Dataset Reduction for Offline Reinforcement Learning using Genetic Algorithms with Image-Based Heuristics

Enrique Mateos-Melero
Universidad Carlos III de Madrid
Leganés, Spain
enmateos@pa.uc3m.es

Miguel Iglesias Alcázar
Universidad Carlos III de Madrid
Leganés, Spain
migigles@pa.uc3m.es

Raquel Fuentetaja
Universidad Carlos III de Madrid
Leganés, Spain
rfuentet@inf.uc3m.es

Fernando Fernández
Universidad Carlos III de Madrid
Leganés, Spain
ffernand@inf.uc3m.es

Abstract

In offline Reinforcement Learning (RL), the size and quality of the training dataset play a crucial role in determining policy performance. Large datasets can lead to excessive training times, while low-quality data can result in sub-optimal policies, particularly for deep learning-based RL frameworks. To address these challenges, we propose a novel approach that leverages genetic algorithms for efficient dataset reduction, paired with image-based learning using Convolutional Neural Networks (CNNs) to reduce the evaluation time of the fitness function. Specifically, our method predicts the performance of policies (fitness) learned from offline RL datasets (phenotype) and identifies optimized subsets that preserve or enhance policy quality. We evaluate our approach across three well-established RL domains, demonstrating that it effectively reduces dataset size while maintaining or improving policy performance. Furthermore, we show the transferability of the learned models to similar tasks, enabling efficient dataset optimization via transfer learning.

CCS Concepts

• **Computing methodologies** → **Machine learning**; **Genetic algorithms**; *Markov decision processes*; Neural networks.

Keywords

Offline Reinforcement Learning, Genetic algorithms, Dataset Quality, Learning Performance Prediction, Convolutional Neural Networks

ACM Reference Format:
Enrique Mateos-Melero, Miguel Iglesias Alcázar, Raquel Fuentetaja, and Fernando Fernández. 2025. Dataset Reduction for Offline Reinforcement Learning using Genetic Algorithms with Image-Based Heuristics. In *Genetic and Evolutionary Computation Conference (GECCO '25), July 14–18, 2025, Malaga, Spain.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3712256.3726364

GECCO '25, July 14–18, 2025, Malaga, Spain

ACM ISBN 979-8-4007-1465-8/2025/07
https://doi.org/10.1145/3712256.3726364

1 Introduction

In classical Reinforcement Learning (RL) methods, such as tabular approaches [21], computational time was not a significant concern as learning costs were typically measured based on the resources required to gather training examples. However, recent research in RL, such as off-line RL [13] or batch approaches in on-line RL [10], which can be computationally intensive, has highlighted the need to reduce learning time.

A fundamental approach to achieving this reduction is decreasing the number of training instances, as computational time is typically at least linearly related to this number [8]. However, for a specific task with a training set of size M consisting of experience episodes and other RL parameters (e.g., representation model, algorithm), selecting a smaller subset of experiences that still allows for near-optimal policy learning is a complex problem. In this paper, our focus is on providing an accurate, cost-efficient, and transferable solution to this problem.

By "accurate", we mean that the performance of the learned policy with the reduced training set should be at least as good as with the entire dataset. By "cost-effective", we mean that the computational cost of determining this subset and learning with it should be lower than the cost of learning with the entire dataset. By "transferable", we mean that the solution learned to determine the performance for a specific task can be applied to similar tasks, increasing utility. Our approach addresses several questions sequentially: a) Can we find a method to reduce a training set of size M to a smaller one of size N, where $N \ll M$, without sacrificing performance? b) Can we do this efficiently? c) Can the models generated by this method be reused in similar tasks?

To achieve this, we propose framing the reduction of the training set as an optimization problem, where the objective is, given a set of episodes and a learning framework (RL algorithm, parameters, etc.), to select a subset of episodes that maximizes a specific metric. The goal is to minimize the training set while retaining episodes that enable accurate policy learning with the RL framework. Due to the vast number of possible episode subsets (the power set of the set of episodes), finding an optimal solution is infeasible. Instead, we employ a genetic algorithm, where each individual represents a different subset of the training set. The fitness function is determined by the accuracy of the policy learned from the corresponding subset and is inversely proportional to its size, favoring more accurate policies and smaller training sets.

To address the computational cost issue, we introduce a heuristic that reduces the overhead of computing the fitness function. Instead of conducting RL processes, we learn a predictive model for estimating the policy's performance derived from a training set. In offline RL, where dataset quality is crucial for learning an effective policy, we leverage Convolutional Neural Networks (CNNs) [12] to efficiently evaluate datasets and forecast their impact on policy learning. Our approach enables rapid estimation of dataset performance, which is critical for applications like transfer learning [23, 31], curriculum learning [14], Sim2Real [30], and batch RL [11].

Furthermore, we assess the transferability of predictive models by applying them to new scenarios with modified state space sizes or state transition functions. The results demonstrate promising capabilities for reusing models in similar tasks, opening up opportunities for applications in various RL contexts. Additionally, we explore the use of genetic algorithms applying our predictive model to enhance dataset quality, providing a method to create efficient datasets for RL training.

The structure of this paper is as follows: Section 2 provides background on RL, Section 3 formalizes the Episode Selection Problem as an optimization task and describes our genetic-based solution, Section 4 describes how we efficiently estimate the fitness function with Deep Learning and its image-based solution, and the results in three different environments. Section 5 reviews related work, and Section 6 offers conclusions and directions for future work.

2 Preliminaries

Reinforcement Learning (RL) addresses the problem of learning to control a dynamic system defined by a Markov Decision Process (MDP) [21]. An MDP can be defined as a tuple $\mathcal{M} = \langle S, A, R, \mathcal{P}, \gamma, \mathcal{I}, \beta \rangle$, where S is a set of states; A is a set of actions; R is a reward function $R : A \times S \rightarrow Pr[\mathbb{R}]$, $\mathcal{P} : S \times A \times S \rightarrow Pr[S]$ is a transition distribution function; $\gamma \in [0, 1)$ is a discount factor; $\mathcal{I} : Pr[S]$ is an initial state distribution; and $\beta : S \rightarrow \{0, 1\}$ is an episode termination function. At time step t, the state of the environment is s_t and the agent, using its behavior policy $\pi : S \rightarrow Pr[A]$, selects an action a_t that when executed in the environment, produces a reward $r_t \sim R(s, a)$ and leads to a new state $s_{t+1} \sim \mathcal{P}(s, a)$. The sequence (s_t, a_t, r_t, s_{t+1}) is called an *experience*. A sequence of experiences is an *episode*. When $\beta(s') = 1$ or a horizon T is reached the episode ends. The sequence of state-action transitions in an episode is a *trajectory*, $\tau = (s_0, a_0, s_1, a_1, \dots)$, which can be derived directly from the episodes.

Given a policy π, the *long-term discounted return* $\mathcal{J}(\pi)$, defined by Equation 1, is its expected cumulative discounted reward. An *optimal policy* π^* is a policy maximizing $\mathcal{J}(\pi)$: $\pi^* = \arg\max_\pi \mathcal{J}(\pi)$. The objective of an RL agent is to learn an optimal policy.

$$\mathcal{J}(\pi) = \mathbb{E}_{s_0 \sim \mathcal{I}, s_{t+1} \sim \mathcal{P}(s_t, \pi(s_t)), r_t \sim R(s_t, \pi(s_t))} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \quad (1)$$

In online RL, the agent learns through trial and error by interacting with the environment [21]. However, in offline RL, which is a fully data-driven method, the agent is not allowed to interact with the environment during learning [11, 13]. Instead, it receives a dataset $\mathcal{D}_{env}$ consisting on set of M episodes sampled from the environment, $\mathcal{D}_{env} = \{e_i\}_{i=1,\dots,M}$. Since the training data is given, and it is usually finite, the agent can not expect to come up with an optimal policy. Then, the objective of the agent is to derive the *best* possible policy from the dataset for the given environment [11].

3 Episode Selection Problem

In offline RL, the input dataset, denoted as $\mathcal{D}_{env}$, can often be overwhelmingly large, making the execution time of the learning algorithm impractical. Additionally, the dataset's source may be a mixture of "good" and "bad" policies. Therefore, a fundamental question arises: Can we significantly reduce the size of $\mathcal{D}_{env}$ while still enabling the learning algorithm to produce a high-quality policy? This question essentially frames the problem as a multi-objective optimization task. On one hand, we aim to minimize the size of the resulting dataset and, on the other hand, we seek to maximize the quality of the policy learned from it. However, it is possible to simplify this into a single-objective optimization problem by combining both objectives into a single metric. Let us define the Episode Selection Problem (ESP) metric, denoted as ψ, as follows:

Definition 1 (ESP Metric ψ). *Let $\mathcal{D}$ be a set of episodes, θ be a representation of the RL framework (algorithms, parameters, ...) and let $\hat{\mathcal{J}}(\mathcal{D}|\theta)$ represent an estimate of the performance of the best possible policy $\pi_{\mathcal{D}}$ that an RL agent can learn from $\mathcal{D}$ given θ, such that $\hat{\mathcal{J}}(\mathcal{D}|\theta) \approx \mathcal{J}(\pi_{\mathcal{D}}|\theta)$. Then, the ESP metric $\psi : \mathcal{D} \rightarrow \mathbb{R}$ is defined as:*

$$\psi(\mathcal{D}|\theta) = f(\hat{\mathcal{J}}(\mathcal{D}|\theta), |\mathcal{D}|)$$

where f is a weighted sum of $\hat{\mathcal{J}}(\mathcal{D}|\theta)$ and $\frac{1}{|\mathcal{D}|}$.

Note that we always maintain the dependency with the RL framework, θ. It is easy to understand that, if this framework changes (for instance, the generalization method), the gain obtained by learning with the same dataset may vary. For simplicity of the notation, we eliminate this dependency in the notation in the rest of the paper. Therefore, with this metric in place, we can formally define the Episode Selection Problem (ESP) as follows:

Definition 2 (Episode Selection Problem, ESP($\mathcal{D}, \psi$)). *Given a set of episodes $\mathcal{D}$ and a metric ψ, find a reduced subset $\mathcal{D}^* \subseteq \mathcal{D}$ that maximizes ψ:*

$$\mathcal{D}^* = \arg\max_{\mathcal{D}' \subseteq \mathcal{D}} \psi(\mathcal{D}')$$

An optimal ESP could theoretically be solved by a brute-force approach, by generating all possible subsets of the input dataset and evaluating them using the metric ψ under θ. However, given the potentially enormous size of $\mathcal{D}_{env}$ and the exponential number of subsets ($2^{|\mathcal{D}_{env}|}$), such a brute-force approach is not feasible. Instead, we propose a method to find a suboptimal solution using stochastic search. Specifically, we employ a genetic algorithm [7, 15], where each individual in the population represents a different subset of the training set, and the fitness function is defined in terms of the ESP metric, ψ.

3.1 Genetic Algorithm for Solving the Episode Selection Problem

To solve an episode selection problem, ESP($\mathcal{D}, \psi$), with a genetic algorithm (GA) we need to represent the reduced datasets as individuals of the population and define the fitness function.

Each chromosome is encoded as a string of binary values representing which episodes in the input dataset $\mathcal{D}$ are selected. Each individual thus represents a reduced dataset $\mathcal{D}' \subseteq \mathcal{D}$ containing the selected episodes. The length of the string is the number of episodes in $\mathcal{D}$, $|\mathcal{D}|$. Each gene of the chromosome represents the presence of the corresponding episode in $\mathcal{D}'$. Formally, let $\mathcal{D} = \{e_1, e_2, \ldots, e_M\}$ be the input dataset, where e_i is the i-th episode in $\mathcal{D}$ and $|\mathcal{D}| = M$. Let X be the space of all possible individuals. Then, every $x \in X$ is defined as $x = x_1, x_2, \ldots, x_M$, where x_i is the i-th gene of the individual and $x_i \in \{0, 1\}$. The mapping function from the genotype to the phenotype is $\rho : X \to 2^{\mathcal{D}}$ so that given an individual $x \in X$, the corresponding reduced dataset is $\rho(x) = \{e_i \in \mathcal{D} \mid x_i = 1\}$.

The fitness of each $x \in X$ is computed using the ESP metric ψ (Definition 2), just by applying ψ to $\rho(x)$. Thus, the fitness function, $\phi : X \to \mathbb{R}$, is defined as $\phi(x) = \psi(\rho(x))$. However, we need to define a way to estimate the performance of the best policy that can be learned by an RL agent in a specific RL framework θ for each subset. A direct approach to obtain that estimate is to perform the RL process to learn a policy $\pi_{\mathcal{D}'}$ from $\mathcal{D}'$, and then evaluate that policy through executions in the environment. We define the estimate of the performance for $\mathcal{D}'$ obtained by RL as the average discounted return obtained in those episodes:

$$\hat{\mathcal{J}}_{RL}(\mathcal{D}') = \frac{1}{|\mathcal{D}'|} \sum_{e \in \mathcal{D}'} \sum_{t=0}^{t_e} \gamma^t r_{e,t} \tag{2}$$

where t_e and $r_{e,t}$ represent the last time step of episode $e \in \mathcal{D}'$ and the reward obtained at time step t in that episode, respectively.

3.2 Running Example with Frozen Lake

As a running example to show the ability of the GA to reduce the size of the dataset, we utilize the Frozen Lake environment. This is a grid-like environment from the Toy Text section of OpenAI's Gymnasium[1]. The goal is to reach the treasure from the starting position. It is a discrete, bi-dimensional environment with a delayed reward.

Our running example, uses a 12x12 grid map (see Figure 1), with initial and goal positions randomly placed within the grid. Additionally, approximately 20% of the grid's surface is randomly populated with holes.

The input dataset, denoted as $\mathcal{D}_{env}$, is generated through an online RL process in the new environment using the online Q-learning algorithm [21] with an ϵ-greedy approach. $\mathcal{D}_{env}$ is constructed by running the sub-optimal model obtained during online training 100 times. Thus, $|\mathcal{D}_{env}| = 100$. We employ an offline tabular Q-learning algorithm as our fixed RL agent for evaluating dataset performance using Equation 2. After each learning process, we evaluate the resulting policy by running it 10 times, starting from a fixed initial state, and computing the average undiscounted return. This approach simplifies the metric, ensuring it yields binary values (0 or 1) for deterministic environments.

[1]https://gymnasium.farama.org/

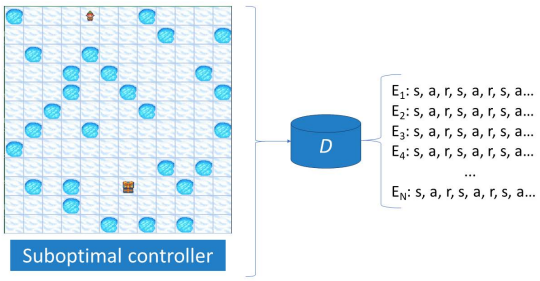


Figure 1: Example of dataset gathering.

From the GA perspective, we use $\psi(\mathcal{D}') = 0.8 \times \hat{\mathcal{J}}(\mathcal{D}' \mid \theta) + 0.2 \times \frac{1}{|\mathcal{D}'|}$ as ESP metric, where $\hat{\mathcal{J}}_{RL}(\mathcal{D}')$ is the evaluation value obtained for $\mathcal{D}'$ by the Q-learning algorithm. The genetic algorithm runs for 70 generations with a population size of 50 randomly sampled individuals containing approximately 50% of the total episodes. The selection process is performed by tournament selection with a tournament size of 5. Mutation switches gene values with a probability of $\frac{1}{|\mathcal{D}|}$ and the crossover is performed gene-wise by selecting the gene from one of two predecessors.

3.3 Results

Figure 2 illustrates the evolutionary trends (evolution of the policy and evolution of the number of selected episodes) resulting from five executions of the genetic algorithm. Notably, the dataset generated by the genetic algorithm consistently retains approximately 10-20% of the original dataset's episode count, all while preserving the optimal policy.

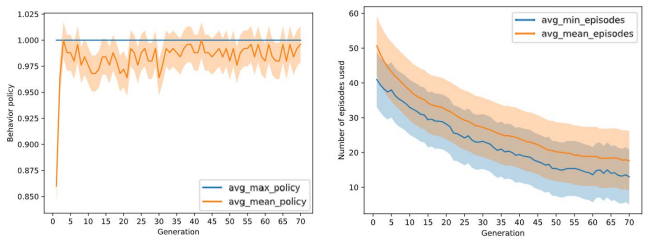


Figure 2: RL genetic algorithm results. On the left is the policy evolution and on the right is the number of episodes evolution. The orange lines represent the population's average values and the blue lines the best individual of the population.

4 Image-based Heuristics

It is important to note that using the previous scheme each time the fitness function is computed we need to compute $\hat{\mathcal{J}}_{RL}(\mathcal{D}')$ and subsequently learn $\pi_{\mathcal{D}'}$ under the framework θ. Then, the

described method is not useful when it is necessary to estimate the performance of a dataset quickly. It would be preferable to have a predictive model that allows obtaining a rapid estimate directly from the entire dataset at once. We define formally the problem of finding this model as the *Performance Prediction Problem (P3)*.

Definition 3 (Performance Prediction Problem (P3)). *Let $\mathcal{D}$ be a set of episodes and θ be an RL framework (algorithm, parameters, etc.). The P3 consists of finding a function $\hat{\mathcal{J}}_M : \mathcal{D} \rightarrow \mathbb{R}$ that approximates the return that would be obtained with the best policy, $\pi_{\mathcal{D}}$, that can be learned from $\mathcal{D}$ by an RL agent given θ: $\hat{\mathcal{J}}_M(\mathcal{D} \mid \theta) \approx \mathcal{J}(\pi_{\mathcal{D}} \mid \theta)$.*

Given Definition 3, we need to obtain a function that can approximate the return obtained with the best policy. This kind of function does not exist or it can not be computed mathematically to do it quickly. Thus we propose to use a neural network (NN) model as our function approximator.

In most cases, the datasets are composed of numerical values. The representation of states of the environment is a set of tuples with a numerical value for each feature (like position x and y). To prepare datasets for input into predictive models, a suitable representation is essential. If we use the basic numerical representation we would need to consider each episode E in the dataset with its corresponding experience tuples T, and the state S usually has more than one dimension. Then, the input data into the model would correspond to $E \times T \times S$ features which is impractical in most cases. Thus, we adopt a novel approach by representing entire datasets as images, tailored to the specific domain characteristics from which the dataset originates.

Representing datasets as images arises from the potential of visualizing the entire dataset in a single image. In addition, we consider the significance of spatial information in determining dataset quality. Deep neural networks, particularly convolutional neural networks (CNNs), stand to benefit from this data format, leveraging their ability to extract features from spatial representations.

4.1 Representing Datasets as Images

Representing datasets as images depends on the state dimensionality or the environment dynamics. Since the representation of datasets as images must be adapted to each domain, it is difficult to define a general method to define the images. Then, we propose different alternatives. We study their applicability by selecting three domains that reflect these characteristics: *Frozen Lake*, *Mountain Car* and *Acrobot*. These domains were selected to cover a range of RL environment characteristics, including state space (discrete or continuous), the number of state space dimensions, and the type of reward (delayed or instant).

Mountain Car is a classic control problem also from OpenAI's Gymnasium. The goal is to reach the top of the hill using momentum. It is a continuous, bi-dimensional environment with a delayed reward. Originally, Gymnasium's implementation featured an instant reward, but it was modified to match the characteristics of Frozen Lake, with a final goal to achieve. **Acrobot** is another classic control problem. It consists of two pendulums aiming to surpass a given height. This domain has a continuous, 6-dimensional state space with instant reward. Snapshots from these environments are shown in the top row of Figure 3.

Our image representation focuses solely on the states within the datasets. While actions could also be included, we opt to omit them from the images. This decision stems from the observation that actions can often be interpreted based solely on state distribution, and integrating them into images may not always be straightforward. Additionally, our experimental results demonstrate that utilizing only state information yields sufficiently accurate results. We propose two different ways of representing datasets as images: state (or state abstractions) scatter plots and rendered images.

State (Abstractions) Scatter Plots. The idea is to use a straightforward approach by creating scatter plots of all states, providing a visualization of the spatial distribution. For bi-dimensional state spaces, this method captures the bi-dimensional structure of the environment and provides additional contextual information. Points represent the states, with the start position indicated by a red point and the goal position by a green point or area. In high-dimensional state spaces, generating those scatter plots is unfeasible. One solution is to employ dimensionality reduction techniques such as Principal Component Analysis (PCA) [17], t-SNE [24] and autoencoders [3] to transform states into two-dimensional spaces for visualization. After testing with the three approximations, we decided to use autoencoders since the others obtained worse results. Then we represented the bi-dimensional abstraction of the original states with a scatter plot using the latent representation of the autoencoders. The middle row of Figure 3 shows examples of each environment using this representation.

Rendered Images. The idea is to use rendered versions of the environments.[2] The first step is gathering the frames of each state represented in the dataset. The images are then converted to grayscale and thresholded to separate the background (black) from the environment objects (white). This step reduces the number of channels that feed CNN and captures the motion in the environment. Next, we sum all the pixel values from the dataset frames and normalize the result between 0 and 255. Examples for each environment are shown in the bottom row of Figure 3.

Formally, the input value for the CNN model would be an image obtained by transforming the whole dataset $\mathcal{D}$ using a function $f_{abstraction}(\mathcal{D})$ or $f_{rendered}(\mathcal{D})$. The value to predict would be $\hat{\mathcal{J}}_{RL}(\mathcal{D}')$.

4.2 Experimental Results

In this section, we evaluate the CNN models in each environment to validate them as heuristics for our GA. Additionally, we showcase experiments where we compare the reduction benefits of using the GA with the CNN models against using the complete datasets.

4.2.1 CNN models experiments. Usually, in RL a change in the transition function means that a new policy must be learned. However, the spatial distribution of states will share characteristics in similar tasks. Hence, if we can train the CNN models for some specific transition functions, we would be able to transfer this knowledge to different transition functions in the same environment.

Given this assumption, we create various datasets with different configurations to modify these functions. Additionally, we modified

[2]Note this is not possible in all environments.

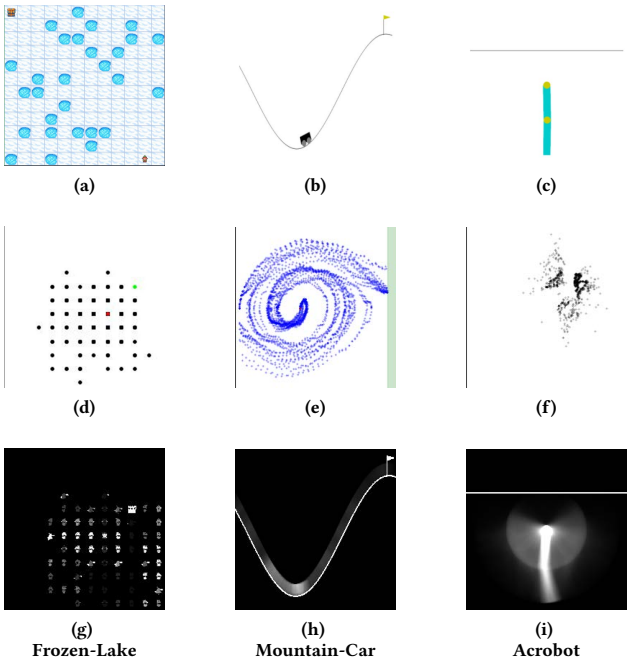


Figure 3: Representations of the Open AI's environments used in the paper. On the top row, is a snapshot of each environment. On the middle row, is the state representation/abstraction of the corresponding environments. On the bottom row, the rendered representation of the environments. Each column represents one domain: Frozen Lake, Mountain Car, and Acrobot respectively.

the state space when possible (Frozen Lake) and the quality of the datasets (Mountain Car and Acrobot) to prove the robustness of our solution. In Frozen Lake, we create 23 maps, where 20 are of size 12x12 and the remaining with increasing sizes 24x24, 48x48 and 96x96. All maps are created by placing the start and goal positions randomly and covering 20% of the map with holes. In this case, 10 of the 12x12 maps are used for training, 4 of them for validation and the rest joined with the bigger maps for testing. In Mountain Car, we create 4 configurations with different acceleration forces and gather datasets with varying policy quality. In this case, the training datasets were obtained with two specific values for the applied force and two policy qualities and the test datasets comprise two different applied forces and four policy qualities. Finally, in Acrobot we create 4 setups with different torque and pendulum lengths and obtain datasets following policies with different qualities. Training datasets used a single torque and pendulum length configuration for several policy qualities and the test set used the remaining configurations. We follow a similar approach to the one explained in Section 3.2 to generate the datasets from sub-optimal policies. After obtaining the datasets, we generate the images following the scheme in Figure 4. Images are generated using the GA that creates a bunch of examples of subsets.

Given the characteristics of the reward of each environment, the output of the CNN differs from one to another. In the case of Frozen Lake and Mountain Car, where the result is a value of 0 (not reaching the goal) or 1 (reaching the goal), the output is a value between 0 and 1, considered as the confidence level of being a good subset. In the case of Acrobot, the output of the CNN is a value between -500 and 0 (the steps needed to surpass the mark). These models are trained by receiving as input examples of images from subsets of the original datasets. Figure 5 shows the results from the CNN models using rendered (left column) and state (right column) representations across different domains, which are summarized as follows.

Frozen Lake. Both image representations of the environment use the same CNN model architecture. After training the models, the accuracy obtained for the scatter plotted representation is 99% and for the rendered images 92%. This accuracy is obtained using the threshold in the range $[0, 1]$ that maximizes the Area Under the Curve (AUC). As illustrated in Figures 5a and 5b, the representation using a scatter plot of the states can discriminate between "good" and "bad" datasets. However, with the rendered representation it is more difficult to make this differentiation. After analyzing the results, we found that the red pick around 0.0 confidence in Figure 5a corresponds to all the results for a specific map. This map has the goal close to the start point and it may be more difficult for the model to distinguish the existence of a path between the red and green points.

Mountain Car. Upon completion of training, the predictor trained with the images representing the states as points in two dimensions achieved an accuracy of approximately 79% on the test set. Using the rendered images the model barely surpasses the 50% accuracy. Given the results provided in Figure 5d, the state representation can accurately differentiate between cases. Additionally, "bad" cases are better discriminated than "good" ones. As expected from the low accuracy of the model, the rendered images (Figure 5c) are not suitable to represent "good" and "bad" cases in this environment.

Acrobot. Following training, an L1 error, which measures the average absolute differences between predicted and actual values, of approximately 22 was attained when utilizing rendered images. Conversely, an error of 25 was recorded when employing images of an autoencoder-based dimensionality reduction. Given the reward ranges from -500 to -70 these are good error values. In Figures 5e and 5f we provide the predicted labels against real labels scatter plots. Both representations distribute the results along the dashed diagonal line that assumes the perfect model. However, when using the autoencoder representation, the model demonstrates underperformance in higher value ranges. The rendered image model is capable of such distinction.

Already in Definition 3 we considered the hyperparameters and type of algorithm in the RL process. Given the results of the predictors, we can observe that the representation used is also a key factor for each environment. In our results, the scatter plots representation is better for the bi-dimensional environments. Conversely, the rendered images are more suitable for Acrobot.

These prediction models allow us to reduce the execution time of GA from 30-40% in the worst cases (Frozen Lake due to its simplicity) to 80-90% in the best scenarios (Acrobot due to the use of DQN as the offline training algorithm) compared to those cases in which

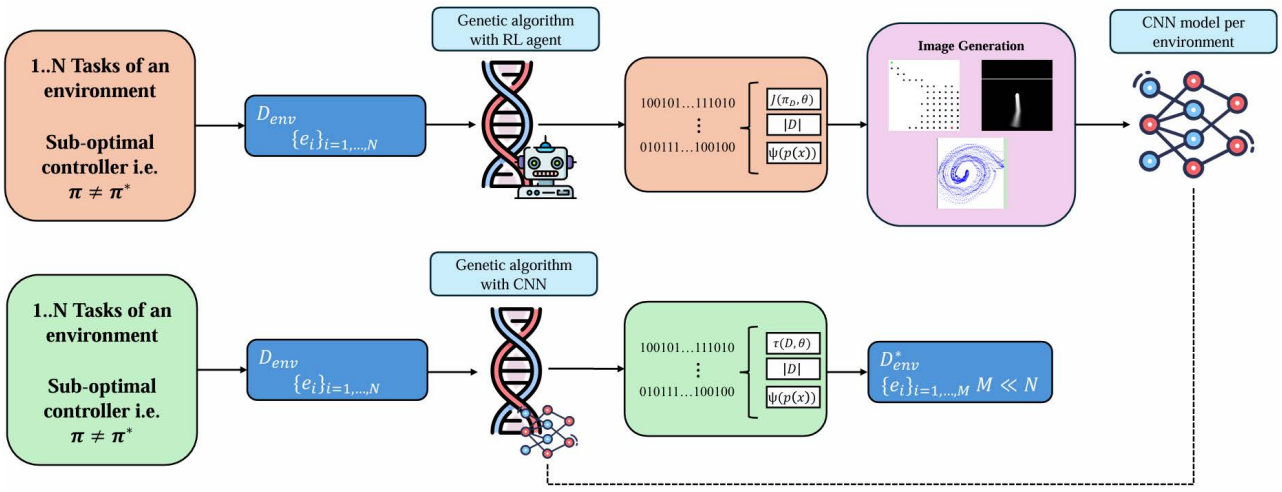


Figure 4: Scheme used to reduce the datasets. In orange, we consider the environment configurations used to train the CNN models. Labels for each image generated (square in pink) are obtained from Equation 1 given an RL training and evaluation process for each individual of the genetic algorithm. In green, we consider the environment configurations and datasets used to test our solution. The fitness function uses the trained CNN models. The output is a reduced dataset that maintains or improves the learned policy.

the CNN where not used in the GA both in the training and test sets.

4.2.2 GA experiments. Additionally, we compare the policies obtained from using the GA approach with our heuristic to use the complete dataset to perform the training process. We evaluate the performance of the best individual (best-reduced dataset) obtained when the ESP is solved by the GA using the CNN predictor, in comparison to the real value of the performance of the original dataset, computed by Equation 2 and using the policy learned by the offline RL algorithm. We use the same domains as before. Experiments are divided into two groups, described below.

In the first group, we evaluate the generalization ability of the method when datasets gather data from configurations with different transition functions and, additionally, in Frozen Lake, we consider datasets with modified state space. Specifically, in Frozen Lake, the 12x12 maps are joined as the modification of the transition function, and bigger maps are presented separately. In Mountain Car, we consider the different forces as the modification of the transition function. In Acrobot, we consider different lengths of the lower pendulum and the torque force applied to modify the transition function. Frozen Lake is the only environment with state space modifications because it is the only one with this possibility (changing the map size).

In the second group, we evaluate the generalization ability of the method when there are changes intrinsic to the RL training process by using different policy qualities to gather the data. Specifically, we include 3 policy values and a single epsilon value for both Mountain Car and Acrobot. Frozen Lake is not included in this group due to its learning simplicity and thus difficulty in obtaining different policy expertise.

All experiments use $\psi(\mathcal{D}) = 0.8 \times \hat{\mathcal{J}}(\mathcal{D} \mid \theta) + 0.2 \times \frac{1}{|\mathcal{D}|}$ as ESP metric, where $\hat{\mathcal{J}}(\mathcal{D} \mid \theta)$ is obtained by the trained CNN models used in Section 4.2. The initial population for the GA is generated randomly with 50% chance for each episode to appear in the given subset.

All configurations were executed 5 times. Table 1 shows the results for the first and second groups of experiments in terms of the average and standard deviation of the performance metric and the number of tuples used for both the complete dataset and the resulting subset from the GA process, depending on the type of images used for the CNNs. The results show that our method generally maintains (Frozen Lake) or improves (Mountain Car and Acrobot) the learned policy. This is not fulfilled in the rendered version of bigger maps in Frozen Lake. Moreover, we can observe between 70% to even 95% dataset size reduction in most cases.

In Figure 6, we present a comparative analysis of the performance of datasets generated through the genetic algorithm in contrast to random datasets of varying sizes. Note that we also include the complete dataset in this comparison. Each figure in the plot series corresponds to the training progression of the respective reinforcement learning (RL) algorithm. While training, the policy obtained at each interval was evaluated, with the x-axis denoting the training time and the y-axis representing the reward achieved. We executed each case 100 times and computed the mean value and standard deviation. In the case of Mountain Car and Acrobot, we use a moving average for visualization purposes. Results show that our solution outperforms the random selection and the complete dataset in the number of updates needed for the training to converge, rewards obtained, or both.

Table 1: Results of generalization of the method for the first and second group of experiments, separated by a double line. Column *Original* denotes the complete dataset. Columns *States* and *Rendered* refer to the type of images used for the CNNs (state scatter plots and rendered environment images, respectively). Domains are represented by one letter followed by the type of modification performed. Frozen Lake (F), Mountain Car (M), Acrobot (A). Modifications to transition function ($\mathcal{P}$), state space (XS where X is multiplying factor to the base 12x12), policy expertise (π') and epsilon (ϵ).

	Obtained reward			Number of tuples		
Task	Original	States	Rendered	Original	States	Rendered
F$\mathcal{P}$	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	100%	8.25%	11.72%
F2S	1.00 ± 0.00	1.00 ± 0.00	0.00 ± 0.00	100%	10.52%	34.26%
F3S	1.00 ± 0.00	1.00 ± 0.00	0.00 ± 0.00	100%	9.71%	19.40%
F4S	1.00 ± 0.00	1.00 ± 0.00	0.00 ± 0.00	100%	14.61%	37.91%
M$\mathcal{P}$	0.67 ± 0.50	0.80 ± 0.42	0.66 ± 0.50	100%	30.05%	38.58%
A$\mathcal{P}$	-341.92 ± 86.87	-157.73 ± 70.56	-183.99 ± 84.97	100%	6.65%	6.76%
Mπ'	0.50 ± 0.55	0.83 ± 0.41	0.83 ± 0.41	100%	33.32%	43.59%
Mϵ'	1.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	100%	31.36%	42.89%
Aπ'	-397.16 ± 85.58	-188.50 ± 38.94	-158.97 ± 59.65	100%	9.04%	3.53%
Aϵ	-439.87 ± 50.25	-381.59 ± 59.22	-163.82 ± 93.69	100%	9.16%	3.05%

5 Related work

In offline RL, data quality is an important factor. Data quality is not considered in many of the offline RL algorithms available in the literature diminishing their performance. We will divide this section between approaches that estimate dataset performance and dataset reduction. Additionally, we include a section that explores the usage of GA in data-related problems.

5.1 Dataset performance estimation

There are different approaches to estimating the performance of datasets in Offline RL. Some metrics have been developed to quantify the data quality like Estimated Relative Return Improvement (ERI) [22] or TQ and SACo [19]. However, these factors do not actually reflect the possible performance after a training process with the dataset. The metrics cannot determine the quality of a dataset since they are based on averaging the expected returns from the episodes that constitute the dataset. Additionally, they do not take into consideration the existence of datasets created from a mixture of policies and still rely on the expectation formula (Equation 1).

Such considerations are reflected in algorithms like LBRAC-v [29]. The algorithm assumes the datasets are generated from different behavior policies. With this assumption, it tries to learn a latent variable model that is capable of grouping all the trajectories in the dataset into their corresponding behavior policy. This latent variable is used to solve the degeneration problem from the BRAC-v [25] algorithm. However, it still does not estimate the performance of a dataset as a whole.

An example of dataset estimation is DVORL [1]. This algorithm has a sub-module that performs data valuation which consists of estimating datum quality in overall performance [4, 28]. This is performed on a subset of the dataset to make the best possible selection for the target task. However, this algorithm assumes the existence of a target dataset, which may not exist or be unknown. The algorithm is unusable without the target dataset since it is used for a KL divergence metric [9] during the learning process. We can estimate without the need for an auxiliary target dataset.

5.2 Dataset reduction

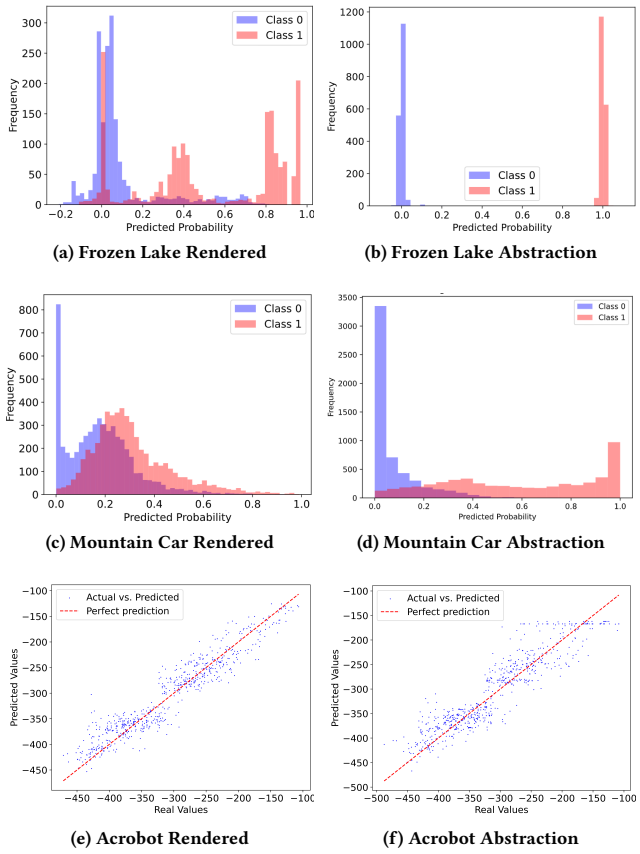
Offline Imitation Learning algorithms are useful for learning from demonstrations. Most of them are based on Behavioural Cloning (BC) [2] which uses supervised learning techniques to learn. However, the main problem with this type of algorithm is that they imitate equally data with expert and sub-optimal behavior. This leads to bad performances. Hence there are various approaches to reduce datasets to keep only the useful data for the training process.

The approaches to solving this issue are based on using discriminators to distinguish between expert data and non-expert data. In *Discriminator-Weighted Offline Imitation Learning from Sub-optimal Demonstrations* [27] the idea is to build a discriminator on top of a BC algorithm to weight the input data depending on its expertise. They use a cycle to learn both the model and discriminator. COIL [14] assumes that each trajectory has been collected by an independent policy. It measures the KL divergence of each trajectory to determine the similarity with the learned policy and filter the dataset. Other examples like 2IWIL and IC-GAIL [26] based their discrimination strategies on importance weighting or Generative Adversarial Networks (GANs) [5] using GAIL [6] as their base algorithm. Our approach creates a new "cleaned" dataset instead of using the original one.

5.3 GAs for dataset management

Genetic algorithms (GAs) have been widely used for feature selection and dimensionality reduction in machine learning. By leveraging evolutionary optimization, GAs facilitate the selection of optimal feature subsets, improving computational efficiency and predictive performance in various models.

Raymer et al. [18] introduced a GA-based method that optimizes feature weights and employs a masking vector for improved selection, demonstrating effective dimensionality reduction while preserving classification accuracy. Mohammed et al. [16] extended this idea by integrating GAs with artificial neural networks (ANNs) to enhance feature reduction in big data applications, showing improved computational efficiency and classification performance.



(a) Frozen Lake Rendered (b) Frozen Lake Abstraction

(c) Mountain Car Rendered (d) Mountain Car Abstraction

(e) Acrobot Rendered (f) Acrobot Abstraction

Figure 5: Results from the CNN models using rendered (left column) and state (right column) representations across different domains. Each row corresponds to a different domain: Frozen Lake, Mountain Car, and Acrobot. The plots for Frozen Lake and Mountain Car show the distribution of confidence provided by the model for "good" (red) and "bad" (blue) datasets. The plots for Acrobot compare the true labels of the test set instances with the predicted values, with the dashed red line representing the ideal scenario.

Similarly, [20] applied GAs for data reduction and non-algebraic feature construction, successfully minimizing dataset sizes while retaining predictive power.

6 Conclusions

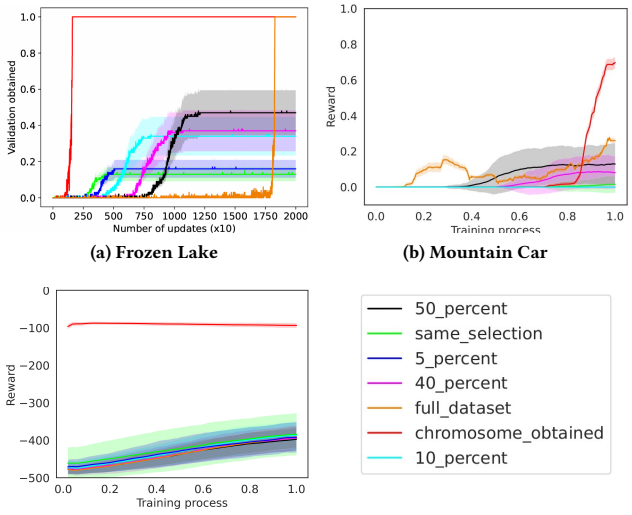
Our paper proposes an efficient method for reducing dataset size in offline reinforcement learning while maintaining or improving the resulting policy quality. We use a genetic algorithm to optimize this process. Additionally, we present an efficient method for estimating the performance of a dataset in Offline RL without a conventional training process. We tackle different RL problems such as continuous and high dimensional state spaces and delayed and immediate reward functions. Hence, we have represented datasets as images and used a convolutional neural network (CNN) as our approximation function.

We evaluated our approach in three well-known RL domains, Frozen Lake, Mountain Car, and Acrobot, achieving significant dataset size reductions while keeping or enhancing policy quality in different scenarios. We also found potential for reusing learned models and efficiently estimating fitness functions using CNNs.

In the future, we plan to test our method in more complex environments, continuous action spaces, and different dataset sizes to better understand the method's scalability. Adapting our method to other representation forms is another possibility. From the GA point of view, we plan to test our method on pure multiobjective settings with dedicated algorithms. Additionally, exploring new RL applications could lead to innovative dataset-reduction techniques. This research lays the foundation for refining and expanding efficient dataset reduction methods in offline and batch RL.

Acknowledgments

This work was partially developed while Fernando Fernández was a visiting researcher at the University of Texas at Austin founded by the Universidad Carlos III de Madrid and Ministerio de Universidades under the program "Estancias de personal docente y/o investigador senior en centros extranjeros" grant number PRX22/00198. This work was partially funded by grant PID2021-127647NB-C21 from MICIU/AEI/10.13039/501100011033, by the ERDF "A way of making Europe", and by the Madrid Government under the Multiannual Agreement with UC3M in the line of Excellence of University Professors (EPUC3M17) in the context of the V PRICIT (Regional



(a) Frozen Lake (b) Mountain Car

(c) Acrobot

Figure 6: Comparison of performance of the genetic algorithm against random and full datasets for Frozen Lake, Mountain Car, and Acrobot. "chromosome_obtained" stands for the selection achieved using the GA, "same_selection" stands for a random selection using the same number of episodes as the solution from the GA and the percentages correspond to a random selection of episodes from the dataset corresponding to the given proportion.

Programme of Research and Technological Innovation). We thank Peter Stone for his insightful comments and ideas that helped us during the process.

References

[1] Amir Abolfazli, Gregory Palmer, and Daniel Kudenko. 2022. Data Valuation for Offline Reinforcement Learning. *arXiv* 2205.09550 (2022), 9 pages.

[2] Michael Bain and Claude Sammut. 1999. A Framework for Behavioural Cloning. In *Machine Intelligence 15, Intelligent Agents*. Oxford University, Oxford, 103–129.

[3] Dor Bank, Noam Koenigstein, and Raja Giryes. 2023. Autoencoders. In *Machine Learning for Data Science Handbook: Data Mining and Knowledge Discovery Handbook*. Springer International Publishing, 353–374.

[4] Amirata Ghorbani and Jame Zou. 2019. Data shapley: Equitable valuation of data for machine learning. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR, 2242–2251.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems. Proceedings of the twenty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2014)*, Vol. 27. Curran Associates, 2672–2680.

[6] Jonathan Ho and Stefano Ermon. 2016. Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems. Proceedings of the thirtieth Annual Conference on Neural Information Processing Systems (NeurIPS 2016)*, Vol. 29. Curran Associates, 4565–4573.

[7] John H. Holland. 1975. *Adaptation in Natural and Artificial Systems.* MIT Press.

[8] Varun Raj Kompella, Thomas Walsh, Samuel Barrett, Peter R. Wurman, and Peter Stone. 2023. Event Tables for Efficient Experience Replay. *Transactions on Machine Learning Research* (2023), 32 pages.

[9] Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The Annals of Mathematical Statistics* 22(1) (1951), 79–86.

[10] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative Q-Learning for Offline Reinforcement Learning. In *Advances in Neural Information Processing Systems. Proceedings of the thirty-fourth Annual Conference on Neural Information Processing Systems (NeurIPS 2020)*, Vol. 33. Curran Associates, 1179–1191.

[11] Sascha Lange, Thomas Gabel, and Martin Riedmiller. 2012. *Batch reinforcement learning.* Springer. 45–73 pages.

[12] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521 (2015), 436–444.

[13] Sergey Levine, Aviral Kumar, G. Tucker, and Justin Fu. 2020. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. *ArXiv* 2005.01643 (2020), 43 pages.

[14] Minghuan Liu, Hanye Zhao, Zhengyu Yang, Jian Shen, Weinan Zhang, Li Zhao, and Tie-Yan Liu. 2021. Curriculum Offline Imitating Learning. In *Advances in Neural Information Processing Systems. Proceedings of the thirty-fifth Annual Conference on Neural Information Processing Systems (NeurIPS 2021).* Curran Associates, 1–12.

[15] Z. Michalewicz. 1996. *Genetic algorithms + data structures = evolution programs (3nd, extended ed.).* Springer, New York, NY, USA.

[16] Tareq Abed Mohammed, Shaymaa Alhayali, Oguz Bayat, and Osman N Uçan. 2018. Feature reduction based on hybrid efficient weighted gene genetic algorithms with artificial neural network for machine learning problems in the big data. *Scientific Programming* 2018, 1 (2018), 2691759.

[17] Karl Pearson. 1901. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 2(11) (1901), 559–572.

[18] M.L. Raymer, W.F. Punch, E.D. Goodman, L.A. Kuhn, and A.K. Jain. 2000. Dimensionality reduction using genetic algorithms. *IEEE Transactions on Evolutionary Computation* 4, 2 (2000), 164–171.

[19] Kajetan Schweighofer, Markus Hofmarcher, Marius-Constantin Dinu, Philipp Renz, Angela Bitto-Nemling, Vihang Prakash Patil, and Sepp Hochreiter. 2021. Understanding the Effects of Dataset Characteristics on Offline Reinforcement Learning. In *Deep RL Workshop NeurIPS 2021*. 19 pages.

[20] Leila S. Shafti and Eduardo Pérez. 2008. Data Reduction by Genetic Algorithms and Non-Algebraic Feature Construction: A Case Study. In *2008 Eighth International Conference on Hybrid Intelligent Systems*. 573–578.

[21] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press.

[22] Phillip Swazinna, Steffen Udluft, and Thomas Runkler. 2021. Measuring Data Quality for Dataset Selection in Offline Reinforcement Learning. In *Proceedings of the 2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1–8.

[23] Matthew E Taylor and Peter Stone. 2009. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research* 10, 56 (2009), 1633–1685.

[24] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* 9, 86 (2008), 2579–2605.

[25] Yifan Wu, George Tucker, and Ofir Nachum. 2019. Behavior Regularized Offline Reinforcement Learning. *arXiv* 1911.11361 (2019), 25 pages.

[26] Yueh-Hua Wu, Nontawat Charoenphakdee, Han Bao, Voot Tangkaratt, and Masashi Sugiyama. 2019. Imitation Learning from Imperfect Demonstration. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97. PMLR, 6818–6827.

[27] Haoran Xu, Xianyuan Zhan, Honglei Yin, and Huiling Qin. 2022. Discriminator-Weighted Offline Imitation Learning from Suboptimal Demonstrations. In *Proceedings of the 39th International Conference on Machine Learning*, Vol. 162. PMLR, 24725–24742.

[28] Jinsung Yoon, Sercan Arik, and Tomas Pfister. 2020. Data valuation using reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*. PMLR, 10842–10851.

[29] Guoxi Zhang and Hisashi Kashima. 2023. Behavior estimation from multi-source data for offline reinforcement learning. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence*. AAAI Press, 11201–11209.

[30] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. 2020. Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey. In *Proceedings of the 2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 737–744.

[31] Zhuangdi Zhu, Kaixiang Lin, Anil K. Jain, and Jiayu Zhou. 2020. Transfer Learning in Deep Reinforcement Learning: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45 (2020), 13344–13362.