SimuDICE: Offline Policy Optimization Through Iterative World Model Updates and DICE Estimation

Cătălin-Emanuel Brița[1]
Supervisor(s): Frans Oliehoek[1]**, Stephan Bongers**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Cătălin-Emanuel Brița
Final project course: CSE3000 Research Project
Thesis committee: Frans Oliehoek, Stephan Bongers, Catholijn Jonker

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

In offline reinforcement learning, deriving a policy from a pre-collected set of experiences is challenging due to the limited sample size and the mismatched state-action distribution between the *target policy* and the *behavioral policy* that generated the data. Learning a dynamic model of the environment can improve the sample efficiency of the algorithm, but this mismatch can lead to the generation of suboptimal experiences. We propose SimuDICE, an algorithm that enhances the sampling of imaginary experiences using Dual stationary DIstribution Correction (DICE), and iteratively improves the DICE estimations with synthetically generated experiences. SimuDICE addresses the *objective mismatch* issue by iteratively updating both the world model and the DICE estimator, aligning the model's training objective (imitating the environment) with its usage objective (policy improvement). We show that SimuDICE requires less pre-collected data and fewer simulated experiences to achieve comparable results to other algorithms while having greater robustness to lower data quality.

1 Introduction

Reinforcement Learning (RL) [1] has recently shown great success in various domains such as games [2], robotics [3], and conversational systems [4], largely due to simulation-based trial and error [5, 6]. While feasible in environments with simulators, deploying policies in some environments can be risky or costly. Offline RL [7, 8] addresses this challenge by training agents from static pre-collected datasets. However, the amount of data is limited and collected under different unknown policies, known as *behavior policies*, making direct transitions from online to offline settings problematic [9, 10]. The mismatch in the state-action distribution can lead to divergence in off-policy learning [11, 12].

Model-based RL (MBRL) improves the sample efficiency and stability of policy optimization in the offline setting by learning a dynamic model of the environment to generate imaginary rollouts. This approach has shown success in online RL through different world models such as Latent Dynamics Models [13, 14, 2] or Diffusion Models [15, 16]. However, building a perfect world model is often unfeasible, leading to hallucinations even for the most simple environments [17], significantly decreasing the performance of the policy. This issue is amplified by the offline setting, where the learned model can hardly generalize due to the limited amount of data and the complexity of the task [18]. To mitigate this, several works use prioritized experience replay for policy learning [19, 20] and for action sampling in world models [21, 22]. This prioritization usually accounts for model confidence or the state-action distribution mismatch between the *target policy* and the *behavioral policy*.

Over the years, various approaches have been developed to mitigate the policy-induced state-action distribution mismatch. Precup et al. [23] tackle the problem using products of importance sampling ratios, though this approach suffers from a large variance. To correct the distribution mismatch without incurring a large variance, Hallak et al. [24] and Liu et al. [25] propose learning the density ratio between the state distribution of the target policy and the sampling distribution directly. DualDICE [26] is a relaxation of previous methods, enabling learning from multiple unknown behavior policies via a change of variable technique. GenDICE [27] generalizes DualDICE, stabilizing estimation in the average reward setting. GradientDICE [28] outlines that GenDICE is not a convex-concave saddle-point problem in all settings and proposes a new, provably convergent method under linear function approximation. Despite their differences, all these algorithms use minimax optimizations, allowing them to be combined under the regularized Lagrangians of the same linear problem [29].

Most of the prior work in offline MBRL (e.g., [30, 31]) pre-train a one-step forward model via maximum likelihood estimation to be a simple mimic of the world and then uses it to improve the policy, without further improving the dynamic model. This results in an *objective mismatch*, namely the objective function used for model training (accurately predicting the environment) is unrelated to its utilization (policy optimization). Recent works have identified *objective mismatch* in the model training and utilization as problematic [32, 33].

In this work, we introduce SimuDICE, an algorithm designed to learn policies from offline data by iteratively updating both the world model and the DICE estimation using data generated by the other component. Specifically, the world model generates samples more likely to be encountered by the *target policy* using the DICE estimation, while the DICE estimation is improved using these simulated experiences. We have extended the framework proposed in Dyna-Q [34] that uses a Tabular World Model [35] to the offline setting and used DualDICE [26] for the behavior-agnostic state-action distribution mismatch estimation. Our experiments show that SimuDICE requires less offline data to converge to the *target policy* compared to similar algorithms and shows greater robustness to lower data quality. The algorithm is publicly available on GitHub [1].

2 Background

In this section, we outline the Problem setting 2.1 in which this work lies. We also discuss some prerequisite concepts and prior work necessary for understanding this paper, specifically the Online Dyna-Q algorithm 2.2 and DualDICE estimation 2.3.

2.1 Problem setting

Setting: We consider a Markov Decision Process (MDP) [36], in which the environment is defined by a tuple $\mathcal{M} = \langle S, A, R, T, \mu_0, \gamma \rangle$ where S represents the state space, A is the action space, R is a reward function, T is the transition probability function, μ_0 is the initial state distribution, and $\gamma \in [0, 1)$ is the discount factor.

Policy definition: A policy π in an MDP decides what action the agent should take given some state s. Formally, it is a mapping $\pi : S \to \Delta(A)$, where $\pi(s)$ represents the probability distribution over actions A in state s. Figure 1 shows how a policy interacts with an environment over n steps.

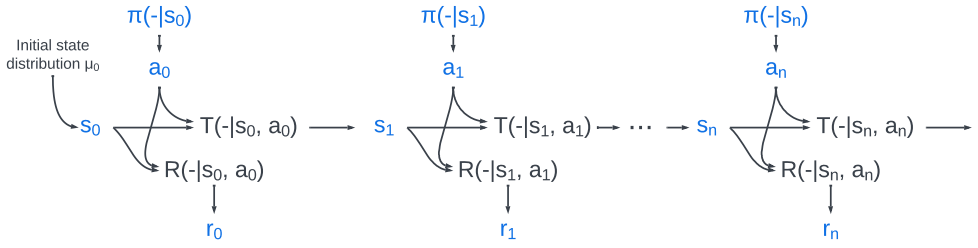

Figure 1: The policy starts at $s_0 \sim \mu_0$ and samples an action $a_t \sim \pi(s_t)$ at each step t from the policy. This action is applied to the environment, resulting in a reward $r_t \sim R(s_t, a_t)$ and the environment transitions to a new state $s_{t+1} \sim T(s_t, a_t)$.

[1] https://github.com/Catalin-2002/SimuDICE

Goal of reinforcement learning (RL): The goal of the agent is to maximize the cumulative expected reward (its return), given by Eq. (1).

$$\rho(\pi) = \mathbb{E}_{s_0 \sim \mu_0} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid a_t \sim \pi(\cdot \mid s_t) \right] \tag{1}$$

To evaluate the performance of a policy, we define two functions: the Value function $V^\pi(s)$, which is the expected return of policy π from state s (Eq. 2), and the Q-value function $Q^\pi(s, a)$, which represents the expected return following a policy π starting from state s with action a (Eq. 3).

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s \right] \tag{2}$$

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_0 = s, a_0 = a \right] \tag{3}$$

Bellman equation: The Bellman equation provides a recursive definition of the Q-value by decomposing it into immediate reward and the discounted value of the next state-action pair. Eq. (4) shows how the Bellman equation is applied to the Q-value policy function $Q^\pi(s, a)$.

$$Q^\pi(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim T(s,a), a' \sim \pi(\cdot|s')} \left[Q^\pi(s', a') \right] \tag{4}$$

The Bellman operator $\mathcal{B}_\pi$ iteratively applies the Bellman equation to update the Q-values until convergence, leading to a formulation as in Eq. (5).

$$\mathcal{B}^\pi Q(s, a) = R(s, a) + \gamma \mathbb{E}_{s' \sim T(s,a), a' \sim \pi(\cdot|s')} [Q(s', a')] \tag{5}$$

Offline RL: The focus of this work is offline RL. Unlike online RL, where the agent actively interacts with the environment to gather data and update its policy, offline RL aims to derive the optimal policy π from a fixed dataset of experiences. Specifically, we assume access to a finite dataset $\mathcal{D} = \left\{ \left(s_0^{(i)}, s^{(i)}, a^{(i)}, r^{(i)}, s'^{(i)} \right) \right\}_{i=1}^{N}$, where $s_0 \sim \mu_0$, $(s^{(i)}, a^{(i)}) \sim d^{\mathcal{D}}$ are samples from an unknown distribution $d^{\mathcal{D}}$, $r^{(i)} \sim R(s^{(i)}, a^{(i)})$, and $s'^{(i)} \sim T(s^{(i)}, a^{(i)})$.

2.2 Online Dyna-Q

Dyna-Q [34] is a classic RL approach that integrates model-free and model-based techniques. It uses simulated experiences generated by a model of the environment to optimize the Q-values. Dyna-Q integrates planning, acting, and learning in the following ways:

1. **Learning the world model:** the agent learns a model $\hat{T}$ that predicts the next state s' and the reward r given the current state s and an action a: $\hat{T}(s, a) = (s', r)$. Given the focus on deterministic tabular environments, a simple Tabular Model [35] is used.

2. **Direct RL Updates:** the agent directly interacts with the environment to collect experiences of the form (s, a, r, s'), and uses them to update the Q-values using the Q-learning formula: $Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right]$, where α is the learning rate and γ the discount factor.

3. **Planning updates:** The agent uses $\hat{T}$ to sample new experiences (s, a), predict the reward received from acting and the resulting new state (r, s'), and update the Q-values using the same Q-learning rule.

The pseudocode of Dyna-Q can be found in Algorithm 1. Steps $4-5$ show the acting part, step 6 shows the direct RL updates, while steps $7-13$ are the planning steps. Note that by removing steps $7-13$ the remaining algorithm is one-step tabular Q-learning [37].

Algorithm 1 Dyna-Q - adapted from *Reinforcement Learning: An Introduction* [1]

Initialize $Q(s,a)$ and $Model(s,a)$ for all $s \in S$ and $a \in A(s)$
while True **do**
  $S \leftarrow$ current (nonterminal) state; $A \leftarrow \epsilon$-greedy(S,Q)
  Execute action A; observe resultant reward, R, and state, S'
  $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
  $\hat{T}(S,A) \leftarrow R, S'$ ▷ assuming deterministic environment
  for $i = 1$ to n **do**
    $S, A \leftarrow$ random previously observed state and action
    $R, S' \leftarrow \hat{T}(S,A)$
    $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)]$
  end for
end while

2.3 DualDICE estimation

In this part, we will elaborate on DualDICE estimation [26], an algorithm for off-policy evaluation. They obtained impressive results by reducing the off-policy estimation problem to a density ratio estimation problem and doing a change of variable optimization trick. The value of the policy can be rewritten using the importance weighing trick as in Eq. (6).

$$\rho(\pi) = \mathbb{E}_{(s,a)\sim d^\pi}[r(s,a)] = \mathbb{E}_{(s,a)\sim d^\mathcal{D}}\left[\frac{d^\pi(s,a)}{d^\mathcal{D}(s,a)} r(s,a)\right], \tag{6}$$

where d^π is the discounted state visitation distribution, and expressed as in Eq. (7).

$$d^\pi(s,a) := (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \cdot \Pr[s_t = s, a_t = a \mid s_0 \sim \mu_0, \pi] \tag{7}$$

The equation can be rewritten in the offline setting as a weighted average (Eq. 8), reducing the problem to estimating the density ratios (Eq. 9) for policy correction.

$$\mathbb{E}_{(s,a)\sim d^\mathcal{D}}\left[\frac{d^\pi(s,a)}{d^\mathcal{D}(s,a)} r(s,a)\right] = \frac{1}{N}\sum_{i=1}^{N} \frac{d^\pi(s^{(i)},a^{(i)})}{d^\mathcal{D}(s^{(i)},a^{(i)})} r^{(i)} \tag{8}$$

$$w_{\pi/\mathcal{D}}(s,a) := \frac{d^\pi(s,a)}{d^\mathcal{D}(s,a)} \tag{9}$$

Consider the zero-reward Bellman operator $\mathcal{B}^\pi \nu(s,a) := \gamma \mathbb{E}_{s'\sim T(s,a), a'\sim\pi(s')}[\nu(s',a')]$ of a bounded function ν [2], $\nu : S \times A \to \mathbb{R}$. DualDICE optimizes the expression in Eq. (10).

$$\min_{\nu: S\times A\to\mathbb{R}} J(\nu) := \frac{1}{2}\mathbb{E}_{(s,a)\sim d^\mathcal{D}}\left[(\nu - \mathcal{B}^\pi\nu)(s,a)^2\right] - (1-\gamma)\mathbb{E}_{s_0\sim\mu, a_0\sim\pi(s_0)}\left[\nu(s_0,a_0)\right], \tag{10}$$

where the first term of the equation is the expected squared zero-reward Bellman error and the second specifies the initial stage. The authors of DualDICE [26] state that the first term alone leads to a trivial solution $\nu \equiv 0$, which is avoided by the second term that ensures $\nu^* > 0$. Moreover, they prove that the Bellman residuals of ν^* are exactly the desired distribution corrections (Eq. 11).

$$w_{\pi/\mathcal{D}}(s,a) = (\nu^* - \mathcal{B}^\pi\nu^*)(s,a). \tag{11}$$

[2] Note that ν is a state-action value function, analogous to Q-values, although Q-values were not utilized in this context for completeness and to maintain generality.

3 SimuDICE

In this work, we introduce SimuDICE, a novel algorithm for policy optimization in the offline setting. The essence of SimuDICE lies in its conceptual framework, a flexible way to improve policy performance, agnostic to the specific algorithms used at each stage. In this section, we discuss the high-level idea of the algorithm (Section 3.1), then we elaborate on the specific algorithm components in Section 3.2, and discuss how they interact with each other in Section 3.3. Each part starts with a general discussion of its role within the framework, followed by details about our implementation.

3.1 High level idea

SimuDICE utilizes a dynamic interplay between a learned world model and DICE estimations to generate high-quality synthetic experiences, which in turn are used to enhance policy learning. By continuously updating both the world model and the DICE estimations, the synthetic experiences are aligned with the policy optimization objective, while also improving the sample efficiency and robustness. Figure 2 shows the three main components and their interactions, which will be explained in more detail in Sections 3.2 and 3.3.

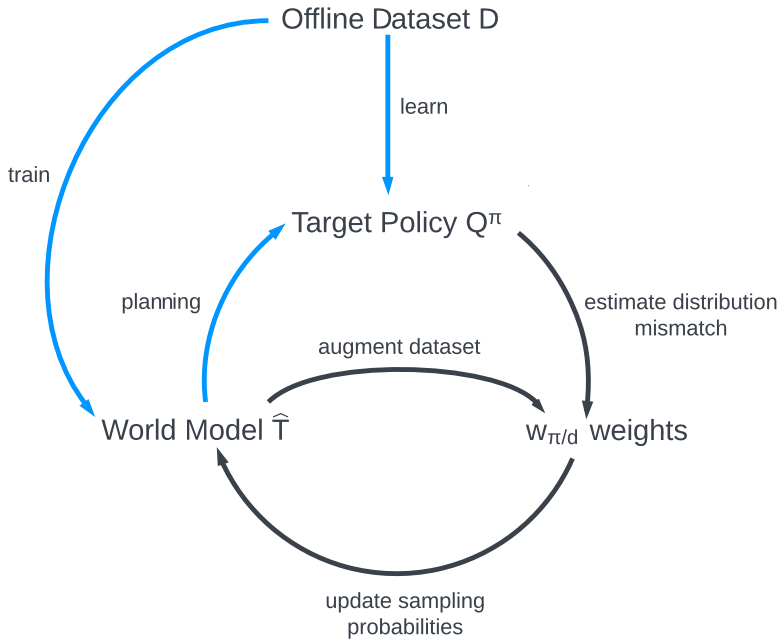

Figure 2: The components of SimuDICE and their interactions. Transitions adapted from Dyna-Q [34] are in blue, while those unique to SimuDICE are depicted in black.

The algorithm extends the Dyna-Q [34] framework to iterate until convergence, enabling continuous updates of both the DICE estimations and the sampling probabilities within the world model. Initially, offline data is utilized to pre-train a one-step forward dynamic model of the environment and to learn an initial target policy. This policy is iteratively refined by sampling experiences that are likely to be encountered by the *target policy*, as determined by the $w_{\pi/\mathcal{D}}$ weights, and the world model's confidence in sampling those experiences. The $w_{\pi/\mathcal{D}}$ estimates are improved using the synthetically generated experiences.

3.2 Algorithm components

In this section, we outline the general purpose of the three main components of SimuDICE: the *Target Policy* Q^π 3.2.1, the *World Model* $\hat{T}$ 3.2.2, and the $w_{\pi/\mathcal{D}}$ *weights* 3.2.3. This is followed by a discussion about our specific implementation details.

3.2.1 Target policy Q^π

The target policy in SimuDICE is used in the agent's decision-making process by using information from both the offline dataset and simulated experiences generated by the world model. This integration allows the agent to adapt to newly created scenarios as well as the ones it has previously seen.

We use the ϵ-greedy policy to balance exploration and exploitation, preventing the agent from getting stuck in local optima by randomly selecting an action with probability ϵ. The state-action value of the policy is represented by the Q-values $Q^\pi(s, a)$, which is the expected return by following the policy π starting at state s and performing action a. Note that in case all the Q-values for a state s are equal, we randomly select an action.

3.2.2 World model $\hat{T}$

The world model is a dynamic representation of the environment, enabling the augmentation of the offline dataset with synthetic experiences. This enhances the sample efficiency of the algorithm by covering unseen state-action pairs, helping both policy optimization and state-action distribution mismatch correction. It performs two main tasks: prediction and sampling. The prediction task involves mimicking the environment, specifically predicting the next state and the reward for an action: $(s, a) \rightarrow (s', r)$. The sampling task involves choosing a state-action pair (s, a) based on probabilities generated using the $w_{\pi/\mathcal{D}}$ weights and the model's confidence in that specific prediction.

In SimuDICE, we have implemented a Tabular World Model [35] that acts like a memory-based model, a lookup table that replicates previously seen experiences. The model averages the rewards present for a state-action pair present in the offline dataset and considers the next state one of the options, given that the environments are deterministic. We consider the confidence in predicting correctly an experience to be the number of occurrences of the state-action pair has in the offline dataset, divided by the total number of elements in the offline dataset.

3.2.3 $w_{\pi/\mathcal{D}}$ weights

The $w_{\pi/\mathcal{D}}$ weights represent the density ratio between the state-action visitation distribution of the *target policy* and that of the *behavioral policies* that collected the offline dataset. They help steer the sampling probabilities of the world model towards state-action pairs that the target policy is more likely to encounter. This focus on relevant state-action pairs avoids sampling states that might never be reached using the current policy and reduces the risk of generating hallucinated synthetic experiences.

Although this approach might be considered conservative because it limits the imaginary exploration of the entire state-action space, we believe that it is beneficial in environments where making mistakes is costly or risky. This design choice was made because having a slightly worse policy is better than having a completely wrong one due to hallucinated states.

3.3 Component interactions

In this section, we discuss how the different components of the algorithm interact. Section 3.3.1 shows how we have adapted several components from Dyna-Q [34] to this framework. Section 3.3.2 discusses how the distribution mismatch is estimated, while Section 3.3.3 shows how the sampling probabilities for the world model are updated using the $w_{\pi/\mathcal{D}}$ weights and the estimated confidence of the prediction. Lastly, in Section 3.3.4 we discuss how distribution mismatch estimation can be improved using the synthetically generated experiences.

3.3.1 Adapted Dyna-Q components

In this part, we discuss how each part of Dyna-Q [34] is adapted to the offline setting, except the acting component which is completely removed due to the offline setting.

World model training: In this stage, a one-step forward dynamic model via maximum likelihood estimation (MLE) is pre-trained on the offline dataset, aiming to be a simple mimic of the world. Since the scope of SimuDICE is limited to deterministic grid world environments, the world model averages the reward received for each state-action pair from the offline dataset and uses the next seen state as the prediction.

Initial policy learning: In this stage, we use the offline pre-collected dataset to derive an initial policy, using Experience Replay [5], which makes the policy more stable. Experience Replay has shown several improvements in breaking correlations, faster convergence, and avoiding bias [38]. Since SimuDICE uses the ϵ-greedy policy using Q-values, the initial policy is learned using implicit Q-learning [39], by shuffling the experiences in the dataset and applying the Q-learning formula as in Eq. (12).

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right], \tag{12}$$

where

- $Q(s,a)$ is the current Q-value for state s and action a.
- α is the learning rate.
- r is the reward received after taking action a in state s.
- γ is the discount factor.
- s' is the next state.
- $\max_{a'} Q(s',a')$ is the maximum Q-value over all possible actions a' for the state s'.

Planning Steps: The planning phase of the algorithm improves the policy using synthetically generated data, enabling the agent to learn from a more diverse range of experiences. The policy is updated using the same method as experiences from the offline dataset. In SimuDICE, experiences (s,a) are sampled using the World model $\hat{T}$ with different probabilities, calculated based on how likely is the experience to be encountered by running the policy and how confident is the world model on the prediction. The world model predicts the subsequent reward and next state (r,s') and updates the Q-values using Eq. 12.

3.3.2 Distribution mismatch estimation

In this stage, we aim to improve the performance of the target policy by accurately estimating the discounted stationary distribution ratios - correction terms that quantify the likelihood that an experience is visited by the new policy, normalized by the probability with which the state-action pair appears in the offline dataset.

In this work, we chose to estimate the Dual stationary DIstribution Corrections. Given the simple nature of the environments, we used DualDICE [26]. Moreover, since the state-action space is relatively small, the function estimation and minimization are performed via matrix multiplication instead of using a neural network to approximate the function's value. In SimuDICE, we assume that the *behavior policy* is not known; hence, the estimation is based only on the state-action distribution. If the *behavior policy* is known, DualDICE [26] proposes a more robust and efficient method based on policy ratios, which requires less data. Appendix A, Algorithm 2 shows the pseudocode of the algorithm is implemented. Note that DualDICE [26] optimizes ζ, which we assume equivalent to $w_{\pi/\mathcal{D}}$.

3.3.3 Sampling probabilities update

The goal of this part is to convert the $w_{\pi/\mathcal{D}}$ weight estimates and the prediction confidence of the world model into a probability function that can bias the sampling of experiences used in the planning step.

The $w_{\pi/\mathcal{D}}$ weights are estimated using DualDICE [26]. We consider the confidence of a prediction of this world model to be the number of occurrences of the state-action pair (s, a) in the offline dataset normalized by its size. Let $\mathcal{C}(s, a)$ denote the confidence of a prediction for the state s and action a.

Let $\mathcal{P}(s, a)$ be the probability that the world model will sample state s and action a. We consider $\mathcal{P}(s, a)$ as the normalized sum of the confidence $\mathcal{C}(s, a)$ and the regularized softmax of the $w_{\pi/\mathcal{D}}$ weights. The regularization term λ is introduced because in environments with a large state-action space the $w_{\pi/\mathcal{D}}$ are too small and floating point errors occur. Below the derivation of the probability function $\mathcal{P}(s, a)$ is shown. First, we define an intermediate probability $\tilde{\mathcal{P}}(s, a)$ as shown in Eq. (13).

$$\tilde{\mathcal{P}}(s, a) = \mathcal{C}(s, a) + \frac{e^{w_{\pi/\mathcal{D}}(s,a)\cdot\lambda}}{\sum_{(s',a')} e^{w_{\pi/\mathcal{D}}(s',a')\cdot\lambda}}/\lambda \tag{13}$$

Next, $\tilde{\mathcal{P}}(s, a)$ is normalized to obtain the final probability function $\mathcal{P}(s, a)$, as in Eq. (14):

$$\mathcal{P}(s, a) = \frac{\tilde{\mathcal{P}}(s, a)}{\sum_{(s',a')} \tilde{\mathcal{P}}(s', a')} \tag{14}$$

3.3.4 Dataset augmentation

The dataset augmentation step is beneficial only when the distribution mismatch estimation is done using a neural network that is impacted by the number of trajectories. Adding an experience to improve the $w_{\pi/\mathcal{D}}$ comes to finding a possible location for the state-action pair (s, a) in another trajectory (i.e., finding in the offline dataset the index of that state), and γ-discount the expected reward using that step number.

4 Experiments

In this section, we evaluate the performance of SimuDICE on three discrete grid-world environments and compare it with other similar algorithms as detailed in Section 4.2. Additionally, we conduct an ablation study to understand the efficiency of certain components of the proposed algorithm, as explained in Section 4.3. The setup for these experiments is described in Section 4.1.

4.1 Experimental setup

Environments: We evaluate the performance of different agents in three discrete grid-world environments: *Taxi*, *FrozenLake* and *CliffWalking* from the *Gymnasium Toy Example Library* [40]. These environments were chosen for their simplicity, deterministic nature, and tabular format, having relatively small state-action spaces. Figure 3 shows an illustration of these environments, and Appendix B elaborates on their reward model.

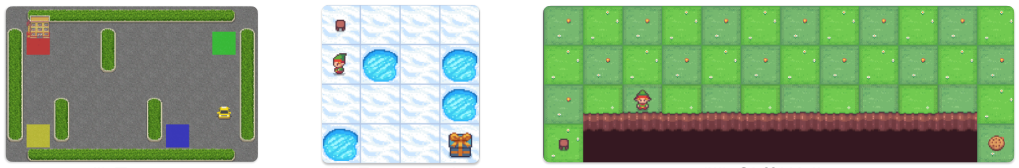

Figure 3: Illustration of the three gridworld environments used for evaluating the algorithm, adapted from [40]. The environments include: (I) Taxi, where the agent must pick up and drop off passengers at designated locations, (II) Frozen Lake, where the agent must navigate across a surface to reach a goal while avoiding holes, and (III) Cliff Walking, where the agent must traverse a grid while avoiding a cliff edge to reach the goal.

Off-policy Data Collection Method: Given the offline setting of this work, we chose to create the statically pre-collected datasets through an online process, implemented in two stages using Q-learning [37]. In the first stage, the agent learns optimal Q-values by interacting with the environment over $10,000$ episodes. To ensure diverse data collection, we use an ϵ-greedy policy where the agent selects the action with the highest Q-value $1-\epsilon$ of the time and a random action the remaining ϵ of the time, balancing exploration and exploitation. We collect the dataset for a limited number of episodes, each with a maximum of 100 environment steps, using three different ϵ values: 0.1, 0.4, and 0.7. These diverse data collection policies enable a more effective evaluation of the algorithm's ability to learn from differing-quality datasets.

SimuDICE Hyperparameters: Throughout these experiments, we use the following hyperparameters for the SimuDICE algorithm unless stated otherwise:

- **Learning rate** (α): 0.1
- **Discount factor** (γ): 0.99
- **Planning steps**: 20 (number of planning steps performed for each offline experience)
- **Number of iterations**: 3 (number of updates for the sample probabilities)
- **Probability regularizer** (λ): 100

4.2 Discrete gridworld environments results

We compare SimuDICE with two other algorithms: Implicit Q-learning [39] and offline adapted Dyna-Q [34]. The Implicit Q-learning algorithm is part of SimuDICE that learns the target policy only from the offline data, without further improvement. The offline adapted Dyna-Q is another component of SimuDICE, and it improves the policy in a single iteration using equal sampling probabilities for each state-action pair.

As shown in Figure 4, our SimuDICE either outperforms or matches the performance of other algorithms across various settings. In particular, in the Taxi environment, which is considered more challenging due to its larger state-action space and the diversity of actions and penalties, SimuDICE significantly outperforms the others. Even with only 10 planning steps, SimuDICE maintains close performance levels to those with 20 planning steps, often surpassing 20-step Dyna-Q.

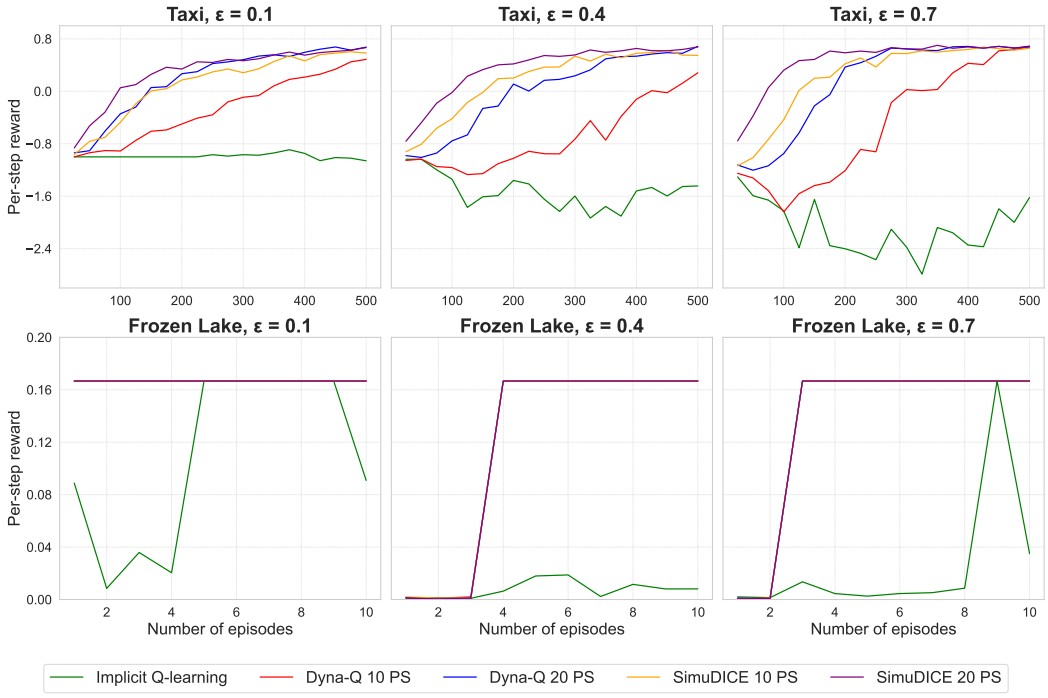

Figure 4: Comparison of algorithm performance in discrete tabular environments (Taxi and FrozenLake) under different ϵ-greedy data collection policies. The different ϵ values simulate varying offline data qualities. The performance is measured as the per-step reward (averaged over 500 plays) given different numbers of trajectories in the offline data. PS represents the planning steps for both SimuDICE and Dyna-Q algorithms.

In the FrozenLake environment, there is no noticeable difference between the algorithms that use planning, all having the exact same per-step reward. The only observation is that the planning-based algorithms outperform vanilla Q-learning in all cases, irrespective of the quality of the offline data. This conclusion also applies to the CliffWalking environment, as shown in Appendix D, Figure 8.

4.3 Ablation study

In this section, we conduct an ablation study to evaluate the effect of various added parameters on the performance of SimuDICE. Our analysis focuses exclusively on the *Taxi* environment, as it represents the most complex scenario out of the three.

Planning Steps: *How does the number of planning steps affect the model's performance?*

We carry out an experimental evaluation to determine how different numbers of planning steps affect the agent's performance. Figure 5 shows that while the number of planning steps improves the performance, the relationship is not linear. The benefit is more obvious in environments with higher data quality (bigger ϵ), where the *behavioral policy* tends to exploit rather than explore, as seen in the Taxi environment.

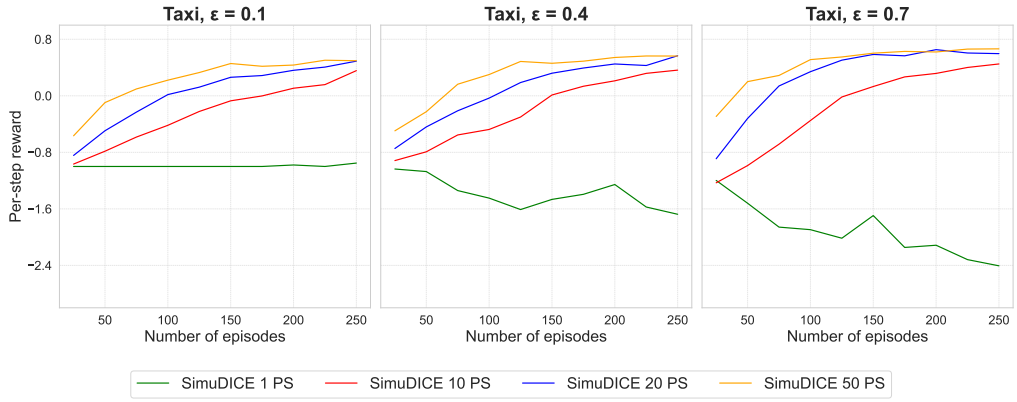

Figure 5: Impact of the number of planning steps on per-step reward under different ϵ values.

Number of iterations: *How does the number of iterations (the number of updates to the sample probabilities) change the performance of the algorithm?*

To verify the effectiveness of the number of iterations (i.e., the frequency of updating the sampling probabilities) on the performance of the agent, we conducted a comparative analysis using the SimuDICE with 20 planning steps.

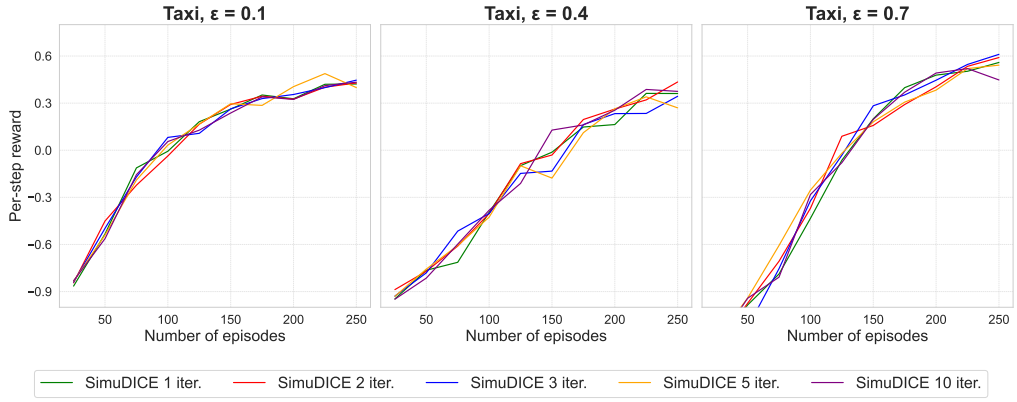

Figure 6: Effect of the number of iterations on SimuDICE under different ϵ-values.

Figure 6 shows that varying the number of iterations has a negligible effect on the performance of SimuDICE in the Taxi environment, showing that SimuDICE is unaffected by the number of iterations in this environment.

Different Sampling Probabilities Formulas: *How does the algorithm's performance change when we alter the method for estimating sampling probabilities?*

We compare our SimuDICE with three alternative variants, each employing different formulas for converting model confidence and $w_{\pi/\mathcal{D}}$ weight estimations into probabilities. This comparison aims to assess their effectiveness in guiding the world model to sample more 'valuable' synthetic experiences. For simplicity, we assume that each probability function is normalized to sum 1. Formula 1 is detailed in Section 3.3.3 and shown in Eq. 15. Formula 2 applies the softmax function to the sum of the world model's confidence and the $w_{\pi/\mathcal{D}}$ estimates, as presented in Eq. 16. Formula 3 applies the softmax function solely to the $w_{\pi/\mathcal{D}}$ weight estimates, without considering the world model's confidence (Eq. 17). All of these methods are compared with the baseline of randomly sampling experiences.

$$\tilde{\mathcal{P}}(s,a) = \mathcal{C}(s,a) + \frac{e^{w_{\pi/\mathcal{D}}(s,a)\cdot\lambda}}{\sum_{(s',a')} e^{w_{\pi/\mathcal{D}}(s',a')\cdot\lambda}}/\lambda \tag{15}$$

$$\tilde{\mathcal{P}}(s,a) = \frac{e^{\mathcal{C}(s,a)+w_{\pi/\mathcal{D}}(s,a)}}{\sum_{(s',a')} e^{\mathcal{C}(s',a')+w_{\pi/\mathcal{D}}(s',a')}} \tag{16}$$

$$\tilde{\mathcal{P}}(s,a) = \frac{e^{w_{\pi/\mathcal{D}}(s,a)\cdot\lambda}}{\sum_{(s',a')} e^{w_{\pi/\mathcal{D}}(s',a')\cdot\lambda}}/\lambda \tag{17}$$

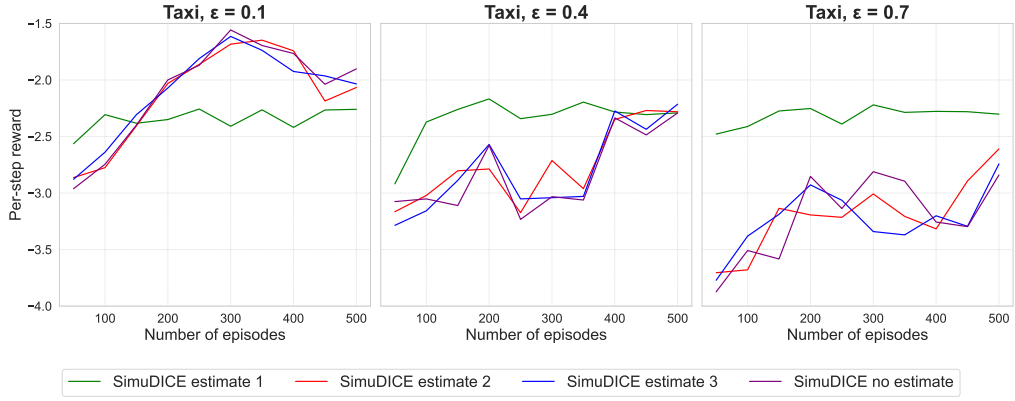

Figure 7: Comparison of per-step rewards across various sample probability estimation formulas, based on different trajectory counts (evaluated every 50 trajectories) and ϵ-values.

Figure 7 demonstrates that the formula used in SimuDICE is more robust than others under varying data qualities. However, when the *target policy* is close to the *behavioral policy* used for data collection, alternative sampling methods may outperform it. Specifically, the SimuDICE formula excels in scenarios with diverse data but yields inferior results when the data lacks diversity and is already close to the desired distribution.

5 Discussion

The purpose of this work is to propose a novel framework designed for policy optimization in the offline reinforcement learning setting by addressing the *data needs*, the *state-action distribution mismatch*, and the *objective mismatch*. In this section, we discuss the findings and their implications, including limitations and areas for improvement.

Key findings and implications:

- **Improved sampled efficiency:** We demonstrate that SimuDICE preserves the efficiency improvements introduced by Dyna-Q [34], particularly in generating imaginary rollouts. We believe that the process of generating these synthetic experiences, referred to as 'imagination', is vital for developing effective decision-making agents.
- **Distribution mismatch correction:** SimuDICE achieves similar results with fewer planning steps by updating the sampling probabilities of the world model. In the Taxi environment, SimuDICE required half the planning steps (10) to achieve comparable performance to Dyna-Q (20). This efficiency improvement is significant as it reduces the need for both pre-collected and simulated data. By requiring fewer simulated experiences, the risk of encountering hallucinated experiences that could lead to policy divergence in more complex environments is decreased. Moreover, our ablation study highlights that our formula used to calculate the sampling probabilities is robust to different offline dataset qualities.
- **Objective mismatch improvement:** Our ablation study shows no performance gain by updating the sampling probabilities multiple times during a run. We believe this is because the objective mismatch was theorized for more complex world models in complex environments by [32, 33], and in our implementation, this problem does not arise. Future research should be done in this area. Still, we anticipate that when scaled to more complex environments SimuDICE will achieve similar improvements as the AMPL algorithm [18] due to constantly updating the objective of the world model.

Limitations:

- **Simple environments:** The experimental study was conducted in simple, deterministic grid-world environments such as Taxi, FrozenLake, and CliffWalking. These environments do not capture the complexity and stochastic nature of real-world scenarios, which might affect the generalizability of the results.
- **Sensitivity to sample probabilities formula:** In the ablation study, we show that SimuDICE's performance is significantly influenced by the formula used for estimating sampling probabilities. Given the limited number of environments in the current experimental setup, a more extensive evaluation across a wider range of environments is necessary to assess the effectiveness of the sampling probability formula thoroughly.
- **Comparison with other algorithms:** This study compared SimuDICE only with an offline version of Dyna-Q and Implicit Q-learning. One limitation of this work is the exclusion of other algorithms from the comparison, as these algorithms might offer different advantages or better performance.

6 Conclusion and Future Work

This study introduces SimuDICE, a novel framework designed to optimize policies in the offline reinforcement learning setting through iterative updates of both the world model's sampling probabilities and the Dual stationary DIstribution Correction (DICE) estimation using synthetically generated experiences. The core advancement of SimuDICE is its ability to correct the state-action distribution mismatch between the *behavior policy* and the *target policy*, achieved through a bi-objective optimization of the realism and diversity of the generated experiences. The world model samples experiences that are more likely to be encountered by the target policy, ensuring diversity, while maintaining high confidence in the generated experiences' accuracy, ensuring realism. Optimizing solely for realism can lead to a less diverse, mode-collapsed model [41] while focusing only on diversity can result in hallucinated states [17].

Our experiments show that SimuDICE outperforms similar algorithms, such as an offline version of Dyna-Q [34] and Implicit Q-learning [39]. The performance improvement compared to Implicit Q-learning shows better sample efficiency. SimuDICE achieves results similar to Dyna-Q with only half the planning steps, indicating its ability to correct for state-action distribution mismatches. Additionally, SimuDICE is less affected by the quality of offline data compared to the other algorithms. Our ablation study reveals that SimuDICE is sensitive to the probabilities the world model samples with while being robust to changes in other added hyperparameters like planning steps and the number of iterations.

Future work: This work introduces a proof-of-concept algorithm that steers the distribution of synthetically generated experiences towards both 'relevant' and 'confident' ones. Future research should extend this algorithm to more complex environments to evaluate scalability and compare it with state-of-the-art methods.

Enhancing the current framework involves using a sample efficient world model that can generate novel experiences, using neural-network-based approaches [2]. Additionally, using a more stable DICE estimation is important to ensure convergence [29]. Implementing an actor-critic policy could further improve decision-making within the algorithm [42]. These enhancements will require exploring the necessary changes in transitions between components, allowing us to evaluate the robustness, efficiency, and scalability of this framework.

Acknowledgments

I want to thank our professor, Frans Oliehoek, and our supervisor, Stephan Bongers, for their support, guidance, and helpful feedback, which have been instrumental in shaping this project. I also want to express my gratitude to my group members for the discussions and feedback we shared during our meetings, as well as to the anonymous reviewers for their insightful feedback.

7 Responsible Research

We developed SimuDICE by closely adhering to the principles outlined in the Dutch Code of Conduct for Research Integrity [43]: honesty, scrupulousness, transparency, independence, and responsibility.

Honesty: The implementation of the algorithm, experimental results, and logs from the runs that produced the presented results are publicly available in a GitHub repository. We acknowledge the importance of sharing research and have chosen the Apache License 2.0, which allows other researchers to contribute to this work.

Scrupulousness and Transparency: The environments used to evaluate the algorithms are publicly available in the Gymnasium Library [40], with no modifications made to them or their reward models. These environments were selected based on their type (e.g., deterministic, tabular) rather than their topic or theme.

Independence: Our evaluation criteria focused on being scientific and reproducible. We ensured that no algorithm was disadvantaged by running all experiments multiple times. All result logs are available in the repository, and no findings have been intentionally excluded.

Responsibility: According to the tripartite model (users, engineers, politicians) [44], engineers are accountable for creating products and upholding moral principles. Therefore, we recommend careful consideration before using this algorithm in real-world scenarios, as independent evaluations of risks and other factors are necessary.

This paper clearly explains the assumptions and decisions made during the development of this novel algorithm. It analytically discusses the discovered limitations and suggests directions for future work. We believe that this thorough approach is particularly relevant given the current reproducibility crisis in science [45].

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[2] D. Hafner, J. Pasukonis, J. Ba, and T. Lillicrap, "Mastering diverse domains through world models," *arXiv preprint arXiv:2301.04104*, 2023.

[3] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[4] J. Gao, M. Galley, and L. Li, "Neural approaches to conversational AI," *CoRR*, vol. abs/1809.08267, 2018.

[5] 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.

[6] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[7] S. Lange, T. Gabel, and M. Riedmiller, "Batch reinforcement learning," in *Reinforcement Learning: State-of-the-Art*, M. Wiering and M. van Otterlo, Eds. Springer, 2012, ch. 2, pp. 45–73.

[8] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[9] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 2052–2062.

[10] A. Kumar, J. Fu, G. Tucker, and S. Levine, "Stabilizing off-policy q-learning via bootstrapping error reduction," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[11] L. Baird, "Residual algorithms: Reinforcement learning with function approximation," *Machine Learning*, vol. 20, no. 1-2, pp. 65–81, 1995.

[12] J. N. Tsitsiklis and B. Van Roy, "Analysis of temporal-difference learning with function approximation," in *Advances in Neural Information Processing Systems 9 (NIPS 1996)*. MIT Press, 1996, pp. 1075–1081.

[13] D. Hafner, T. Lillicrap, M. Norouzi, and J. Ba, "Dreamer: Reinforcement learning with latent world models," in *International Conference on Learning Representations*, 2020.

[14] D. Hafner, J. Schrittwieser, D. Mankowitz, A. Barreto, and T. Lillicrap, "Mastering atari with discrete world models," *arXiv preprint arXiv:2010.02193*, 2020.

[15] E. Alonso, A. Jelley, V. Micheli, A. Kanervisto, A. Storkey, T. Pearce, and F. Fleuret, "Diffusion for world modeling: Visual details matter in atari," 2024.

[16] Z. Ding, A. Zhang, Y. Tian, and Q. Zheng, "Diffusion world model," 2024.

[17] T. Jafferjee, E. Imani, E. J. Talvitie, M. White, and M. Bowling, "Hallucinating value: A pitfall of dyna-style planning with imperfect environment models," *arXiv preprint arXiv:2006.04363*, 2020.

[18] S. Yang, S. Zhang, Y. Feng, and M. Zhou, "A unified framework for alternating offline model training and policy learning," in *Advances in Neural Information Processing Systems*, 2022. [Online]. Available: https://arxiv.org/abs/2210.05922

[19] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *International Conference on Learning Representations*, 2015.

[20] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, W. Pieter Abbeel, OpenAI, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, 2017.

[21] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less real time," in *Machine Learning Proceedings 1993*, 1993.

[22] T. Zhang, P. Ball, P. Ammanabrolu, D. Peng, S. Singh, and J. Pineau, "Learning to plan with uncertainty," in *International Conference on Learning Representations*, 2021.

[23] D. Precup, R. S. Sutton, and S. Dasgupta, "Off-policy temporal-difference learning with function approximation," in *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, C. E. Brodley and A. P. Danyluk, Eds. San Francisco, CA: Morgan Kaufmann, 2001, pp. 417–424.

[24] A. Hallak and S. Mannor, "Consistent on-line off-policy evaluation," in *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)*. PMLR, 2017, pp. 1372–1383.

[25] Q. Liu, L. Li, Z. Tang, and D. Zhou, "Breaking the curse of horizon: Infinite-horizon off-policy estimation," in *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. Curran Associates, Inc., 2018, pp. 5356–5366.

[26] O. Nachum, Y. Chow, B. Dai, and L. Li, "Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections," in *Advances in Neural Information Processing Systems*, 2019, pp. 2315–2325.

[27] R. Zhang, B. Dai, L. Li, and D. Schuurmans, "Gendice: Generalized offline estimation of stationary values," in *International Conference on Learning Representations*, 2020.

[28] S. Zhang, B. Liu, and S. Whiteson, "Gradientdice: Rethinking generalized offline estimation of stationary values," *arXiv preprint arXiv:2001.11113*, 2020.

[29] M. Yang, O. Nachum, B. Dai, L. Li, and D. Schuurmans, "Off-policy evaluation via the regularized lagrangian," in *International Conference on Learning Representations*. ICLR, 2020.

[30] P. Swazinna, S. Udluft, and T. A. Runkler, "Overcoming model bias for robust offline deep reinforcement learning," *Engineering Applications of Artificial Intelligence*, vol. 104, p. 104366, 2021.

[31] C. Cang, A. Rajeswaran, P. Abbeel, and M. Laskin, "Behavioral priors and dynamics models: Improving performance and domain transfer in offline rl," *arXiv preprint arXiv:2106.09119*, 2021, abs/2106.09119.

[32] N. Lambert, B. Amos, O. Yadan, and R. Calandra, "Objective mismatch in model-based reinforcement learning," *arXiv preprint arXiv:2002.04523*, 2020.

[33] B. Eysenbach, A. Khazatsky, S. Levine, and R. Salakhutdinov, "Mismatched no more: Joint model-policy optimization for model-based rl," *arXiv preprint arXiv:2110.02758*, 2021.

[34] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM SIGART Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[35] R. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," *Proceedings of the Seventh International Conference on Machine Learning*, pp. 216–224, 1990.

[36] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: John Wiley & Sons, 1994.

[37] C. J. Watkins, "Learning from delayed rewards," Ph.D. dissertation, University of Cambridge, 1989.

[38] W. Fedus, P. Ramachandran, R. Agarwal, Y. Bengio, H. Larochelle, M. Rowland, and W. Dabney, "Revisiting fundamentals of experience replay," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 3061–3071.

[39] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit q-learning," *arXiv preprint arXiv:2110.06169*, 2021.

[40] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, "Gymnasium," Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025

[41] P. Astolfi, M. Careil, M. Hall, O. Mañas, M. Muckley, J. Verbeek, A. R. Soriano, and M. Drozdzal, "Consistency-diversity-realism pareto fronts of conditional image generative models," 2024.

[42] I. Grondman, L. Busoniu, G. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 42, no. 6, pp. 1291–1307, 2012.

[43] KNAW NWO VH VSNU, NFU, "Netherlands code of conduct for research integrity," Sep. 2018.

[44] I. Van de Poel and L. M. M. Royakkers, *Ethics, Technology, and Engineering: An Introduction*. Wiley-Blackwell, 2011.

[45] M. Baker, "1,500 scientists lift the lid on reproducibility," *Nature*, vol. 533, pp. 452–454, May 2016.

A DualDICE

Algorithm 2 Get Weight Estimates for DualDICE

Require: Offline behavioral data $\mathcal{D}$, Q-values Q, target policy π, regularizer $\lambda = 10^{-8}$

$\delta \leftarrow \mathbf{0}_{|A|\cdot|S|\times|A|\cdot|S|}$
total_weights $\leftarrow \mathbf{0}_{|A|\cdot|S|}$
initial_weights $\leftarrow \mathbf{0}_{|A|\cdot|S|}$
for each episode in $\mathcal{D}$ **do**
    $step = 0$
    for each (s, a, r, s') in episode **do**
        $\nu_{\text{index}} \leftarrow s \cdot |A| + a$
        $weight \leftarrow \gamma^{step}$
        $\delta[\nu_{\text{index}}, \nu_{\text{index}}] \leftarrow \delta[\nu_{\text{index}}, \nu_{\text{index}}] + weight$
        total_weights$[\nu_{\text{index}}] \leftarrow$ total_weights$[\nu_{\text{index}}] + weight$
        $p(\pi_s) \leftarrow \pi$.get_state_probabilities(s)
        for each $(a, p(\pi_s, a))$ in $p(\pi_s)$ **do**
            $\nu_{\text{next\_index}} \leftarrow s \cdot |A| + a$
            $\delta[\nu_{\text{next\_index}}, \nu_{\text{index}}] \leftarrow \delta[\nu_{\text{next\_index}}, \nu_{\text{index}}] - weight \cdot \gamma \cdot p(\pi_s, a)$
        end for
        $p(\pi_{s_0}) \leftarrow \pi$.get_state_probabilities$(s)$
        for each $(a, p(\pi_{s_0}, a))$ in $p(\pi_{s_0})$ **do**
            $\nu_{\text{next\_index}} \leftarrow s_0 \cdot |A| + a$
            $\delta[\nu_{\text{next\_index}}, \nu_{\text{index}}] \leftarrow \delta[\nu_{\text{next\_index}}, \nu_{\text{index}}] - weight \cdot \gamma \cdot p(\pi_{s_0}, a)$
        end for
        $step \leftarrow step + 1$
    end for
end for
Normalization:
td_residuals $\leftarrow \frac{\delta}{\sqrt{\lambda + \text{total\_weights}[None,:]}}$
td_errors $\leftarrow$ td_residuals $\cdot$ td_residuals^T
Solve for ν in the linear system td_errors $+ \lambda \cdot I =$ initial_weights $\cdot (1 - \gamma)$
$\zeta \leftarrow \frac{\nu \cdot \text{td\_residuals}}{\sqrt{\lambda + \text{total\_weights}}}$

B Environment Reward Models

Taxi: The Taxi environment involves picking up passengers from designated locations and delivering them to their destinations. The agent receives positive rewards (+20) for successfully delivering passengers and negative rewards for any movement (-1) or for incorrectly picking up or dropping off a passenger (-10).

FrozenLake: The FrozenLake environment requires navigating a frozen lake to reach a goal while avoiding holes. For simplicity, there is no added stochasticity to the environment. The agent receives a positive reward (+1) for reaching the goal and zero reward for each step taken. Falling into a hole ends the episode with no reward. The primary challenge is to find the optimal path to the goal while avoiding the holes.

CliffWalking: The CliffWalking environment involves navigating a grid world where the agent must reach a goal at the opposite end of the grid while avoiding a cliff. The agent incurs a significant negative reward (-100) if it steps off the cliff and is teleported back to the start. Additionally, the agent receives a small negative reward (-1) for each step taken. The challenge is to find a path to the goal that minimizes both the risk of falling off the cliff and the number of steps taken.

C Offline data collection method

Offline data collection is done using Q-learning [37] in 2 stages:

1. **Q-value learning:** In this phase the data collection agent learns the Q-values by interacting with the environment for a pre-defined number of episodes using the ϵ-greedy policy.
2. **Data collection phase:** In this phase the previously learned Q-values are used to store the interactions with the environment as a list of the form (s, a, r, s'). The collection method also uses ϵ-greedy as policy.

This data collection mechanism has three parameters, each of which influences the state-action distribution mismatch:

- **Number of episodes used for Q-value learning:** mostly influences the quality of the policy the data is collected under. When the number of episodes is high, the behavioral policy is closer to the optimal policy.
- **Number of episodes used for Data collection phase:** affects the amount of diverse experiences available. When the number of episodes is high, more robust data is generated, covering a more complete state-action distribution.
- **Selected exploration term ϵ:** controls the trade-off between exploration and exploitation. A higher ϵ encourages more exploration, which can help with discovering better policies by sampling a wider range of actions. A lower ϵ favors exploitation of the current knowledge, potentially improving the policy faster, but at the risk of converging to a suboptimal solution.

D CliffWalking Results

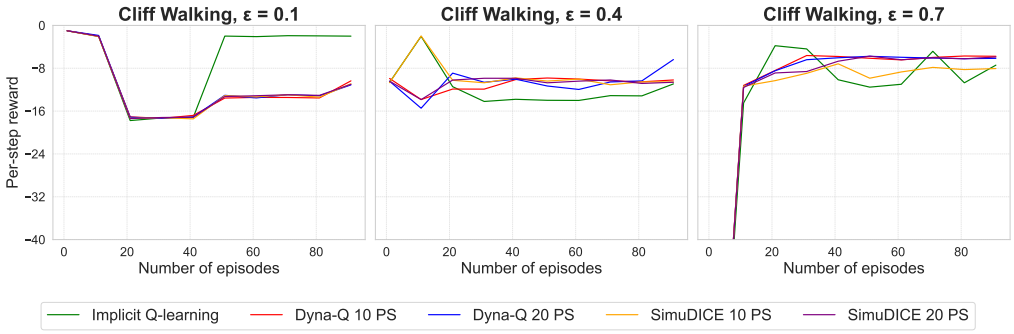

Figure 8: Comparison of Per-Step rewards across various sample probability estimation formulas, based on different trajectory counts and ϵ-values.