D4RL benchmark of OpenAI Gym MuJoCo tasks [3, 9]. We train each algorithm for 500K gradient steps and evaluate every 5000 steps to collect evaluation performance in environments for 10 episodes. The details of hyperparameters are reported in Appendix C.

The final performance results are reported in Table 2. The scores are normalized to the range between 0 and 100 [9]. TD3+BC performs well across all datasets despite its simple implementation. Interestingly, PLAS and PLAS+P performed closely to TD3+BC although those were not often compared in the previous researches.

The existing works for offline RL algorithms usually report only final performance results. However, each algorithm has different trends of overfitting datasets, which makes it difficult to evaluate the potential performance in comparison for the same gradient steps. In this paper, we also report the best performance results in Table 3. This performance metrics will be more important once we establish a method to detect policy overfitting. Surprisingly, Table 3 shows that the best performance of AWAC and CRR, the algorithms with supervised regression policy training, is significantly better than the



(a) Multi-step Learning (b) Distributional Q-function

Figure 4: Training curves on continuous control tasks with d3rlpy's training features enabled. (a) Multi-step learning performance in Walker2d-v2 environment. x-axis represents environment steps and y-axis represents average returns in evaluation episodes. The scores are averaged over 3 random seeds. (b) Distributional Q-function performance in HalfCheetah-v2 environment.

final performance. This suggests that the supervised regression policy training algorithms potentially perform well, but overfitting is more critical in those algorithms.

The raw scores and side-by-side comparisons against reference scores are reported in Appendix D.

6.2 Online RL with multi-step learning and distributional Q-functions

We evaluate d3rlpy's training features with MuJoCo tasks in online training scenarios. In this experiment, we evaluate multi-step learning described in Section 4.5 and distributional Q-functions described in Section 4.4. The multi-step learning and distributional Q-functions are used with the standard SAC algorithm without changing hyperparameter. We train each algorithm for 1M environment steps and evaluate every 10K steps to collect evaluation performance in environments for 10 episodes.

Figure 4 shows the result of continuous control tasks. In particular environments, simple modifications by enabling multi-step learning or distributional Q-functions improve the performance from the standard algorithm. This suggests that it is possible that users achieve better performance than SOTA scores reported in their papers only with d3rlpy. The rest of results are reported in Appendix E and Appendix F.

7 Conclusion

In this paper, we introduced an offline deep reinforcement learning library, d3rlpy, and its GUI interface, MINERVA. d3rlpy provides user-friendly APIs and practical features for the research and development projects. Also, the implemented algorithms are benchmarked with D4RL datasets, which makes d3rlpy comparable to the existing and future works.

In the future work, we will continuously add new RL algorithms and features based on demands from the open-source community and RL research trends. Specifically, we will support off-policy evaluation methods [29] since evaluating the trained policies only with static datasets is important for the practical applications. We will also benchmark discrete-control algorithms such as CQL and BCQ with Atari 2600 datasets [1, 16] since we only benchmarked continuous-control algorithms in this paper.

Hopefully, d3rlpy will contribute to the standardization of offline RL implementations and push offline RL algorithms to the more diverse applications. And, we believe that d3rlpy solves reproducibility issues [18] by providing the exact scripts used for benchmarking.

Acknowledgement

This work is supported by Information-technology Promotion Agency, Japan (IPA), Exploratory IT Human Resources Project (MITOU Program) in the fiscal year 2020. We would like to express our genuine gratitude for the contributions made by the voluntary contributors. We would also like to thank our users who have provided constructive feedbacks and insights. Lastly, we would like to thank Yu Ishihara and Shunichi Sekiguchi from Sony R&D Center Tokyo for their fruitful comments about the paper.

References

[1] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

[2] Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. `https://github.com/onnx/onnx`, 2019.

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

[4] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018.

[5] Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

[6] Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pages 1096–1105. PMLR, 2018.

[7] Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[8] Shuby Deshpande, Benjamin Eysenbach, and Jeff Schneider. Interactive visualization for debugging rl. *arXiv preprint arXiv:2008.07331*, 2020.

[9] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[10] Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Dürr. Superhuman performance in gran turismo sport using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 6(3):4257–4264, 2021.

[11] Scott Fujimoto, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*, 2019.

[12] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*, 2021.

[13] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

[14] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1587–1596, 2018.

[15] Yasuhiro Fujita, Prabhat Nagarajan, Toshiki Kataoka, and Takahiro Ishikawa. Chainerrl: A deep reinforcement learning library. *Journal of Machine Learning Research*, 22(77):1–14, 2021.

[16] Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gomez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel Mankowitz, Cosmin Paduraru, et al. Rl unplugged: Benchmarks for offline reinforcement learning. *arXiv e-prints*, pages arXiv–2006, 2020.

[17] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[18] P Henderson, R Islam, P Bachman, J Pineau, D Precup, and D Meger. Deep reinforcement learning that matters. arxiv 2017. *arXiv preprint arXiv:1709.06560*, 2017.

[19] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[20] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE, 2019.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *arXiv preprint arXiv:1906.00949*, 2019.

[23] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[24] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47), 2020.

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

[26] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representation*, 2016.

[27] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[29] Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. *arXiv preprint arXiv:1906.04733*, 2019.

[30] Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[31] Fabio Pardo. Tonic: A deep reinforcement learning library for fast prototyping and benchmarking. *arXiv preprint arXiv:2011.07537*, 2020.

[32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[33] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[34] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT Press Cambridge, 1998.

[35] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. corr abs/1509.06461 (2015). *arXiv preprint arXiv:1509.06461*, 2015.

[36] Ziyu Wang, Alexander Novikov, Konrad Żołna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. *arXiv preprint arXiv:2006.15134*, 2020.

[37] Wenxuan Zhou, Sujay Bajracharya, and David Held. Latent action space for offline reinforcement learning. In *Conference on Robot Learning*, 2020.

Appendix

A Serialized metadata

The metadata is serialized in JSON text format. The example of serialized metadata used by the CLI is shown as follows:


```
{
  "algorithm": "CQL",
  "observation_shape": [
    17
  ],
  "action_size": 6
  "action_scaler": null,
  "actor_encoder_factory": {
    "type": "vector",
    "params": {
      "hidden_units": [
        256,
        256,
        256
      ],
      "activation": "relu",
      "use_batch_norm": false,
      "dropout_rate": null,
      "use_dense": false
    }
  },
  "actor_learning_rate": 0.0001,
  "actor_optim_factory": {
    "optim_cls": "Adam",
    "betas": [
      0.9,
      0.999
    ],
    "eps": 1e-08,
    "weight_decay": 0,
    "amsgrad": false
  },
  .
  .
  .
}
```


Using this serialized data, the algorithm object can be constructed in the program, which is especially convenient when continuing the training from the trained models:

```
# construct CQL algorithm object from the serialized data
cql = d3rlpy.algos.CQL.from_json("d3rlpy_logs/.../params.json")
# load the trained models
cql.load_model("d3rlpy_logs/.../model_500000.pt")
# resume training
cql.fit(...)
```

B Dataset format for MINERVA

The example dataset is shown below. In this case, the observation is 2-dimensional vector and the action is 3-dimensional continuous vector.

```
episode,observation:0,observation:1,action:0,action:1,action:2,reward
0      ,0.030        ,1.120        ,0.2     ,0.1     ,1.3     ,1.0
0      ,0.032        ,0.241        ,1.4     ,1.4     .0.3     ,0.0
.
.
1      ,0.030        ,1.120        ,0.1     ,1.5     .0.4     ,0.0
1      ,0.032        ,0.312        ,1.2     ,0.3     .1.0     ,1.0
.
.
```

The example of image observation dataset is shown below. In this case, the observation is pixel and the action is discrete. When uploading the image observation datasets, zipped image files need to be additionally uploaded along with the CSV file.

```
episode,observation:0,action:0,reward
0      ,image1.png   ,1       ,0.0
0      ,image2.png   ,0       ,1.0
.
.
1      ,image13.png  ,1       ,0.0
1      ,image14.png  ,0       ,1.0
.
.
```

C Experimental Details

Table 4 shows hyperparameters used in benchmarking. We used discount factor of 0.99, target update rate of 5e-3 and an Adam optimizer [21] across all algorithms. The default architecture was MLP with hidden layers of [256, 256] unless we explicitly explain it.

Table 4: Hyperparameters used in benchmarking.

Algorithm	Hyperparameter	Value
SAC	Critic learning rate	3e-4
	Actor learning rate	3e-4
	Mini-batch size	256
TD3	Critic learning rate	3e-4
	Actor learning rate	3e-4
	Mini-batch size	256
	Policy noise	0.2
	Policy noise clipping	(-0.5, 0.5)
	Policy update frequency	2
AWAC	Critic learning rate	3e-4
	Actor learning rate	3e-4
	Mini-batch size	1024
	λ [30]	1
	Actor hidden units	[256, 256, 256, 256]
	Actor weight decay	1e-4
BCQ	Critic learning rate	1e-3
	Actor learning rate	1e-3
	VAE learning rate	1e-3

	Mini-batch size	100
	λ [13]	0.75
	Critic hidden units	[400, 300]
	Actor hidden units	[400, 300]
	VAE encoder hidden units	[750, 750]
	VAE decoder hidden units	[750, 750]
	VAE latent size	$2 \times \|A\|$
	Perturbation range	0.05
	Action samples	100
BEAR	Critic learning rate	3e-4
	Actor learning rate	1e-4
	VAE learning rate	3e-4
	α learning rate	1e-3
	Mini-batch size	256
	VAE encoder hidden units	[750, 750]
	VAE decoder hidden units	[750, 750]
	VAE latent size	$2 \times \|A\|$
	MMD σ	20
	MMD kernel	laplacian (gaussian for HalfCheetah datasets)
	MMD action samples	4
	α threshold	0.05
	Target action samples	10
	Evaluation action samples	100
	Pretraining steps	40000
CQL	Critic learning rate	3e-4
	Actor learning rate	1e-4
	Fixed α	5 (10 for medium datasets)
	Mini-batch size	256
	Critic hidden units	[256, 256, 256]
	Actor hidden units	[256, 256, 256]
	Action samples	10
CRR	Critic learning rate	3e-4
	Actor learning rate	3e-4
	Mini-batch size	256
	Action samples	4
	Advantage type	mean
	Weight type	binary
PLAS	Critic learning rate	1e-3
	Actor learning rate	1e-4
	VAE learning rate	1e-4
	Mini-batch size	100
	Critic hidden units	[400, 300]
	Actor hidden units	[400, 300]
	VAE encoder hidden units	[750, 750] ([128, 128] for medium-replay datasets)
	VAE decoder hidden units	[750, 750] ([128, 128] for medium-replay datasets)
	λ [13]	1.0
	VAE latent size	$2 \times \|A\|$
	VAE pretraining steps	500000
PLAS+P	Critic learning rate	1e-3
	Actor learning rate	1e-4
	VAE learning rate	1e-4
	Mini-batch size	100
	Critic hidden units	[400, 300]
	Actor hidden units	[400, 300]
	VAE encoder hidden units	[750, 750] ([128, 128] for medium-replay datasets)
	VAE decoder hidden units	[750, 750] ([128, 128] for medium-replay datasets)
	λ [13]	1.0
	VAE latent size	$2 \times \|A\|$
	VAE pretraining steps	500000

	Perturbation range	0.05
TD3+BC	Critic learning rate	3e-4
	Actor learning rate	3e-4
	Mini-batch size	256
	Policy noise	0.2
	Policy noise clipping	(-0.5, 0.5)
	Policy update frequency	2
	α	2.5
	Observation preprocess	standardization

D Additional Benchmark Results

Dataset	SAC	TD3	AWAC	BCQ	BEAR	CQL	CRR	PLAS	PLAS+P	TD3+BC
halfcheetah-random-v0	3546.6	3674.7	2005.6	-1.1	-0.3	3401.6	2452.4	3076.1	3260.1	1092.7
walker2d-random-v0	136.6	166.2	20.0	189.4	228.3	244.0	165.6	316.9	348.4	126.0
hopper-random-v0	5.8	16.0	335.7	326.3	308.7	334.3	323.8	323.4	336.6	339.2
halfcheetah-medium-v0	2831.6	3568.4	4749.3	4715.7	4304.5	4892.3	4911.4	4707.4	4906.0	5014.3
walker2d-medium-v0	325.9	5.3	2972.2	2103.1	2607.3	3451.7	2451.8	1532.9	3039.7	3442.1
hopper-medium-v0	6.6	4.2	1294.2	1286.8	1071.2	1483.7	1391.2	2220.9	1881.8	2800.0
halfcheetah-medium-replay-v0	4928.7	4197.9	4922.4	4577.5	4385.2	4703.6	4852.8	5198.6	5322.2	5056.9
walker2d-medium-replay-v0	375.1	375.5	1112.2	734.0	594.7	864.8	1229.6	1021.7	159.4	1164.4
hopper-medium-replay-v0	0.1	321.0	918.6	528.8	831.9	974.4	1193.3	506.3	336.4	989.9
halfcheetah-medium-expert-v0	-331.0	-490.3	2008.8	6927.8	5338.8	842.1	1342.7	9995.3	7947.5	10510.4
walker2d-medium-expert-v0	2.7	43.8	2556.3	2401.6	2886.3	3209.4	2341.2	4505.3	3872.6	4410.2
hopper-medium-expert-v0	360.9	262.0	1647.2	3641.6	2692.2	3262.0	621.0	3571.6	3223.3	3624.4

Table 5: The raw scores collected with the final trained policy in D4RL. The scores are averaged over 3 random seeds.

Dataset	SAC	TD3	AWAC	BCQ	BEAR	CQL	CRR	PLAS	PLAS+P	TD3+BC
halfcheetah-random-v0	3932.2	3931.2	2570.4	6.2	129.7	4037.0	2644.2	3816.7	4047.6	1271.6
walker2d-random-v0	1088.3	859.3	365.0	298.6	548.1	712.5	496.7	984.4	1042.2	980.3
hopper-random-v0	466.2	124.3	360.3	336.9	341.1	344.2	467.5	336.6	342.4	344.0
halfcheetah-medium-v0	3593.5	3880.9	4966.1	4816.2	4408.9	5005.2	4979.9	4777.2	4979.5	5105.1
walker2d-medium-v0	759.9	694.7	3680.0	3495.2	3505.8	3784.4	3346.5	3103.2	3710.9	3787.4
hopper-medium-v0	94.8	424.5	3316.3	3251.2	3148.2	3252.9	3072.2	3186.0	3276.5	3260.2
halfcheetah-medium-replay-v0	5560.7	4933.1	5174.3	4966.3	4652.9	5467.8	5049.9	5256.2	5438.9	5259.0
walker2d-medium-replay-v0	1431.6	1253.4	2336.3	1601.8	1586.2	2272.7	2050.0	2080.6	2062.2	2453.1
hopper-medium-replay-v0	985.7	1609.3	3184.9	1350.4	1893.9	2100.4	2787.2	1718.3	1669.2	2080.5
halfcheetah-medium-expert-v0	428.7	214.1	3122.6	10459.6	7630.9	2755.0	2648.1	12469.9	11327.3	12124.2
walker2d-medium-expert-v0	640.7	783.5	5144.8	4081.5	4184.1	5091.1	4506.3	4964.3	5104.1	5108.4
hopper-medium-expert-v0	920.4	1590.4	3650.5	3704.9	3651.5	3713.5	2828.5	3677.3	3712.5	3690.4

Table 6: The raw scores collected with the best trained policy in D4RL. The scores are averaged over 3 random seeds.

Dataset	**AWAC**		**BCQ**		**BEAR**		**CQL**		**PLAS**		**PLAS+P**		**TD3+BC**	
	d3rlpy	Ref [30]	d3rlpy	Ref [9]	d3rlpy	Ref [9]	d3rlpy	Ref†	d3rlpy	Ref [37]	d3rlpy	Ref [37]‡	d3rlpy	Ref [12]
halfcheetah-random-v0	18.4	2.2	2.2	2.2	2.3	25.1	29.7	28.5	27.0	25.8	28.5	28.3	11.0	10.2
walker2d-random-v0	0.4	5.1	4.1	4.9	4.9	7.3	5.3	1.2	6.7	3.1	7.6	6.8	2.7	1.4
hopper-random-v0	10.9	9.6	10.6	10.6	10.1	11.4	10.9	10.6	10.6	10.5	11.0	13.3	11.0	11.0
halfcheetah-medium-v0	40.5	37.4	40.2	40.7	36.9	41.7	41.7	38.8	40.2	39.3	41.8	42.2	42.6	42.8
walker2d-medium-v0	64.7	30.1	45.8	53.1	56.8	59.1	75.2	48.7	33.4	44.6	66.2	66.9	74.9	79.7
hopper-medium-v0	40.4	72.0	40.2	54.5	33.5	52.1	46.2	31.2	68.7	32.9	58.4	36.9	86.7	99.5
halfcheetah-medium-replay-v0	42.0	–	39.1	38.2	37.6	38.6	40.1	44.9	44.1	43.9	45.1	45.7	43.0	43.3
walker2d-medium-replay-v0	24.2	–	16.0	15.0	12.9	19.2	18.8	25.5	22.2	30.2	3.4	14.3	25.3	25.2
hopper-medium-replay-v0	28.8	–	16.9	33.1	26.2	33.7	30.6	30.1	16.2	27.9	11.0	51.9	31.0	31.4
halfcheetah-medium-expert-v0	18.4	36.8	58.1	64.7	45.3	53.4	9.0	11.3	82.8	96.6	66.2	99.3	86.9	97.9
walker2d-medium-expert-v0	55.6	42.7	52.3	57.5	62.4	40.1	69.7	75.5	98.1	89.6	84.3	94.7	96.0	101.1
hopper-medium-expert-v0	51.2	80.9	112.5	110.9	83.3	96.3	100.9	100.0	110.4	111.0	99.7	108.7	112.0	112.2

Table 7: The side-by-side comparison against the reference normalized scores collected with the final trained policy in D4RL. (†) The reference scores of CQL were collected with the author's implementation with the hyperparameters suggested in GitHub. (‡) The reference scores of PLAS+P were reported by picking the best result of sweeeping a parameter of the perturbation modelfor each dataset [37] while we fixed the parameter because the author did not provide the chosen values.

E Additional Multi-step Learning Results

We additionally evaluated the multi-step learning feature with tasks provided by Bullet [1], an open-sourced simulator as well as other MuJoCo tasks. Interestingly, we observed more performance improvements in Bullet tasks. This different trend indicates that Bullet tasks could be more difficult tasks than MuJoCo tasks so that the powerful methods can obtain significant performance improvements with Bullet tasks.

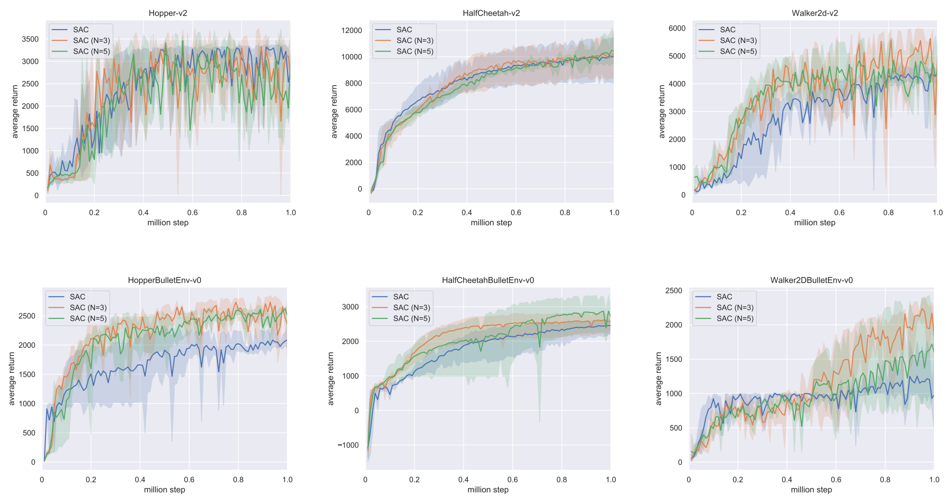


Figure 5: Training curves on three different MuJoCo and Bullet tasks for multi-step learning.

F Additional Distributional Q-function Results

We additionally evaluated the distributional Q-function feature with tasks provided by Bullet as well as other MuJoCo tasks. Just same as multi-step learning results, we also observed more performance improvements in Bullet tasks.

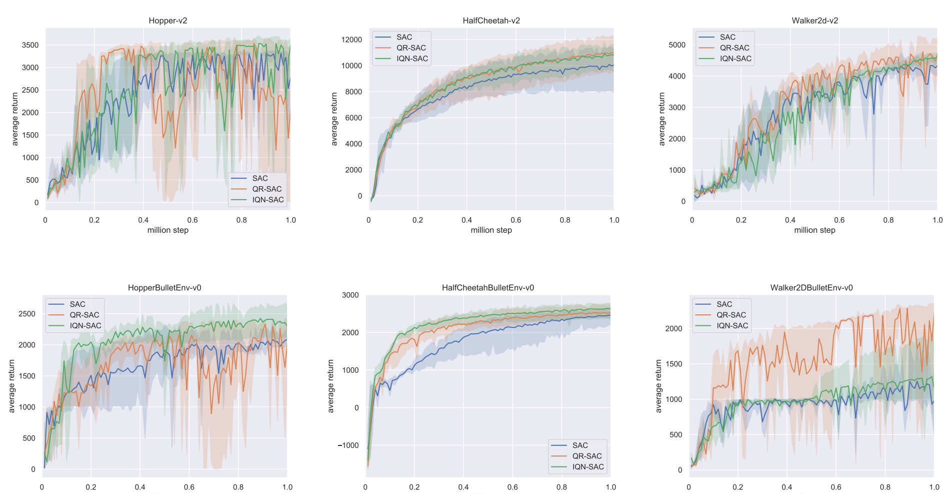


Figure 6: Training curves on three different MuJoCo and Bullet tasks for distributional Q-functions.

[1] https://github.com/bulletphysics/bullet3