Nazarbayev University

School of Engineering and Digital Sciences (SEDS)

Technical Report

Uniformly Distributed Data Effects in Offline RL: A Case Study in Gridworld Setting

NAZARBAYEV UNIVERSITY

Department of Computer Science

May 23, 2024

Uniformly Distributed Data Effects in Offline RL: A Case Study in Gridworld Setting

Kuanysh Tokayev and Jurn-Gyu Park
School of Engineering and Digital Sciences
Nazarbayev University, Astana, Kazakhstan
kuanysh.tokayev, jurn.park@nu.edu.kz

***Abstract*—In the emerging landscape of off-policy reinforcement learning (RL), challenges arise due to the significant costs and risks tied to data collection. To address these issues, there is an alternative path for transitioning from off-policy to offline RL, known for its fixed data collection practices. This stands in contrast to online algorithms, which are sensitive to changes in data during the learning phase. However, the inherent challenge of offline RL lies in its limited interaction with the environment, resulting in inadequate data coverage. Hence, we underscore the convenient application of offline RL, 1) starting from the collection of a static dataset, 2) followed by the training of offline RL models, and 3) culminating with testing in the same environment as off-policy RL methodologies. This involves the utilization of a *uniform dataset* gathered systematically via non-arbitrary action selection, covering all possible states of the environment. Utilizing the proposed approach, the Offline RL model employing a Multi-Layer Perceptron (MLP) achieves a testing accuracy that falls within 1% of the results obtained by the off-policy RL agent. Moreover, we provide a practical guide with datasets, offering valuable tutorials on the application of Offline RL in Gridworld-based real-world applications. The guide can be found in this GitHub[1] repository.**

***Index Terms*—Offline RL, Data Distribution, Deep Learning, DQN, Machine Learning, Tutorial**

I. Introduction

Reinforcement Learning (RL) has shown great potential in solving complex decision-making problems [1].

However, its wide distribution is hindered by a fundamental limitation - the online nature of RL algorithms [2]. This online nature makes data collection costly and risky [3] [4], especially in areas such as robotics, autonomous driving, and healthcare [2]. Due to the limitations of online learning, there is a growing interest in Offline RL, which involves learning from pre-collected datasets while avoiding costly and risky online data collection [5]. Nonetheless, the major limitation to the widespread adoption of Offline RL is the absence of continuous exploration of the environment, a crucial component of online RL algorithms. This deficiency in exploration results in insufficient dataset coverage for any Offline RL algorithm [6] [7]. Moreover, the complex nature of both off-policy and offline RL algorithms pose a significant challenge in the adoption of methodologies enhancing their synergy and bridging the gap between them [2]. This knowledge gap hinders the adoption and development of Offline RL algorithms. Current approaches aimed at improving the performance of Offline RL models operating with potentially biased data include pessimistic strategies and dataset-centric interventions. Pessimistic strategies in Offline RL often include the construction of a penalty function (pessimism) to ensure a conservative estimation of the policy. By incorporating this penalty function, the policy avoids being overly optimistic about the data, thereby reducing its susceptibility to potential noise or suboptimal data [6]. On the other hand, dataset-centric interventions primarily aim to improve data collection procedures, thereby improving data quality and consequently yielding higher accuracies for trained Offline RL models [8].

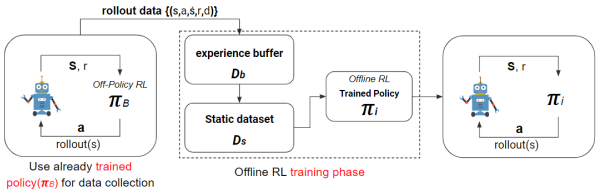


Fig. 1: An Overview.

Our approach based on the data-centric interventions proposes the following steps to solve the research problem: First and foremost, it is imperative to adopt a step-by-step approach to acquire a uniform dataset from the off-policy RL setup to train an Offline RL model independently in the Gridworld environment (see Fig. 1). The environment will be constructed using the Open AI Gym package as the first step, followed by the creation of a custom Gridworld environment using basic packages. This approach creates a controlled environment for Offline RL experiments, allowing researchers to evaluate and compare different approximation algorithms systematically. To assess the *scalability* of our approach, it is tested across various Gridworld sizes.

Secondly, to obtain a more objective and fair evaluation of Offline RL algorithms, various approximation models, and simple value functions are tested and compared using collected static datasets (both uniform and non-uniform) (see Fig. 1). This rigorous evaluation process will help identify the strengths and weaknesses of various model approximations

[1]K. Tokayev, "Offline RL Dataset," https://github.com/ZeroNegativity/OfflineRLDataset.git, 2023.

This technical report is an extension of paper [BigComp'24]:
Uniformly Distributed Data Effects in Offline RL: A Case Study in Gridworld Setting.
DOI:10.1109/BigComp60711.2024.00033

and value functions, providing insight into their suitability for Offline RL applications.

The contributions of this research are as follows:

- ***Dataset collection***: Uniformly and non-uniformly distributed static datasets were obtained from the off-policy RL experience that was used to train an Offline RL model. As a result, agents trained on a uniform dataset predominate, demonstrating superior performance compared to agents trained on non-uniform datasets.
- ***Offline RL model training***: An efficient learning process for Offline RL models has been empirically demonstrated, which includes 1) fitting various value functions and 2) optimizing different model approximators on static datasets.
- ***Gridworld setup testing***: The performance of the Offline RL agent is assessed within both custom and OpenAI Gym Gridworld environments. Furthermore, the evaluation was extended by increasing the dimensionality of the Gridworld setup to rigorously test the proposed methodology.

II. Motivation and Related Work

In this section, we discuss the motivation behind this paper and explore related research works.

A. Motivation

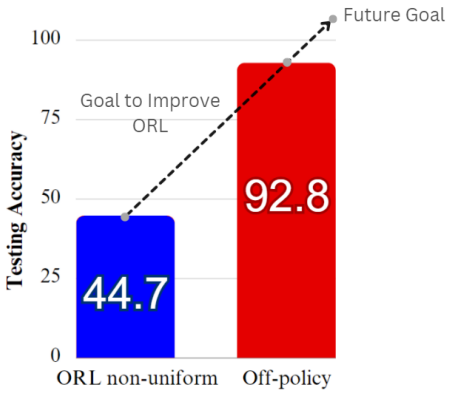


Fig. 2: Motivation Example.

Note: Term **ORL** represents Offline Reinforcement Learning.

Fig. 2 highlights a huge gap in testing accuracy between off-policy RL results obtained from training with DQN and non-uniformly collected simple ORL trained on MLP, which replicates the parameters of the DQN. This significant gap observed in these results can be due to a *distributional shift* in the collected dataset. In our case, it occurs because the function approximator in Offline RL is trained under one distribution (non-uniform dataset) while the evaluation of its accuracy happens on a different distribution [2]. In other words, the Offline RL policy is trained on one set of visited states, while tested on another. This could occur as a result of the random initial state selection during training, the epsilon-greedy policy of action selection, or, more subtly, the action of maximizing the expected return [2]. The solution to this issue is to train and test the policy on identical distributions. However, during testing, the state of the environment is selected randomly, making it impossible to predict its distribution. This requires the adoption of a uniformly collected dataset that comprehensively stores every state of the environment, thereby ensuring the inclusion of the testing distribution within the collected dataset. Thus, the primary motivation of this paper is to present a straightforward and effective methodology using the uniformly collected dataset for the seamless transition from off-policy to offline learning, while ensuring no loss in test accuracy performance.

B. Related Work

Algorithm-centric ORL: Offline RL models aim to learn policies from a static dataset without interacting with the real environment [2]. Deploying off-policy RL algorithms directly in an offline setting faces a challenge known as the distributional shift problem, which can lead to a significant drop in performance, as highlighted in [3]. To mitigate this issue, model-free algorithms strive to either constrain the learned policy close to the behavior policy [3] [9] or penalize the values of state-action pairs that lie outside the distribution of the collected data [10] [11]. On the other hand, model-based algorithms [12] [13] simulate the real environment to generate additional data for policy training. Most of these approaches necessitate imposing pessimistic constraints on the learned policy.

Data-centric ORL: Recent advancements in Offline RL have predominantly utilized datasets generated by task-oriented policies. For instance, the benchmark suites D4RL [14] and RL Unplugged [15] [16] comprise datasets acquired through policies designed to optimize rewards specific to particular tasks. These datasets are primarily sourced from either the replay buffers of the training process or trajectories obtained at a particular point during the training phase. This methodology of static dataset collection has found applications in various domains, including continuous control for robotics [3], navigation [17], industrial control [18], and Atari video games [19]. Another state-of-the-art approach consists of using unsupervised exploratory data collection [8] [20] which is more suited for multi-task ORL algorithms. This approach includes relabeling data with a downstream reward after collection and before training a policy with Offline RL [8].

In contrast, our proposed methodology acquires a static uniform dataset, eliminating the possibility of distributional shifts. As a result, our algorithm does not require the imposition of additional pessimistic constraints on the learned policy. Our methodology shares similarities with datasets gathered by supervised task-specific policies. However, our data collection process occurs after the completion of the training phase, ensuring the acquisition of a uniform dataset.

III. Preliminaries

A. Reinforcement Learning (RL)

1) ***Generic Off-policy RL****:* is an approach that estimates the value function of a policy using a set of transition data generated by another policy [21]. The algorithm can take several input variables, including the current state of the environment. The agent applies the maximum Q-value to the environment using the specified states and actions, resulting in a new state and reward from the environment [22]. The value function estimate is updated using the Bellman operator, which considers observed transitions:

$$\Delta\theta = \alpha\Big[\beta(s) - Q_\theta(s,a)\Big]\nabla Q_\theta(s,a). \tag{1}$$

Here, $Q_\theta(s,a)$ is a Q-function approximated using parameters/weights θ, $\beta(s)$ denotes the value of the approximated Bellman operator, and α represents the learning rate. The Bellman Operator equation is given by:

$$\beta(s) = r(s') + \gamma \max Q_\theta(s',a'). \tag{2}$$

In this equation, r represents the reward, γ is the discount factor, and $Q_\theta(s',a')$ corresponds to the Q-values for the next state s' and action a'.

To facilitate learning, experience replay is employed, where past experience in the form of a tuple (s,a,s',r') is stored in the buffer list. This approach allows the algorithm to learn from a variety of experiences, improving its overall performance. The primary advantage of DQN over online Q learning is that DQN can learn from data more effectively through updating its weights by extracting information from each step of experience multiple times [23]. Another advantage of DQN is that it learns from randomly sampled experiences from the replay stored as *minibatch* rather than consecutive experiences which leads to the decrease in updates variance [24]. Consecutively DQN demonstrates resilience against getting trapped in poor local minima [24]. For instance, if the optimal action is moving upward, algorithms like online Q learning focus too heavily on upward movements, risking getting stuck in local minima. On the other hand, DQN avoids this issue by smoothing out such biases by averaging its past experiences [25].
The main idea of the DQN in our work is to connect the reinforcement learning algorithm to a deep neural network to learn directly from the state representation of Gridworld (see Fig. 3)

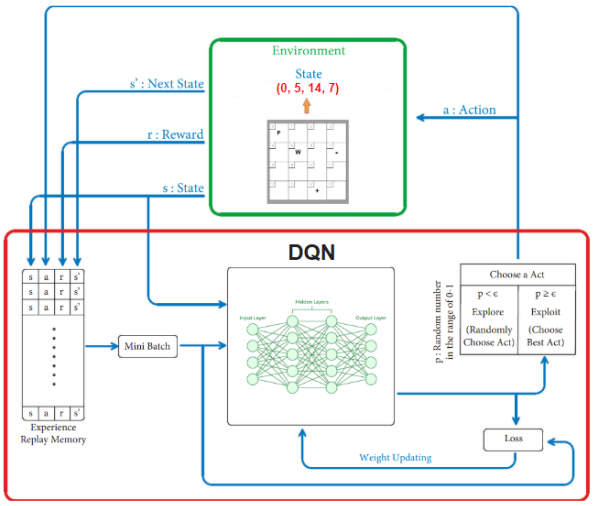


Fig. 3: DQN representation.

As mentioned earlier algorithm learns from the sample of experiences which is named as *minibatch* represented as tuples of $e = (s_t, a_t, r_{t+1}, s'_{t+1})$ which are stored in the *experience replay buffer* which is represented as $D = e_1, ..., e_N$ (see Fig. 4). The experiences are inserted into the replay buffer during each step of the agent which will be initialized after the agent performs an action. The replay buffer has a constant size meaning that if the number of experiences exceeds the size of the buffer then new experiences overwrite the oldest experiences.

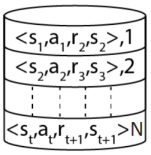


Fig. 4: Experince replay representation.

The minibatch is randomly sampled from the replay buffer during every step. To maintain a proper sampling process the minibatch size must be less or equal to the size of the replay buffer. The pseudocode of the DQN implemented in this work is the following [24]:

In the Deep Q-Network (DQN) algorithm 1, the first step is to initialize the replay memory D with capacity N. This replay memory will store past experiences for training the neural network (line 1). Next, the action-value function Q is initialized with random weights θ. This function represents the neural network that approximates the Q-values for state-action pairs (line 2). The algorithm iterates over a fixed number of episodes, denoted as M. Each episode represents a complete interaction cycle with the environment (line 3). Within each episode, the algorithm starts by initializing the state s_1 from which the agent will begin its exploration of the environment (line 4). Then, for each timestep within the episode, the agent chooses an action based on an exploration-exploitation strat-

Algorithm 1 Deep Q-Network (DQN)

1: Initialize replay memory D with capacity N
2: Initialize action-value function Q with random weights θ
3: **for** episode $= 1$ to M **do**
4: Initialize state s_1
5: **for** timestep $= 1$ to T **do**
6: With probability ϵ select a random action a_t
7: Otherwise select $a_t = \text{argmax}_a Q(s_t, a; \theta)$
8: Execute action a_t in the environment and observe reward r_t and next state s_{t+1}
9: Store transition (s_t, a_t, r_t, s_{t+1}) in D
10: Sample random minibatch of transitions (s_i, a_i, r_i, s_{i+1}) from D
11: Set target for Q-learning update: $y_i = \begin{cases} r_i & \text{episode terminates at } i+1 \\ r_i + \gamma \max_{a'} Q(s_{i+1}, a'; \theta) & \text{otherwise} \end{cases}$
12: Perform gradient descent on $(y_i - Q(s_i, a_i; \theta))^2$ with respect to the network parameters θ
13: **end for**
14: **end for**

Algorithm 2 Deep Q-Network (DQN) in Gridworld 4 x 4

1: Initialize replay memory D to capacity N
2: Initialize action-value function Q with random weights
3: Initialize state s
4: **for** episode = 1 to M **do**
5: Initialize $test\_game$ as a Gridworld of size 4 in sequence $s_1 = x_1$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
6: **for** step = 1 to T **do**
7: With probability ϵ select a random action a_t
8: otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a : \theta)$
9: Execute action a_t in environment and observe reward r_t and new state s_{t+1}
10: Store transition $(\phi(s_t), a_t, r_t, \phi(s_{t+1}))$ in D
11: Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
12: Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
13: Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
14: **if** reward $= 10$ **then** Game won! **end for**
15: **end if**
16: **if** reward $= -10$ and $step > 15$ **then** Game lost! **end for**
17: **end if**
18: **end for**
19: **end for**

egy. With probability ϵ, a random action is selected, otherwise, the action with the highest Q-value is chosen (line 5-6). The chosen action is executed in the environment, leading to a reward r_t and the observation of the next state s_{t+1} (line 7). The transition (s_t, a_t, r_t, s_{t+1}) is stored in the replay memory D, which will be later used for training the neural network (line 8). To train the neural network, a random minibatch of transitions (s_i, a_i, r_i, s_{i+1}) is sampled from the replay memory D (line 9). The target for the Q-learning update is set based on the Bellman equation. If the episode terminates at the next state, the target is simply the immediate reward. Otherwise, it is the immediate reward plus the discounted maximum Q-value of the next state (lines 10-12). Gradient descent is then performed on the mean squared error between the predicted Q-values and the target values, with respect to the network parameters θ (line 13). This process of selecting actions, storing transitions, sampling minibatches, and updating the neural network parameters is repeated for each timestep within each episode, over the entire training duration (lines 4-14).

In the Gridworld DQN Algorithm, we change how we handle states compared to classic DQN. Instead of using the raw state s, we first preprocess it into $\phi(s)$ which is the tensors. Also, we define terminal states based on the rewards of Goal and Pit states. The final pseudocode for the Gridworld DQN 4 x 4:

In the gridworld scenario (Algorithm 2), the state space is represented as a sequence of symbols, denoted as s_t, and preprocessed into a feature vector $\phi(s_t)$. This preprocessing step is specific to the gridworld environment, allowing the neural network to operate on a more structured input format compared to the general DQN algorithm (line 6). The gridworld environment is explicitly initialized, denoted as $test_game$, with a fixed size of 4×4. This initialization step is specific to the gridworld environment, setting up the environment where the agent will interact (line 5). During action selection, the agent chooses actions based on the maximization of the action-value function Q with respect to the preprocessed state feature vector $\phi(s_t)$. This differs from the general DQN algorithm, where actions are chosen based on the state s_t directly (line 7). Transitions in the replay memory D are stored as tuples $(\phi(s_t), a_t, r_t, \phi(s_{t+1}))$. Here, the states are represented by preprocessed feature vectors $\phi(s_t)$ and $\phi(s_{t+1})$, instead of raw states s_t and s_{t+1}. This distinction is crucial for the gridworld environment, where states are represented as sequences of symbols (line 8). When computing the target value y_j for the Q-learning update, the next state ϕ_{j+1} is checked for terminality. If the next state is terminal, the target value is simply the immediate reward r_j. Otherwise, it is the immediate reward plus the discounted maximum Q-value of the next state, estimated using the target network $\hat{Q}$ (line 9). The termination conditions are specific to the gridworld environment. If the agent reaches a state with a reward of 10, it is considered a win condition. Similarly, if the agent receives a reward of -10 and has taken more than 15 steps, it is considered a loss condition. These conditions signify the end of the episode in the gridworld environment (lines 11-14).

*2) **Offline RL**:* also known as batch RL, is considered different from the traditional real-time interaction with the environment [26] [27]. Instead, it uses a fixed set of transitions obtained from the experience replay buffer [28], usually from a pre-trained or already trained model (π_β in Fig. 6), enabling supervised and cost-effective training. Typically, the input variables for Offline RL are the same as those for generic off-policy RL, which include the current state and action. The main difference is in the static dataset used for policy training. In Offline RL, the dataset is pre-collected, while off-policy RL relies on online interactions. Nevertheless, other policy

specifications, such as the testing phase, remain the same for both Offline RL and off-policy RL [2] [29].

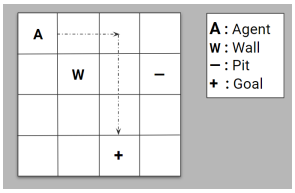
*3) **Evaluation Metrics**:* The evaluation metrics employed in this study during the Gridworld testing are as follows:

- **Testing accuracy** is the primary measure of accuracy in the context of reinforcement learning, which represents the percentage of games won out of all testing epochs/games played. Additionally, we evaluated the model's performance using two other key metrics.
- **Average Cumulative Reward (ACR)**: ACR measures the mean cumulative reward obtained by the agent across all testing games. Here, cumulative reward represents the sum of rewards acquired by the agent within a single game.
- **Average Epoch Length (AEL)**: This metric shows the average number of steps it took for the agent to reach the terminal state in each of the testing games. It provides insights into how efficiently the agent accomplishes its goals.

B. Gridworld Setup and Challenges

The Gridworld environment serves as a simplified research platform for reinforcement learning. In the constructed setup[1][code: *Gridworld Setup.ipynb*], an agent's objective is to navigate through a grid to reach the goal while avoiding pitfalls and walls (see Fig. 5a). Both environments in which the agent was tested used the same Gridworld setup.

1) **Gridworld setup:** The default Gridworld size in our setup is a 4×4 matrix, where each cell represents a position on the grid. In further evaluations, the Gridworld size will be extended up to 5×5 and 6×6 matrices respectively. Positions are divided into four categories: Player, Wall, Goal, and Pit. Each position is assigned a unique numeric identifier ranging from 0 to 15, corresponding to the 16 possible entries in the matrix (see Fig. 5b). For example, the game state can be encoded as an array of four elements such as $[0, 5, 14, 7]$ (as in Fig. 5a), representing the current positions of the player, wall, goal, and pit, respectively.



(a) Gridworld Objective.

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```

(b) Griworld Matrix Positions.

Fig. 5: Gridworld Representation.

Each learning cycle in the Gridworld environment is referred to as an **epoch**. Epoch starts from the **initial state**, randomly selected, and ends upon reaching a **terminal state**. The terminal state is reached when the agent encounters Goal or Pit positions, or when the agent exhausts the maximum number of allowed steps (*Max # of Steps)*.

2) **Challenges:** To tackle the Gridworld problem, a reinforcement learning method, Deep Q-Networks (DQN) that utilizes neural networks, was trained from scratch [21] [24]. The network takes the current state of the board as input and produces four Q-values as output, which correspond to a probability distribution of possible actions.

For an effective action selection strategy, an epsilon-greedy policy was used [24]. This policy includes choosing an epsilon value that balances exploration and exploitation in the action-choosing process. Random action is chosen with a probability equal to ϵ, which allows the agent to explore the environment and collect valuable information. Conversely, with probability $1 - \epsilon$, the action with the highest predicted Q value obtained using $argmax(Q_\theta(s, a))$ is selected for exploitation. Each step in the Gridworld brings a reward of -1, which encourages the agent to take the shortest route to the goal. Falling into a pit incurs a -10 penalty, resulting in the loss of the game. Conversely, successfully reaching the goal gives a reward of +10, leading to victory in the game.

3) **Gridworld envronments***:* An OpenAI Gym Gridworld environment was created using the *minigrid* package, whereas the custom Gridworld was meticulously crafted from the ground up. Both custom and OpenAI Gridworld environments employ class structures, ensuring organized and modular code design. They both allow for randomized positions for agents and objects, enhancing the variability and complexity of tasks. Furthermore, both environments offer rendering capabilities and permit adjustments to grid size, facilitating scalability in experiments.

The main difference between these two environments is that in the custom Gridworld environment, the class doesn't require external inputs, while in OpenAI Gym Mini Grid, the *gym.env* parameter is necessary for greater integration flexibility. Actions in the custom environment are manually initialized, allowing for fine-grained control, whereas OpenAI Gym Mini Grid simplifies this process using *gym.spaces*. The custom Gridworld lacks an observation space, directly printing the grid picture, while OpenAI Gym Mini Grid defines an observation space using *gym.spaces.box*. Custom Gridworld sets positions for agents and objects manually, requiring more extensive code for unique scenarios. In contrast, OpenAI Gym Mini Grid utilizes pre-built functions like *Wall(), Lava(), etc.*, for easier positioning. While both environments support rendering, the custom Gridworld primarily prints the grid picture, whereas OpenAI Gym Mini Grid offers multiple rendering modes, including a more advanced "human" mode.

IV. Methodology

A. Static Dataset Collection and Preprocessing

1) Dataset Collection: To facilitate Offline RL models, it is essential to obtain a static dataset [30]. In this section, we present the methodology used to create both uniformly distributed and non-uniformly distributed datasets for estimation and comparison purposes.

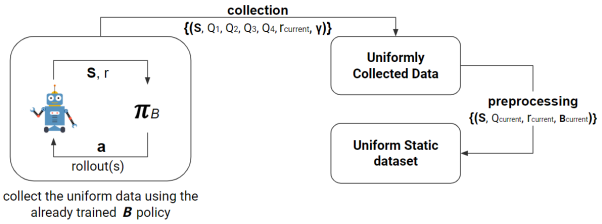
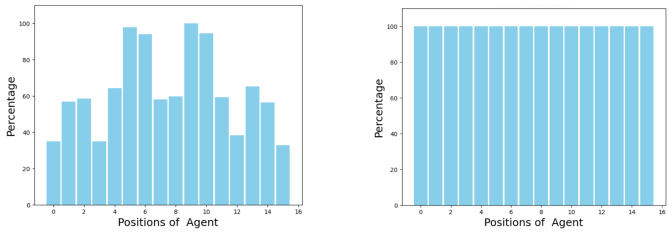


Fig. 6: Uniform static dataset acquisition.

A non-uniform dataset is characterized by an uneven distribution of states and actions across the state-action space (see Fig. 7a). In our case, the action selection is done by epsilon-greedy policy, with the value of epsilon set to 0.3. This means that in 30% of cases, the choice of actions is random (**exploration**) while in the remaining cases, the choice is based on the Q_{max} (**exploitation**). Additionally, the initial state of the agent in every epoch is chosen randomly. Because of the certain randomness in action selection and agents's initial state, certain states or actions are encountered more frequently than others.



(a) Non-Uniform Dataset (b) Uniform Dataset

Fig. 7: Representation of Dataset Distributions

This can pose significant challenges during training, as the agent might excessively emphasize the learning patterns associated with the more frequently occurring states or actions. Consequently, the agent's performance could be compromised in scenarios where states or actions are less frequently encountered. Notably, the non-uniform dataset is commonly gathered for Offline RL, where a fixed dataset is employed for training purposes.

On the other hand, a uniform dataset refers to a data distribution where states and actions are evenly distributed across the state-action space (see Fig. 7b). This implies that the agent encounters a balanced representation of various states and actions during the learning process. In our research approach, even distribution is achieved **by collecting all possible state-action pairs** in the space.

Both datasets, uniform and non-uniform, were generated using the same model and containing identical information columns. Firstly, the off-policy model was trained for 5000 epochs, which is the point where it was evident that the model's convergence occurred. This indicates that the training process reached a stable state, as the training error (loss) stopped decreasing.

Let's name the trained policy as B. After this, the uniform dataset[1] [code: *DataCollection.ipynb*] was first collected using the trained model. This indicates that the dataset was collected not during the training phase but by utilizing an already trained policy B to explore the Gridworld and collect data (see Fig. 6).

Overall, uniform static dataset collection involved a uniform exploration of the entire Gridworld environment using a trained off-policy RL model [10] [14] [31]. During exploration, game states were captured by observing the interactions between the environment and the agent. In this manner, the dataset was populated with the Q-values of all positions, excluding the walls (see Fig. 6). To assemble the uniform dataset, a systematic approach was developed to test and record every possible configuration of the Player, Wall, Goal, and Pit positions.

After collecting the uniform dataset, the non-uniform dataset[1] [data: *DatasetBeforeNonUniform.csv*] was collected by further training of the model with policy B. The data was collected directly from the experience buffer during training as in Fig. 1. Specifically, it was collected during the next 5000 epochs after the collection of the uniform dataset.

In **total**, the model with policy B was trained on 10000 epochs, the **convergence** was indicated on 5000 epochs. A collection of a **uniform** dataset was done by utilizing the trained model with no further training, and the collection of a **non-uniform** dataset was done during training from the 5000 epoch until the 10000 epoch.

Both collected datasets[1] [data: *DatasetBeforeUniform.csv, DatasetBeforeNonUniform.csv*] include important information such as the state of the environment, the rewards received during the exploration, and the corresponding Q values for the actions taken. Q-values are represented as [Q_UP, Q_DOWN, Q_LEFT, Q_RIGHT], providing insight into the agent's decision-making process as in Fig. 6 and Fig. 9a).

During the collection of static datasets from the DQN the applied reward function $R(s, a, s')$ can be defined as follows:

- If the agent moves from state s to state s' and falls into a pit, $R(s, a, s') = -10$.
- If the agent moves from state s to state s' and reaches the goal, $R(s, a, s') = +10$.
- If the agent moves from state s to state s' without falling into a pit or reaching the goal, $R(s, a, s') = -1$.

2) Dataset Preprocessing: is the process of transforming raw data into a format that can be used for training effective machine learning models [32] [33]. In our case, dataset preprocessing plays a critical role in preparing data for training an Offline RL model. In this section, we will discuss the specific steps involved in preprocessing[1] [code: *DataPreprocessing.ipynb*] of the dataset, including obtaining Q-values for the current state and computing the Bellman operator.

Obtaining Q value for the Current State: The dataset went through a preprocessing step to convert the Q-values from the next state (Q_next) to the Q-values for the current state (Q_current).

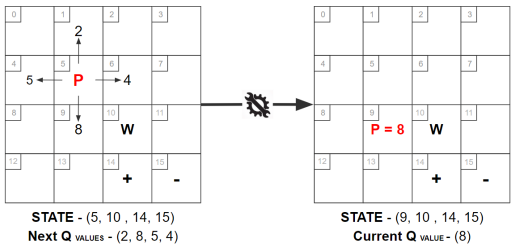


Fig. 8: Example of Dataset Preprocessing.

For example, if the Q_next values for a specific position $[5, 10, 14, 15](P, W, +, -)$ were equal to $[2, 8, 5, 4]$, corresponding to four possible next states based on the action taken, those values are used as the Q_current values for these four positions. Therefore, the value of Q_current at position $[9, 10, 14, 15]$, representing the state after moving DOWN from position $[5, 10, 14, 15]$, will be 8. This preprocessing step ensures that the dataset contains the appropriate Q-values for each state.

Obtaining Bellman Operator: The Bellman operator was manually calculated using the learning rate, the next Q value, and the reward values based on Equation 2. For example, let's consider the following Q-values for four possible next states from Fig. 11:

$$Q_\theta(s', d) = 8,\ Q_\theta(s', u) = 2,\ Q_\theta(s', l) = 5,\ Q_\theta(s', r) = 4 \tag{3}$$

In this scenario, the action that maximizes the Q values is the DOWN action (denoted as d) with a Q value of 8. Applying the Bellman operator equation, the value is computed as:

$$\beta(s) = -1 + 0.9 \cdot 8 = 6.2 \tag{4}$$

Here, the reward r is assigned a value of -1, assuming that the agent took a step without reaching the goal or falling into the pit. Furthermore, γ represents a constant discount factor set to 0.9. Since no further moves are possible after reaching the terminal state , the value of the Bellman operator for the goal position is set to its maximum value of 10, which corresponds to the maximum reward value (r = 10). Similarly, any positions that receive Bellman operator values beyond the value of the goal position are also set to 10, which means that 10 is the maximum value of the Bellman operator that can be assigned.

Player	Wall	Goal	Pit	Reward	Q_UP	Q_DOWN	Q_LEFT	Q_RIGHT

(a) Before Preprocessing.

Player	Wall	Goal	Pit	Reward	Current Q_value	Bellman Operator

(b) After Preprocessing.

Fig. 9: Dataset Preprocessing.

In this manner, both uniform[1] [data: *DatasetAfterUniform.csv*] and non-uniform[1] [data: *DatasetAfterNonUniform.csv*] datasets were received after preprocessing. The final datasets consist of the state information (Player, Wall, Goal, Pit), current Q value, Bellman Operator, and Reward information columns (see Fig. 9b).

B. Offline RL Algorithm

The goal in RL is to maximize long-term discounted reward in a Markov decision process (MDP), defined as a tuple (S, A, R, P, γ) [34], with State-space S, Action space A, Reward function $R(s, a)$, Transition dynamics $P(s_0|s, a)$, and Discount factor $\gamma \in [0, 1)$. The Q-function $Q^\pi(s, a)$ for a policy $\pi(a|s)$ is defined as the expected long-term discounted reward obtained by executing action a at state s and following $\pi(a|s)$ thereafter. Current practical Offline RL Q-learning methods (e.g., [21], [35], [36]) convert the Bellman equation into a bootstrapping-based objective for training a Q-network, Q_θ, via gradient descent. This objective, known as the mean-squared temporal difference (TD) error, is given by:

$$L(\theta) = \mathbb{E}_{s,a}\left[\left(R(s, a) + \gamma \overline{Q}_\theta(s', a') - Q_\theta(s, a)\right)^2\right],$$

where $\overline{Q}_\theta$ is a copy of the Q-function representing the *previous knowledge*. These methods train Q-networks via gradient descent and slowly update the target network by averaging its parameters. The typical abstract algorithm for such a Q method which uses deep Q learning is named fitted Q-iteration(**FQI**) [37] [29] (see Algorithm 3).

Algorithm 3 Fitted Q Iteration(FQI)

1: **Input:** a dataset from buffer $\mu = (s, a, r, s', a')$.
2: **Output:** Q-function $\boldsymbol{Q_\theta}$.
3: **for** fitting iteration k in $\{1, \ldots, N\}$ **do**
4: Compute target values for the experiences in a dataset:
5: $input : (s, a)$
6: Compute $\boldsymbol{Q_\theta(s, a)}$ on $\boldsymbol{(s, a)} \in \mu$
7: $target : y_k(\boldsymbol{s, a}) = r + \gamma \max_{a'} \boldsymbol{Q_{k-1}(s', a')}$ on $\boldsymbol{(r, s', a')} \in \mu$.
8: $loss : \mathcal{L} = (\boldsymbol{Q_\theta}(s, a) - y_k)^2$.
9: $backprop$: update weights $\boldsymbol{\theta}$ via $t = 1, \ldots, T$ gradient descent to minimize $\boldsymbol{\mathcal{L}}$
10: **end for**

In algorithm 3, the tuple (s, a, r, s, a') stored in the replay buffer is utilized as the static dataset upon which the FQI offline RL algorithm is trained (line 1). The algorithm undergoes training over N iterations, representing the length of the dataset μ, signifying that each tuple from the dataset is utilized in training (line 3). The primary objective of the algorithm is to train the Q function $\boldsymbol{Q_\theta}$ (line 2), which, given (s, a) as input (line 5), yields the Q value corresponding to the *next state* based on the action (line 6). The target network of past experiences, denoted as $\boldsymbol{Q_{k-1}}$ (line 7), is employed in the Bellman Operator equation (see. 2) to compute the Bellman Operator value of the current iteration (y_k). During the k-th fitting iteration, FQI trains the Q-function, $\boldsymbol{Q_\theta}$, by employing the previous experience algorithm $\boldsymbol{Q_{k-1}}$ (line 7). The $\boldsymbol{Q_{k-1}}$

serves as the delayed copy of $\boldsymbol{Q_\theta}$, preserving the weights (θ) from the previous iteration ($k-1$). The loss function is then employed to minimize the discrepancy between the Q value of the next state ($\boldsymbol{Q_\theta}(s,a)$) and the Bellman Operator value (y_k) (line 8). Finally, backpropagation is conducted to perform a gradient update of the Q function weights (θ) with a frequency of $t = 1, \ldots, T$, indicating that the weights are updated at each iteration (line 9).

Our methodology particularly uses not the previous experience Q function but rather the Q values collected from the Off-policy RL training itself. By including the Q values in the dataset we can control the data distribution of the dataset we collect. Moreover, by using the Q values in Offline RL the training performs much faster since we do not approximate the state-action on the past experience Q function but rather directly use the Q values (see Algorithm 4).

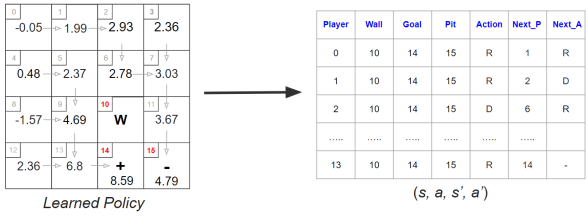
Algorithm 4 Offline RL using Bellman Operator equation

1: **Input:** a dataset $\mu = (s, r, \mathbf{Q}(\mathbf{s}', \mathbf{a_{max}}))$
2: **Output:** the Q-function $\boldsymbol{Q_\theta}$
3: **for** fitting iteration k in $\{1, \ldots, N\}$ **do**
4: $input : (s)$
5: Compute $\boldsymbol{Q_\theta(s)}$ on $(\boldsymbol{s}) \in \mu$
6: Compute target values for the experiences in a dataset:
7: $target : \ y_k = r + \gamma * \mathbf{Q}(\mathbf{s}', \mathbf{a_{max}})$.
8: $loss : \mathcal{L} = (Q_\theta(s) - y_k)^2$
9: $backprop$: update weights $\boldsymbol{\theta}$ using gradient descent to minimize $\boldsymbol{\mathcal{L}}$
10: **end for**

In algorithm 4, it can be observed that instead of utilizing the tuple (s, a, r, s', a'), it directly employs the tuple $(s, r, Q(s', a_{max}))$ (line 1). Here, $Q(s', a_{max})$ is acquired during Off-policy RL training. Hence, it can be stated that our Offline RL $\boldsymbol{Q_\theta}$ is trained by utilizing $\boldsymbol{Q_{DQN}}$ as a *previous experience* network (line 2). Similarly, algorithm 4, as in algorithm 3, training occurred over N iterations, corresponding to the static dataset size (line 3). However, the input was altered from (s, a) to (s) (line 4), enabling the Offline RL algorithm ($\boldsymbol{Q_\theta}$) to predict the Q value for the *current state* (line 5). This adjustment shifts the paradigm of the Q function $Q^\pi(s, a)$, where a denotes the action executed in state s. In our scenario, the current Q value is represented as $Q^\pi(s)$, while the subsequent Q value is denoted as $Q^\pi(s', a)$, where action a leads to the subsequent state s' (see equation 3) (line 5). The Bellman Operator equation was executed akin to the preprocessing stage (see equation 4) (line 7). The Bellman Operator value (y_k) mirrors the value stored in the after-preprocessing static dataset (see Fig. 9b) (line 7). The loss function and backpropagation were similarly performed in our algorithm, as in algorithm 3 (line 8, 9).

The Q value approximation algorithm is similar to the Belaman Operator(Algorithm 4), just as the target value y_k we use the current Q value ($Q(s)$), (see Fig. 9b). Overall we make assumptions that the discount factor is constantly equal to 1 and the reward of all states equal to 0.

(a) From collected dataset to Learned Policy.



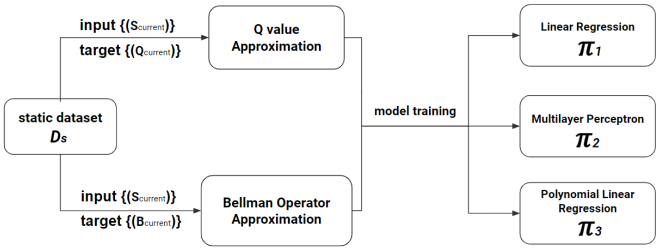
(b) From Learned Policy to Classic RL dataset.

Fig. 10: Dataset Bridge.

The dataset that was collected for the Offline RL $(s, r, Q_1, Q_2, Q_3, Q_4)$ can be easily bridged with the class Offline RL dataset as can be seen in the Fig. 10. It can be done by obtaining the *Learned Policy* and then deriving from them classical dataset (s, a, r, s', a'). In Fig. 10 the reward(r) is not included because it is the same in both datasets.

C. Offline model training

The training process of model[1] [code: *ModelPerformanceAndResults.ipynb*] included a separate approximation of the Q value and Bellman Operator value functions. The target value for training was the Q-value in the first case, and the Bellman operator in the second case (see Fig. 11). To train the models, the input values included information about the current state, which included Player, Wall, Goal, and Pit positions. The goal was to input information about the current state and get the corresponding current value of the Q-value or the value of the Bellman operator as output. The training process used the entire dataset without splitting it into separate datasets for training and testing.



n

Fig. 11: Offline model training.

Three different approximation models were used for training: Linear Regression (LR), Multilayer Perceptron (MLP) Regressor, and Polynomial Linear Regression (Poly LR).

The selection of these models was based on their respective characteristics. LR was chosen as a fundamental and easily interpretable regression model. On the other hand, MLP was chosen to mimic the Deep Q-Network (DQN) model, which has shown successful results in reinforcement learning. Poly LR was chosen as an upgraded version of LR which can show improved performance and give better results.

D. Gridworld setup testing

To evaluate the performance of the proposed framework, a comprehensive evaluation of the trained offline models was conducted in the Gridworld setup using the metrics mentioned in Section III-A3. This evaluation included integrating the offline model policy into the experience buffer to make it available during the testing phase (Fig 12). The testing procedure involved initiating 10,000 epochs/games of the Gridworld, where the positions of the Player, Wall, Goal, and Pit changed each game.

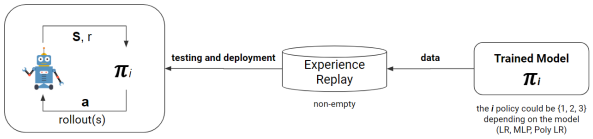


Fig. 12: Gridworld setup testing.

During the testing phase, the decision-making process was delegated to the Offline RL model. This was achieved by evaluating the four Q values or Bellman operators corresponding to the nearest positions of the Player and selecting the action with the highest predicted value. Outside the Gridworld boundaries, position values have been automatically set to -100. To assess the model's performance, we calculated several key metrics based on the outcomes of 10,000 epochs.

V. Results

A. Experimental setup

The experimental setup of this research work consisted of several components:

1) ***Deep Q-Network (DQN):*** First, DQN training for the grid used the Adam optimizer and employed two hidden layers in the neural network. The training process continued until the model converged, which happened after 5000 episodes. The resulting DQN model achieved an impressive accuracy of 92.8% during testing on the 4x4 Gridworld setup, 87% on the 5x5, and 81.2% on the 6x6 Gridworld setups respectively. The parameters used to train those Gridworld setups can be found in Table I.

Parameter tuning was performed on the 5x5 and 6x6 Gridworlds. For details on how it was performed, see Appendix VI. Parameter tuning was not done for the 4x4 Gridworld because it already achieved the optimal accuracy of 80% with the baseline parameters. This optimal accuracy ensures that the collected data is rich enough. Thus, parameter tuning was performed only for the 5x5 and 6x6 setups, and the resulting parameters, achieving more than 80% accuracy, are listed in Table I.

TABLE I: Gridworld Training Parameters

Gridworld Size	Epsilon	# of neurons	Buffer Size	Batch Size	Max # of Steps
4 x 4	0.3	(200, 150)	1000	200	50
5 x 5	0.3	(600, 400)	2000	500	300
6 x 6	0.5	(1000, 500)	4000	800	600

Note: Term **#** represents number.

2) ***Offline RL models:*** The LR model was trained with default parameters, without any tuning. On the other hand, the MLP model was trained to reproduce the parameters of the DQN model. It used the Adam optimizer and had hidden layers of sizes 150 and 200, similar to the DQN model. For Poly LR, the *Model Selection* approach was used to determine the best degree among the range of 1 to 10. Testing accuracy values served as evaluation metrics for both uniform and non-uniform datasets, assessing problems from both RL and regression perspectives. After analyzing the results, a degree of **4** was chosen as the optimal option. It demonstrated the highest performance when approximating the Bellman operator and relatively high results when approximating the Q value. The same Polynomial LR degree (4) was consistently applied to both the 5 x 5 and 6 x 6 Gridworld setups.

B. Data Collection and Testing Process

The process of data collection starts from the Off-Policy Model training[1] [code: *DataCollection.ipynb*] and Uniform Dataset Collection [1] [code: *DataCollection.ipynb*], then the collected data goes through the Preprocessing stage [1] [code: *DataPreprocessing.ipynb*], then the proprocessed data is trained on Offline RL models [1] [code: *ModelPerformanceAndResults.ipynb*] (see Fig. 13).

1) ***Off-Policy Model Training:*** starts with initializing the Gridworld game, which can have sizes 4 x 4, 5 x 5, or 6 x 6 (line 1). For example, let's consider the size 4 x 4. The game is then rendered into tensors with a size of 64, derived by multiplying the length of the grid (16) by the number of unique positions (4) (lines 2-3). Next, the states are used to predict the Q-values of the current state (Q_1) and the next state (Q_2) (lines 5-7). Subsequently, the Bellman Operator equation is applied using the Q-values of the next state (line 9). Then, a loss function is computed between the Bellman Operator value and the Q-values of the current state (lines 10-11). Finally, backpropagation is performed on the obtained loss value, updating the weights of the *model* using gradient descent to minimize the loss (lines 13-14).

2) ***Unifrom Dataset Collection:*** begins by spanning all possible states of the Gridworld (lines 1-9). Next, it calculates Q-values for each state using the Off-policy RL model (lines 11-13). Throughout these operations, both the state and Q-values are stored in a matrix, which is then saved as a dataset in a CSV file (lines 15-16). The non-uniform dataset was collected using a similar approach, albeit with randomly selected states instead of covering all possible states.

3) ***Preprocessing:*** begins immediately after the dataset collection step. In this stage, the collected dataset undergoes

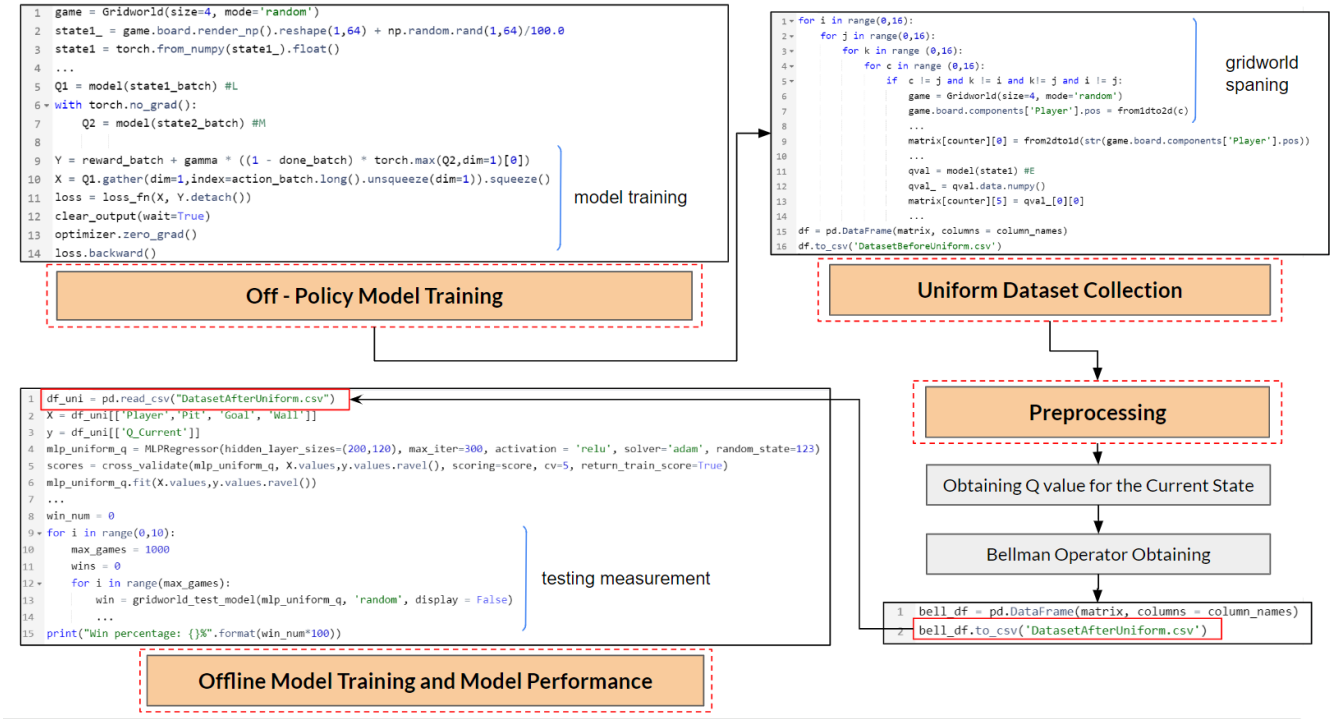


Fig. 13: Data Collection and Tessting Process.

preprocessing to obtain the Q-value for the current state and the Bellman Operator value for the same state (refer to Section IV-A). The output of this stage is the preprocessed dataset which is stored in a CSV file (lines 1-2).

*4) **Offline Model Training and Model Performance**:* starts by segmenting the dataset into dependent (Y) and independent (X) variables (lines 1-3). Subsequently, our Offline RL model (MLP as illustrated in Fig. 13) undergoes training (lines 4-6). Following this, the testing phase begins, wherein a specified number of episodes are conducted, employing our Offline RL model to predict the Q values (lines 8-13). Ultimately, the metric utilized in Fig. 13 is the testing accuracy, which represents the cumulative win percentage over the given number of episodes (line 15).

C. Experimental Results

To assemble a **uniform static dataset**, the positions of the Player, Wall, Goal, and Pit were systematically changed at each step taken by the agent, covering the entire Gridworld. This ensured an even distribution of data points. The uniform dataset size was determined to be *50,400 samples*, which can be calculated by considering all possible states of the Gridworld: $16(\text{Goal}) * 15(\text{Pit}) * 14(\text{Wall}) * 15(\text{Player}) = 50,400$. It should be noted that the Player can occupy Goal and Pit positions, which leads to 15 possible Player positions. Additionally, a **non-uniform dataset** was generated, totaling *23,000 data points*, calculated by multiplying 5,000 epochs with an AEL of approximately 4.6. However, only *8,800 samples* within this dataset are unique, constituting roughly 17% of the total data points in the uniform dataset. Despite increasing the number of epochs for collecting the non-uniform dataset, the count of repeated datapoints remains constant, resulting in a limited number of unique data points.

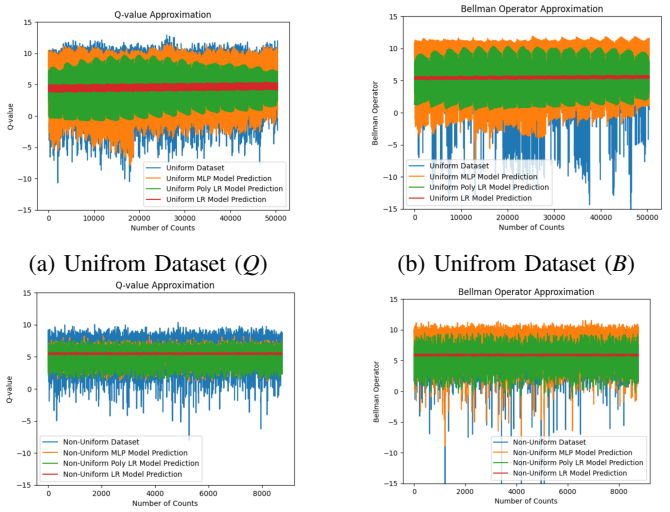
The MLP algorithm applied to the uniform dataset showed remarkable performance, achieving a Q-value approximation of 92% and a Bellman operator approximation of 88% on the 4 x 4 Gridworld setup (see Table II). MLP outperformed other models in the approximation of the Bellman operator trained on the non-uniform dataset, achieving a testing accuracy of 75.5% (see Table II). On the other hand, the Poly LR(4) model achieved the highest testing accuracy of 58.57% when approximating the Q-values trained on the non-uniform dataset (see Table II). On the other hand, the LR algorithm gave an accuracy of 20% for approximating both the Q value and the Bellman operator (see Table II). Although this level of accuracy is comparatively lower than that of the MLP algorithm, it still provides valuable information about the performance of the LR algorithm and its potential applications in a research context.

Fig. 14 gives a visual representation of how each algorithm approximated Q-values and Bellman operator values. First, it is obvious that the LR algorithm approximated all values close to the mean, but it resulted in low accuracy of the results. This indicates that LR tends to generalize values, potentially sacrificing precision in the process [38]. Second, the MLP algorithm consistently reproduces actual values more accurately than other algorithms in most cases. Its performance excels, demonstrating its ability to capture the underlying patterns and nuances of data. However, it is worth noting that

TABLE II: Model Performance 4 x 4 Custom Gridworld

Metric	MLP U.	MLP Non-U.	Poly LR(4) U.	Poly LR(4) Non - U.	LR U.	LR Non-U.
Test Acc. (Q)	**92.4**%	44.7%	48.72%	**58.57**%	20.2%	19.95%
Test Acc. (B)	**88.3**%	**75.5**%	66.02%	64.67%	19.3%	19.4%
ACR(Q)	**6.79**	-4.86	-4.12	**-1.908**	-9.27	-10.1
ACR(B)	**6.01**	3.17	1.23	**0.61**	-10.7	-10.65
AEL(Q)	**3.53**	9.1	8.15	**6.33**	10.4	10.36
AEL(B)	**2.88**	4.5	5.15	**5.74**	10.46	10.5

Note: The terms used in Table II are defined as follows: Term [**U.**] represents a Uniform Dataset, Term [**Non - U.**] represents a Non-Uniform Dataset, Term [**Q**] represents Q-value Approximation, and Term [**B**] represents Bellman Operator Approximation.



(a) Unifrom Dataset (Q) (b) Unifrom Dataset (B)

(c) Non - Unifrom Dataset (Q) (d) Non - Unifrom Dataset (B)

Fig. 14: Comparison of models on Custom 4 x 4 Setup.

Note: The terms used in Figure 14 are defined in the Table II

there is an exception during the approximation of the Q-value on the non-uniform dataset where the Poly LR algorithm replicates the actual values better (see Fig. 14c). This highlights the importance of considering specific characteristics such as sparsity and distribution of the dataset when choosing the most appropriate algorithm. Overall, there is a clear pattern in the figure, indicating that all algorithms tend to compress the actual data toward the mean. This observation indicates that models tend to generalize values, potentially missing important variations and specific features in the data.

Table III presents a comprehensive analysis of the best-performing algorithms across various Gridworld setups, environments, and datasets. Upon initial inspection, it becomes evident that there are no significant disparities between the Custom and Open AI Gym Gridworld environments.

When the impact of increasing the size of the Gridworld setup was examined, a noticeable trend emerged. The testing accuracy values exhibit a decline as the Gridworld dimensions expand. However, it is essential to clarify that this decrease in performance for larger dimensions does not imply that the methodology is inherently ineffective for higher-dimensional scenarios. Rather, it reflects the practical constraints imposed

TABLE III: Performance on Different Setups and Environments

(a) Uniform Dataset

Setup	Environment	Algorithm	Test Acc. (Q)	Test Acc. (B)
4 x 4	Custom	MLP U.	92.4%	88.3%
5 x 5	Custom	MLP U.	85.6%	80.1%
6 x 6	Custom	MLP U.	80.3%	77.5%
4 x 4	Open AI Gym	MLP U.	92.3%	88.3%
5 x 5	Open AI Gym	MLP U.	85.2%	80.7%
6 x 6	Open AI Gym	MLP U.	80.4%	76.9%

(b) Non-Uniform Dataset

Setup	Environment	Algorithm	Test Acc. (Q)	Test Acc. (B)
4 x 4	Custom	MLP Non U.	44.7%	75.5%
5 x 5	Custom	MLP Non U.	40.6%	71.1%
6 x 6	Custom	Poly LR(4) Non U.	41.3%	68.5%
4 x 4	Open AI Gym	MLP Non U.	44.7%	75.4%
5 x 5	Open AI Gym	MLP Non U.	40.5%	72.0%
6 x 6	Open AI Gym	Poly LR(4) Non U.	41.1%	68.9%

Note: The terms used in Figure III are defined in the Table II

by limited computational resources, which restricted the extent of hyperparameter tuning for these larger setups.

TABLE IV: Final Results

Size	Offline RL model: Uniform Test Acc	Offline RL model: Non Uniform Test Acc	Off-policy RL model: DQN Test Acc.
4 x 4	92.4%	44.7%	92.8%
5 x 5	85.6%	40.6%	87.0%
6 x 6	80.4%	41.3%	81.2%

In summary, across all setups with uniform datasets, the MLP algorithm consistently demonstrates superior performance. In contrast, when dealing with non-uniform datasets, the choice of the best algorithm becomes less straightforward. Here, the results suggest that MLP excels in Bellman operator approximation, while Poly LR(4) exhibits superior performance in Q-value approximations. As a result, it was concluded that MLP is the preferred algorithm for 4 x 4 and 5 x 5 setups, while Poly LR(4) emerges as the more effective choice for the 6 x 6 setup (see Table III).

VI. Conclusion

In summary, this study offers a comprehensive guide to Offline RL, highlighting a successful transfer of algorithms from off-policy to Offline RL, with the MLP model proving particularly effective, showcasing only a marginal 1% reduction in testing accuracy compared to the original DQN model (see Table IV). Notably, results differed significantly between models trained on uniform and non-uniform datasets, highlighting the need for further exploration of the Bellman operator and Q-value approximations.

However, it is crucial to recognize limitations and areas for future research, including the scalability of the given framework that needs to be tested in diverse environments beyond the Gridworld setup and the necessity for additional metrics reflecting real-world performance. Moreover, research efforts should focus on techniques that will reduce dimensionality while keeping the uniformness of datasets. Nonetheless, this study establishes a solid foundation for the integration of Offline RL in practical scenarios, such as gaming, autonomous driving, and robotics within the Gridworld framework.

Acknowledgment: The original work is supported by Nazarbayev University (NU), Kazakhstan, under a School of Engineering and Digital Sciences (SEDS) grant.

Appendix

This section contains hyperparameter tuning tables for the 5 x 5 and 6 x 6 Gridworld environments for the DQN algorithm (see Table V & VI). For the 4 x 4 Gridworld, default hyperparameter settings.

In the tables V and VI, ϵ represents the epsilon value used in epsilon-greedy policy, γ stands for the discount factor, α denotes the learning rate, and "# of Neurons" indicates the number of hidden neurons.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] S. Levine *et al.*, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[3] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International conference on machine learning*. PMLR, 2019, pp. 2052–2062.

[4] R.-J. Qin, X. Zhang, S. Gao, X.-H. Chen, Z. Li, W. Zhang, and Y. Yu, "Neorl: A near real-world benchmark for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 753–24 765, 2022.

[5] C. Lu, P. J. Ball, J. Parker-Holder, M. A. Osborne, and S. J. Roberts, "Revisiting design choices in offline model-based reinforcement learning," *arXiv preprint arXiv:2110.04135*, 2021.

[6] Y. Jin, Z. Yang, and Z. Wang, "Is pessimism provably efficient for offline rl?" in *International Conference on Machine Learning*. PMLR, 2021, pp. 5084–5096.

[7] L. Wang, Q. Cai, Z. Yang, and Z. Wang, "Neural policy gradient methods: Global optimality and rates of convergence," *arXiv preprint arXiv:1909.01150*, 2019.

[8] D. Yarats, D. Brandfonbrener, H. Liu, M. Laskin, P. Abbeel, A. Lazaric, and L. Pinto, "Don't change the algorithm, change the data: Exploratory data for offline reinforcement learning," *arXiv preprint arXiv:2201.13425*, 2022.

[9] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," *Advances in neural information processing systems*, vol. 34, pp. 20 132–20 145, 2021.

[10] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1179–1191, 2020.

[11] I. Kostrikov, R. Fergus, J. Tompson, and O. Nachum, "Offline reinforcement learning with fisher divergence critic regularization," in *International Conference on Machine Learning*. PMLR, 2021, pp. 5774–5783.

[12] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, and T. Ma, "Mopo: Model-based offline policy optimization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 14 129–14 142, 2020.

[13] R. Kidambi, A. Rajeswaran, P. Netrapalli, and T. Joachims, "Morel: Model-based offline reinforcement learning," *Advances in neural information processing systems*, vol. 33, pp. 21 810–21 823, 2020.

[14] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," *arXiv preprint arXiv:2004.07219*, 2020.

[15] M. Mathieu, S. Ozair, S. Srinivasan, C. Gulcehre, S. Zhang, R. Jiang, T. Le Paine, K. Zolna, R. Powell, J. Schrittwieser *et al.*, "Starcraft ii unplugged: Large scale offline reinforcement learning," in *Deep RL Workshop NeurIPS 2021*, 2021.

[16] C. Gulcehre, Z. Wang, A. Novikov, T. Paine, S. Gómez, K. Zolna, R. Agarwal, J. S. Merel, D. J. Mankowitz, C. Paduraru *et al.*, "Rl unplugged: A suite of benchmarks for offline reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 7248–7259, 2020.

[17] R. Laroche, P. Trichelair, and R. T. Des Combes, "Safe policy improvement with baseline bootstrapping," in *International conference on machine learning*. PMLR, 2019, pp. 3652–3661.

[18] D. Hein, S. Udluft, M. Tokic, A. Hentschel, T. A. Runkler, and V. Sterzing, "Batch reinforcement learning on the industrial benchmark: First experiences," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 4214–4221.

[19] R. Agarwal, D. Schuurmans, and M. Norouzi, "An optimistic perspective on offline reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 104–114.

[20] N. Lambert, M. Wulfmeier, W. Whitney, A. Byravan, M. Bloesch, V. Dasagi, T. Hertweck, and M. Riedmiller, "The challenges of exploration for offline reinforcement learning," *arXiv preprint arXiv:2201.11861*, 2022.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[22] J.-G. Park, "Offline rl oriented functions design for dynamic power management on cnn workloads," in *2023 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, 2023, pp. 324–325.

[23] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[25] J. Tsitsiklis and B. Van Roy, "Analysis of temporal-diffference learning with function approximation," *Advances in neural information processing systems*, vol. 9, 1996.

[26] S. Lange, T. Gabel, and M. Riedmiller, "Batch reinforcement learning," *Reinforcement learning: State-of-the-art*, pp. 45–73, 2012.

[27] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, pp. 293–321, 1992.

[28] T. Schaul *et al.*, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[29] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, 2005.

[30] L. Monier, J. Kmec, A. Laterre, T. Pierrot, V. Courgeau, O. Sigaud, and K. Beguir, "Offline reinforcement learning hands-on," *arXiv preprint arXiv:2011.14379*, 2020.

[31] C. Gulcehre, S. G. Colmenarejo, Z. Wang, J. Sygnowski, T. Paine, K. Zolna, Y. Chen, M. Hoffman, R. Pascanu, and N. de Freitas, "Reg-

ularized behavior value estimation," *arXiv preprint arXiv:2103.09575*, 2021.

[32] B. Bilalli, A. Abelló, T. Aluja-Banet, and R. Wrembel, "Automated data pre-processing via meta-learning," in *International Conference on Model and Data Engineering*. Springer, 2016, pp. 194–208.

[33] M. A. Munson, "A study on the importance of and time spent on different modeling steps," *ACM SIGKDD Explorations Newsletter*, vol. 13, no. 2, pp. 65–71, 2012.

[34] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[35] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[36] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[37] A. Kumar, R. Agarwal, D. Ghosh, and S. Levine, "Implicit under-parameterization inhibits data-efficient deep reinforcement learning," *arXiv preprint arXiv:2010.14498*, 2020.

[38] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

TABLE V: Hyperparameter Tuning for 5 x 5 Gridworld

ϵ	γ	α	# of Neurons	Buffer Size	Batch Size	Max Moves	Episodes	Accuracy (%)
0.3	0.9	0.001	(200,160)	1000	200	50	1000	32.6
0.3	0.9	0.001	(200,160)	1000	200	200	1000	53.5
0.3	0.9	0.001	(200,160)	1000	200	300	1000	56.4
0.3	0.9	0.001	(400,320)	1000	200	300	1000	61.3
0.3	0.9	0.001	(400,320)	2000	400	300	1000	68.6
0.3	0.9	0.001	(400,320)	2000	400	300	5000	86.6
0.3	**0.9**	**0.001**	**(600,400)**	**2000**	**500**	**300**	**5000**	**87.0**

TABLE VI: Hyperparameter Tuning for 6 x 6 Gridworld

ϵ	γ	α	# of Neurons	Buffer Size	Batch Size	Max Moves	Episodes	Accuracy (%)
0.3	0.9	0.001	(600,400)	2000	500	300	1000	35.4
0.3	0.9	0.001	(800,400)	4000	500	600	1000	40.5
0.3	0.9	0.001	(1000,500)	4000	800	600	1000	41.6
0.4	0.9	0.001	(1000,500)	4000	800	600	1000	46.4
0.5	0.9	0.001	(1000,500)	4000	800	600	1000	54.2
0.5	**0.9**	**0.001**	**(1000,500)**	**4000**	**800**	**600**	**5000**	**81.2**