Chapter 5
Performance Evaluations

Abstract In this chapter, we discuss performance evaluation metrics and methods in details. Performance evaluation of reinforcement learning systems or algorithms are important to system monitoring and development improvements. It can be carried out in both offline and online. Offline evaluation evaluates the agent's performance on one or multiple fixed datasets of experiences. Online performance evaluation assesses the agent's performance in real-time interaction with the environment. Benchmarking is a common process to compare the performance of a newly developed reinforcement learning algorithm or reinforcement learning system to that of the established ones. It's generally carried out offline but some literatures also refer A/B testing as benchmarking in the sense that the method to be compared with is the reference one. Performance evaluation through simulator and emulator is another way to assess the reinforcement learning performance, and the effectiveness of this method itself heavily depends on the quality of the simulator or emulator used.

5.1 Evaluation Metrics

The key metrics for reinforcement learning performance evaluation include reward, sucess rate, learning curve, sample efficiency, exploration vs exploitation, and convergence. By carefully considering these metrics and evaluation methods, you can effectively assess the performance of your RL agents and make informed decisions about their development and deployment. We list the general definition of these metrics as below:

- Reward
  - Average Reward: The cumulative reward divided by the number of episodes or steps. Two popular average reward is used: average reward per episode and average reward per step. Average reward per episode is the total reward divided by the number of episodes. Average reward per step is the total reward divided by the total number of steps.
  - Discounted Reward: The sum of rewards, discounted by factors that give more weight to immediate rewards. In the majority of literatures, weights used grow

exponentially along time with the same discount factor γ. This helps to balance short-term and long-term goals.
  - Average Discounted Reward: The average discounted reward per episode or step.
  - Cumulative Reward: The total reward accumulated over a fixed number of episodes or steps.

- Success Rate: The percentage of episodes in which the agent successfully achieves a predefined goal or reaches a specific state, indicating the effectiveness of the agent's strategy and decision-making capabilities. A higher success rate suggests that the agent is effectively learning and adapting to its environment.
- Episode Length: This metric refers to the average number of steps taken per episode, offering insights into the complexity of the tasks being tackled. Longer episode lengths may indicate more intricate challenges, while shorter lengths might suggest simpler tasks.
- Learning Curve: A graphical representation that illustrates how the agent's performance, such as average reward or success rate, improves over time. This curve allows us to visualize the learning process and understand whether the agent is effectively acquiring new skills and knowledge.
- Sample Efficiency: This term describes the amount of experience, quantified by the number of interactions with the environment, required for the agent to achieve a desired level of performance. High sample efficiency is crucial in environments where obtaining samples is costly or time-consuming.
- Exploration vs. Exploitation: This concept encompasses the trade-off between exploring new actions to uncover potentially better rewards and exploiting known good actions to maximize immediate returns. Striking the right balance is essential for achieving optimal performance in various environments.
- Convergence: This refers to whether the agent's performance stabilizes over time or continues to improve indefinitely. A convergent agent will reach a performance plateau, while one that continues to improve demonstrates ongoing learning and adaptation to its environment.

Among these metrics, cumulative reward, Success Rate, learning curve, sample efficiency, and convergence are used frequently as performance metrics in various reinforcement learning systems. There are additional Considerations that can be used as performance metrics, depending on the applications. We briefly describe each of these considerations as below:

- Task Complexity:The difficulty of the reinforcement learning (RL) task can significantly influence the choice of metrics used for evaluation as well as the expected performance outcomes. More complex tasks may require more sophisticated evaluation metrics to capture the nuances of agent behavior and performance effectively.
- Environment Dynamics:The nature of the environment, whether it is deterministic, stochastic, continuous, or discrete, can profoundly impact the evaluation process. Different dynamics can lead to variations in agent performance and behavior,

which necessitate tailored evaluation strategies to accurately assess the agent's capabilities.
- Agent Architecture:The complexity and design of the RL agent, including its neural network architecture and the algorithms it employs, can substantially affect its performance. Variations in architecture can lead to different learning efficiencies, generalization capabilities, and overall effectiveness in solving tasks.
- Hyperparameters:The choice of hyperparameters, such as learning rate, discount factor, and exploration strategies, can significantly influence the agent's behavior and learning efficiency. The tuning of these parameters is crucial, as they can dramatically alter the learning trajectory and final performance of the RL agent in various environments.

There are also specific metrics for different tasks. We present several examples for specific tasks as below:

- Navigation Tasks
  - Distance Traveled: The total distance covered by the agent.
  - Goal Completion Time: The time taken to reach the goal. It tells how quickly the agent can achieve a goal
  - Energy Consumption: For tasks involving physical agents, energy consumption can be a critical metric.
- Control Tasks
  - Control Error: The deviation from the desired control signal.
  - Stability: The ability to maintain a stable state.
  - Safety Metrics: In safety-critical applications, metrics like collision rate or violation of constraints are essential.
- Game-Playing tasks
  - Win Rate: The percentage of games won.
  - Score: The total score achieved in the game.

Most of these performance metrics and considerations can be either used in offline performance evaluation, online performance evaluation, or both.

5.2 Offline Performance Evaluation

The offline evaluation is usually more straight-forward than online evaluation. The evaluation is carried out on offline datasets that are usually predefined before the training. For RL problem with episodic behaviors, many episodes in the testing dataset are conducted with different initial conditions which are usually randomized and preset parameters like the exploration rate ϵ, for upto a time limit or number limit to avoid the experiment/evaluation runs for too long.

Evaluating the performance of an RL agent offline also presents unique challenges due to the lack of real-time interaction. Here are some common approaches.

- On-Policy Evaluation
  - Direct Policy Evaluation (DPE): This involves directly evaluating the policy on the offline dataset. However, it can be biased if the offline dataset is not representative of the true environment.
  - Importance Sampling: This technique weights the samples in the dataset based on how likely they are to be encountered under the current policy. It can help to reduce bias but can suffer from high variance if the policies are very different.
  - Doubly Robust Estimators: Combine DPE and importance sampling to reduce bias and variance.
- Off-Policy Evaluation
  - Q-value-based Methods: Estimate the Q-values of the current policy on the offline dataset and use them to evaluate its performance.
  - Behavior Cloning: Train a supervised learning model to mimic the behavior of the original policy on the offline dataset. This can be used to evaluate the performance of the original policy indirectly.
  - Counterfactual Estimators: Estimate the counterfactual outcomes of different actions under the current policy, allowing for a more direct evaluation.
- Simulation-Based Evaluation
  - Synthetic Environments: Create simulated environments that mimic the real-world environment and evaluate the agent's performance in these simulations.
- Benchmarking
  - Standard Datasets: Use publicly available datasets to compare the performance of different offline RL algorithms.
  - Real-World Applications: Evaluate the performance of offline RL agents on real-world tasks to assess their practical applicability.

Offline evaluation of reinforcement learning systems faces several challenges and considerations with respect to data quality, distribution shift, evaluation bias and computational cost:

- Data Quality: The quality of the offline dataset is crucial for accurate evaluation. A biased or incomplete dataset can lead to misleading results.
- Distribution Shift: The distribution of states and actions in the offline dataset may not match the distribution in the real world, leading to performance degradation.
- Evaluation Bias: The choice of evaluation method can introduce bias, affecting the accuracy of the results.
- Computational Cost: Some evaluation methods can be computationally expensive, especially for large datasets or complex models.

By carefully considering these factors and selecting appropriate evaluation techniques, you can effectively assess the performance of offline RL agents and make informed decisions about their development and deployment.

5.2.1 On-Policy Evaluation

On-policy evaluation (OPE) methods evaluate a policy using data from the target policy of interest. For RL problems where data is not sparse, OPE data is collected via data sampling methods for feasibility and efficency. However, there is a subtle difference between OPE data and OPE sampling in the context of policy evaluation. Specifically, OPE sampling may fail to match the expected distribution of on-policy data after observing only a finite number of trajectories and this failure may hinder data-effective and data-efficient policy evaluation.

In some scenarios, on-policy sampling may become inefficient and the sample data become biased away from the expected on-policy data distribution. To achieve improved data efficiency, we present a comprehensive analysis demonstrating how non-independent and identically distributed (non-i.i.d.), off-policy sampling techniques can generate data that more closely resembles the expected on-policy data distribution. This alignment is crucial, as it significantly enhances the accuracy and reliability of the Monte Carlo estimator used for policy evaluation. By leveraging these advanced sampling methods, we can optimize the learning process and improve the overall effectiveness of reinforcement learning algorithms. To better understand this point of view, consider a simple yet illustrative example. In this example, a certain target policy repeatedly visits a particular state in which it takes action A with a probability of 0.2 and action B with a probability of 0.8. Under on-policy sampling, after five visits to this state, we might actually observe action A occurring 2 times and action B occurring 3 times, which deviates from the expected frequencies of 1 and 4 times, respectively. This variability highlights the inherent randomness involved in on-policy sampling, where the observed outcomes can fluctuate significantly from theoretical expectations. Alternatively, we could collect data off-policy by deterministically tracking the expected action proportions dictated by the target policy; doing so results in observing the exact expected action frequencies of 1 for action A and 4 for action B. Though the latter case employs off-policy sampling, it produces data that is arguably more in line with on-policy behavior than the data generated by on-policy sampling itself. This phenomenon raises interesting questions about the reliability and effectiveness of different sampling methods in policy evaluation and reinforces the importance of understanding how various approaches impact the quality of data derived from these strategies.

In the realm of policy evaluation, a common problem setting is a specific evaluation policy tasked with estimating the expected return that would be realized when implementing this evaluation policy on a chosen task of interest. This problem holds significant importance for the high-confidence deployment of RL-trained policies. In various RL applications, particularly in fields such as robotics, the significance of data-efficient policy evaluation cannot be overstated. Practitioners and researchers alike desire to achieve the most accurate estimates while minimizing the amount of data collected, as excessive data gathering can be time-consuming and resource-intensive. While extensive research has been conducted on how to efficiently leverage a set of already collected data—known as the off-policy policy evaluation problem —

there remains an implicit assumption within the RL community that, when available, on-policy data is inherently more valuable than off-policy data.

When data can be collected in an on-policy manner, the Monte Carlo estimator becomes a powerful tool, as it computes a mean return estimate by utilizing trajectories that are sampled independently and identically distributed (i.i.d.) through the execution of the evaluation policy. In an ideal scenario, where an infinite number of trajectories are gathered, the empirical proportions of each trajectory will converge to their true probabilities under the evaluation policy, leading the estimate to approach the true expected return. However, in reality, the limitations of any finite sample size come into play, causing the empirical proportions of each trajectory to likely diverge from the true probabilities, thereby introducing error into the estimates. This sampling error is an inherent characteristic of i.i.d. sampling. Specifically, the probability of each new trajectory remains unaffected by the trajectories that occurred previously, which means that the only method to ensure that the empirical distribution aligns with the true probability is to gather a sufficiently large dataset. In other words, it is only in the theoretical limit that on-policy sampling yields data that is precisely on-policy.

The observations made thus far prompt an intriguing question: "Can non-i.i.d., off-policy trajectory sampling lead to a faster convergence of the empirical distribution of trajectories to the expected on-policy distribution?" The answer to this question is affirmative. Methods are developed to solve this problem through adapting the behavior of the data-collecting policy, taking into account the data that has already been collected when selecting future actions. Specifically, we introduce Robust On-Policy Sampling (ROS) that is able to produce an empirical distribution of data that converges more rapidly to the expected on-policy trajectory distribution when compared to traditional on-policy sampling methods.

5.2.1.1 Problem Formulation

We formally define the problem of policy evaluation. In the policy evaluation problem, we are given a policy to be evaluated, π_e, for which we would like to estimate $v(\pi_e)$. Algorithms for policy evaluation involved two steps: colleting data and computing an estimate from that data. The data is collected by running a behavior policy that may or may not be the same as the evaluation or target policy. For on-policy evaluation problems, the behavior policy is the same as the evaluation policy. For off-policy evaluation problems, the behavior policy is different from the evaluation policy. The final performance metric is computed by a policy evaluation estimator (PE) that maps a set of trajectories to a scalar-valued estimator of $v(\pi_e)$, which is called as the average policy return (APR). Thus the goal of policy evaluation is to conduct the evaluation with low mean squared error (MSE) on APR:

$$MSE[PE] := E[(PE(D) - v(\pi_e))^2 | D \sim \pi_b], \tag{5.1}$$

where π_b is the behavior policy that is run to collect D and PE is a generic policy evaluation estimator.

One kind of common policy evaluation methods is Monte Carlo Policy Evaluation (MCPE). Before we get into specific on-policy evalaution alsogrithm, we elucidate how an estimator that leverages on-policy data can significantly benefit from the incorporation of off-policy sampling methodologies. To be more specific, we focus our attention on the Monte Carlo estimator and examine a scenario where we have already amassed a data set, denoted as D1, consisting of various trajectories. Our goal is to gather an additional set of trajectories, referred to as D2, and subsequently compute the Monte Carlo estimate utilizing the combined data set $D1 \cup D2$. It is important to highlight that D1 represents a fixed set of trajectories that have already been observed, whereas D2 is a random variable representing the trajectories that have yet to be collected. This raises the crucial question: how should we go about collecting D2 to ensure minimal MSE in our policy evaluation when applying the Monte Carlo estimator? Our analysis within this section indicates that employing independent and identically distributed (i.i.d.) sampling of trajectories according to policy π_e may not be the most optimal approach available. This insight paves the way for exploring alternative sampling methods that could enhance the accuracy and performance of the Monte Carlo estimator. In this context, the formulation of the Monte Carlo estimator using the combined data set $D1 \cup D2$ can be articulated as follows:

$$MC(D_1 \cup D_2) := \underbrace{\frac{1}{n}\sum_{i=1}^{nD_1} g(h_i)}_{\text{fixed value}} + \underbrace{\frac{1}{n}\sum_{i=1}^{nD_2} g(H_i)}_{\text{random variable}}, \tag{5.2}$$

where nD_1 and nD_2 are the number of trajectories in D_1 and D_2, repectively and $n = nD_1 + nD_2$. This estimator is referred as data-conditioned Monte Carlo estimator (DCMCE).

Viewing the Monte Carlo estimator as a sum comprising both a fixed quantity and a random quantity significantly alters our understanding of its statistical properties. This perspective allows us to better analyze the behavior of the estimator across various sampling scenarios. For instance, while the Monte Carlo estimator is well-established as being unbiased under on-policy sampling conditions, the situation becomes more complex when we consider its data-conditioned estimate. This nuance is highlighted in the following proposition.

Theory 1. The data conditioned Monte Carlo estimator is inherently biased under on-policy sampling of dataset D2, unless the condition $MC(D_1) = v(\pi_e)$ is satisfied or if D_1 is empty ($D1 = \emptyset$). This indicates a fundamental limitation in the estimator's application, as it underscores the dependence of its accuracy on the relationship between the datasets.

Notes that Theory 1 remains valid even when D_1 is obtained through on-policy sampling. In scenarios where D_1 is collected under on-policy sampling, the Monte

Carlo estimator retains its unbiased status when considering all conceivable realizations of D_1. However, it is crucial to note that once the trajectories in D1 are fixed, the potential values they could have assumed become irrelevant. This fixation alters the estimator's statistical behavior, highlighting the intricate balance between sampling methods and the resulting estimates. Therefore, understanding these dynamics is essential for accurate statistical inference within the framework of Monte Carlo methods.

We can indeed reduce the bias of the data-conditioned Monte Carlo estimator by collecting the dataset D_2 with a policy that is different from the evaluation policy π_e. To illustrate this concept, we conclude this section with an example that clearly demonstrates how such an approach can be beneficial. Consider a simple one-step Markov Decision Process (MDP) featuring only one state, denoted as s, and two possible actions, a_0 and a_1. The return for taking action a_0 is 2, while the return for action a_1 is notably higher at 4. The evaluation policy is defined such that $\pi_e(a_0|s) = \pi_e(a_1|s) = 0.5$.

Now, let's suppose that after sampling three trajectories, the dataset D_1 consists of two instances of $s, a_0, 2$ and one instance of $s, a_1, 4$. It is important to note that action a_0 is over-sampled relative to its true probability in state s, while action a_1 is under-sampled. If we proceed to collect an additional trajectory using the evaluation policy π_e, the expected value of the Monte Carlo estimate can be calculated as follows:

$$\frac{1}{4}(2 + 2 + 4 + 2\pi_e(a_0) + 4\pi_e(a_1)) = \frac{11}{4} = 2.75. \tag{5.3}$$

The actual true value, denoted as $v(\pi_e)$, is 3. Thus, conditioned on the prior data, we can see that the Monte Carlo estimate is biased in expectation, as discussed in Theory 1. However, if we were to select the behavior policy such that $\pi_b(a_1) = 1$, we would achieve a situation where neither action is over-sampled nor under-sampled. In this case, the expected value of the Monte Carlo estimate would equal the exact true value:

$$\frac{1}{4}(2 + 2 + 4 + 4) = \frac{12}{4} = 3. \tag{5.4}$$

This example effectively highlights the significance of adapting the behavior policy to take into account the previously collected data, which can substantially lower the expected finite-sample error in policy evaluation processes. In the forthcoming section, we will introduce an innovative adaptive data collection method that dynamically adjusts the behavior policy based on the data that has already been observed. This adjustment aims to minimize the mean squared error (MSE) of the Monte Carlo estimate by leveraging all available observed data. Such an approach promises to enhance the accuracy and efficiency of policy evaluation in various applications.

5.2.1.2 Robust On-Policy Data Collection

We now describe a method that adjusts the data-collecting behavior policy online to minimize sampling error in the data used by Monte Carlo estimator. This method is called as Robust On-Policy Monte Carlo Estimator (dsilver2022robust) [46], Specifically, let D_t denote all trajectories observed up to time-step t of the current trajectory, which includes the partial current trajectory as well. At time-step t, our method sets the behavior policy with the specific goal of reducing the current sampling error. This sampling error is defined as the divergence between $Pr(h|\pi_e)$ and $Pr(h|D_t)$. Our method can be initiated in two ways: starting with $D_t = \emptyset$ or beginning with some pre-existing trajectories in a setting like that described in the preceding section. To effectively reduce sampling error when collecting future trajectories, we seek to adjust the behavior policy in a manner that increases the probability of under-sampled trajectories—specifically, those h for which $Pr(h|D_t) < Pr(h|\pi_e)$. Unfortunately, we face a challenge because the trajectory distributions are inherently unknown, primarily due to the transition function, P, also remaining unknown. To overcome this limitation, we will focus on increasing the probability of under-sampled actions. Let $\pi_D : S \times A \rightarrow [0, 1]$ denote the empirical policy that represents the proportion of times each action was taken in each state within D_t. If $\pi_D(a|s) < \pi_e(a|s)$, it indicates that action a has appeared less frequently in the data than would be expected under π_e. Therefore, we should increase the probability of taking action a in state s for future data collection endeavors. When both the state and action spaces are finite, π_D can be computed exactly as the maximum likelihood policy under D_t:

$$\pi_D := \arg\max_{\pi} L(\pi), \quad L(\pi) := \sum_{h \in D_t} \sum_{t'=0}^{l-1} \log \pi(a_{t'}|s_{t'}), \tag{5.5}$$

where the argmax is taken with respect to all policies. In larger MDPs, we often require function approximation, which can complicate the computation and online updating of π_D as new data is collected. Fortunately, with an additional assumption, we can determine the direction in which to adjust action probabilities without needing to explicitly compute π_D. This assumption posits that π_e belongs to a class of differentiable, parameterized policies that are parameterized by a vector $\Theta \in \mathbb{R}^d$. This assumption is relatively mild for many reinforcement learning applications, as it allows for tabular, linear, and even neural network policy representations. We will use Θ_e to denote the parameter values associated with π_e. In the subsequent subsection, we will demonstrate that the gradient of the log-likelihood at Θ_e, specifically $\nabla_\Theta L(\pi_\Theta)|_{\Theta=\Theta_e}$, can be effectively utilized to implement changes in the behavior policy that reduce sampling error.

Robust On-policy Sampling

Robust On-Policy Sampling (ROS) reduces sampling error by adapting the behavior policy with a single step of gradient descent on the log-likelihood at each

time-step. From this point onward, we denote $\nabla_\Theta \mathcal{L}$ as the gradient of the log-likelihood evaluated at Θ_e. It is essential to observe that $\mathcal{L}$ provides a clear direction for adjusting Θ_e to enhance the probability of actions that have been over-sampled relative to their probability under the evaluation policy Θ_e. Consequently, $\mathcal{L}$ also indicates how to adjust Θ_e in order to decrease the probability of actions that have been over-sampled when $\pi_D(a|s)$ exceeds $\pi_e(a|s)$. With this valuable insight, the ROS algorithm is capable of effectively adapting π_e such that the distribution π_D closely tracks π_e, all without the need to compute π_D explicitly. At each time-step, the ROS framework computes $\mathcal{L}$ utilizing all state-action pairs that have been previously observed, and subsequently modifies the evaluation policy parameters through a single step of gradient descent. This adjustment ensures that under-sampled actions are granted a higher probability than they would otherwise have under the evaluation policy π_e.

The pseudocode for the ROS algorithm is provided in Algorithm **??**. Initially, ROS computes $\mathcal{L}$ based on previously collected trajectories, if any such data is available. Following this, ROS gathers n additional trajectories by actively interacting with the specified MDP. For each action selection, ROS sets the behavior policy parameters according to the equation $\Theta_e - \alpha \nabla_\Theta \mathcal{L}(\pi_\Theta)|_{\Theta=\Theta_e}$. It then calculates $\nabla_\Theta log \pi_\Theta(A|s)|_{\Theta=\Theta_e}$ and updates $\nabla_\Theta \mathcal{L}(\pi_\Theta)|_{\Theta=\Theta_e}$. Finally, the chosen action is executed in the environment, a reward is received, and the agent transitions to the next state. It is important to note that the process of updating $\mathcal{L}$ necessitates per-timestep computation that scales linearly with respect to the number of policy parameters, while remaining constant even as the size of the dataset D increases. This efficiency is crucial for the practical implementation of the ROS algorithm in real-world applications.

ROS Convergence

We present the theoretical analysis that underpins the efficacy of ROS. Firstly, ROS converges to the expected state visitation frequencies under π_e. Second, we demonstrate that, for a fixed state, the distribution $\pi_D(\dot{|}s)$ converges to the optimal policy distribution $\pi_e(\dot{|}s)$ at a significantly faster rate when employing the Reinforcement Learning method known as ROS, as opposed to traditional on-policy sampling methods. This accelerated convergence under ROS is a critical factor in enhancing the efficiency of policy evaluation processes. Finally, we introduce a comprehensive upper bound on the squared error between the Monte Carlo estimate and the true value function $v(\pi_e)$, expressed in terms of the sampling error. This formulation illustrates how the faster convergence achieved by ROS positively impacts the MSE of policy evaluation. These significant results are based on the following foundational assumption.

Assumption 1: The discrete state-space of the MDP is structured as a directed acyclic graph (DAG). More specifically, the states within the set S can be partitioned into l disjoint subsets S_t, each indexed by the episode step. The transition function is defined such that if $P(s'|s,a) > 0$, it must follow that $s \in S_t$ and $s' \in S_{t+1}$. This

Algorithm 13 Robust On-Policy Sampling

Input: Evaluation policy π_e parameterized with Θ_e; step size α, previously collected trajectories to be used for policy evaluation, D_1 (possibly empty), number of trajectories to collect n.
Output: Data set of trajectories.

for each episode **do**
  $k \leftarrow numberofstate - actiontuplesinD_1$
  $\nabla_\Theta \mathcal{L} \leftarrow \frac{1}{k} \sum_{(s,a)\in D_1} \nabla_\Theta log\pi_\Theta(a|s)|_{\Theta=\Theta_e}$
  $D \leftarrow D_1$
  for each trajectory **do**
    $s_0 \sim d_0$
    for each time step **do**
      $\Theta_b \leftarrow \Theta_e - \alpha \nabla_\Theta \mathcal{L}$
      $a_t \leftarrow A \sim \pi_{\Theta_b}(|s_t)$
      $\nabla_\Theta \mathcal{L} \leftarrow \frac{k}{k+1} \nabla_\Theta \mathcal{L} + \frac{1}{k+1} \nabla_\Theta log\pi_\Theta(a_t|s_t)|_{\Theta=\Theta_e}$
      $s_{t+1} \sim P(|s_t, a_t), r_t \leftarrow R(s_t, a_t)$
    end for$D \leftarrow D \cup (s_0, a_0, r_0, \ldots, s_{l-1}, a_{l-1}, r_{l-1})$
  end for
end for
Return $\mathcal{D}$

structure facilitates a clear progression of states over time, ensuring that no cycles are formed.

It is important to note that Assumption 1 is relatively mild, as any finite-horizon MDP can easily be transformed into a DAG by incorporating the current time-step into the state representation.

Assumption 2: The Reinforcement Learning algorithm, referred to as ROS, utilizes a step-size that approaches infinity ($\alpha \rightarrow \infty$). Furthermore, the behavior policy is parameterized using a softmax function, expressed mathematically as $\pi_\theta(a|s) \propto e^{\theta_{s,a}}$. Here, for every state s and action a, a specific parameter $\theta_{s,a}$ is assigned. As formally demonstrated below, this assumption guarantees that ROS consistently selects the most under-sampled action in each state, thereby maximizing exploration of less frequently visited actions.

To further clarify, we introduce the notation $d_t^\pi(s)$, which represents the probability of visiting state s at episode time t while adhering to policy π. Additionally, we define $d_t^n(s)$ as the empirical frequency of visits to state s at episode time t after observing n distinct trajectories. This notation is crucial for understanding the dynamics of our approach and the effectiveness of the sampling strategy employed within the MDP framework.

Theory 1. Under assumptions 1, 2, and ROS action selection, ${d_n}^t(s)$ converges to ${d_\pi}^t(s)$ with probability 1 for all $s \in S$ and $0 < t < l$:

$$\lim_{n\to\infty} {d_n}^t(s) = {d_\pi}^t(s), \forall s \in S, 0 \le t \le l. \tag{5.6}$$

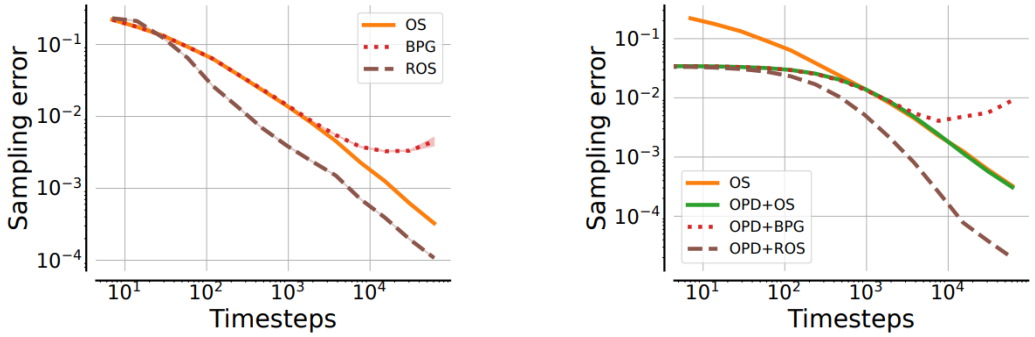
Theory 2. Let s be a specific state that is visited m times during the process of data collection. For our analysis, we will assume that the set $|\mathcal{A}|$ is greater than or equal to 2. Under Assumption 2, we find that $D_{KL}(\pi_D(\dot{|}s)||\pi(\dot{|}s)) = O_p(\frac{1}{m^2})$ when employing ROS sampling. In contrast, we observe that $D_{KL}(\pi_D(\dot{|}s)||\pi(\dot{|}s)) = O_p(\frac{1}{m})$ under on-policy sampling. Here, Op denotes the concept of stochastic boundedness, which is crucial for understanding the behavior of the distributions involved in this context. Furthermore, the theoretical findings are supported by rigorous policy evaluation experiments conducted in both finite and continuous-valued state and action space domains.

Theorem 3. Assume that s is an element of the state space $\mathcal{S}$, and a is an element of the action space $\mathcal{A}$ such that the reward $R(s, a)$ is less than or equal to $R_{\max}$. Under these conditions, the squared error in the Monte Carlo estimate using the dataset D can be upper-bounded by a specific expression that characterizes the relationship between the empirical and true distributions:

$$(v(\pi_e) - MC(\mathcal{D}))^2 \le \sum_{t=0}^{l-1} \gamma^{2t} {R_{max}}^2 \sqrt{2KL({d_n}^t||{d_{\pi_e}}^t) + 2\mathbf{E}_{s\sim {d_n}^t[KL(\pi_D(\dot{|}s)||\pi_e(\dot{|}s))]}}. \tag{5.7}$$

Remark 2. The second term in the bound presented in Theorem 3 represents the Kullback-Leibler (KL) divergence between the policy derived from the dataset D, denoted as π_D, and the optimal policy π_e. The significance of Theorem 2 is that it informs us that this KL-divergence will decrease at a faster rate when employing the ROS action selection method. In contrast, the first term in this upper-bound equation captures the KL-divergence between the empirical state distribution and the true state distribution. This divergence is influenced by two primary sources of sampling error: one arising from action selection and the other from the transition and initial state distributions.

While the former type of error tends to decline at a faster rate under ROS action selection, the latter type decreases at a similar rate for both ROS sampling and on-policy sampling approaches. Consequently, the theoretically faster rate of reduction in sampling error associated with action selection under ROS may be somewhat diminished when faced with high levels of stochasticity within the environment. Such stochastic elements can introduce variability that overshadows the benefits of the ROS strategy. The experimental results illustrated in Fig. 5.1 (a) serve to complement and support this theoretical observation, providing empirical evidence that aligns with the derived theoretical insights.



(a) Without initial data (b) With initial data

Fig. 5.1: Sampling error (KL) curves of data collection in the GridWorld domain are presented in this analysis. Each strategy employed is followed to collect data across 2^12 trillion steps, ensuring a comprehensive dataset for evaluation. All results are averaged over 200 trials to enhance statistical reliability, with shading indicating one standard error intervals to provide a visual representation of uncertainty. Figures (a) and (b) illustrate the sampling error curves of data collection without and with initial data, respectively, highlighting the differences in performance under each condition. The axes in these figures are log-scaled to facilitate better interpretation of the results and to clearly depict trends across a wide range of values [50].

5.2.1.3 Experiment Studies

In this study [46], we aim to conduct a comprehensive empirical investigation of the ROS method in the context of policy evaluation problems. Our primary objective is to address the following pertinent questions that can guide future research and practice in reinforcement learning:

- Does ROS significantly reduce sampling error when compared to traditional on-policy sampling techniques?
- Does ROS lead to a decrease in the mean squared error (MSE) of policy evaluation, particularly when starting with both off-policy data and scenarios without such data?

Experiment Settings

To explore these questions, we design a series of policy evaluation experiments across four distinct domains that encompass both discrete and continuous state and action spaces. These domains include a classic multi-armed bandit problem as outlined in [49], the well-known Gridworld scenario, as well as the CartPole and Continuous CartPole environments [50].

For the comparisons, we primarily utilize on-policy sampling (OS) of independent and identically distributed (i.i.d.) trajectories, applying the Monte Carlo estimator to compute the final policy value estimate, which we denote as OS-MC. Additionally, we conduct comparisons with the Behavior Policy Gradient (BPG) method, which

identifies a minimum variance behavior policy for the ordinary importance sampling (OIS) policy value estimator. The specifics of how the evaluation policy (denoted as π_e) and its corresponding value ($v(\pi_e)$) were determined are elaborated in writting **Pre-trained Evaluation Policy** as below, ensuring that readers have access to all relevant methodological details necessary for understanding the experimental framework.

Pre-trained Evaluation Policy

Each domain requires the creation of an evaluation policy to serve as π_e. In the three domains with a discrete action space, we utilize softmax policies of the following form:

$$\pi_\Theta(a|s) \propto \frac{e^{{\omega_a}^T \phi(s)}}{\sum_{b \in \mathcal{A}} e^{{\omega_b}^T \phi(s)}} \tag{5.8}$$

where ϕ is a one-hot encoding operator for domains characterized by a discrete state space, such as Bandit and GridWorld. For domains with a continuous state space, like CartPole, we employ a feed-forward neural network. Specifically, for the ContinuousCartPole, our formulation is expressed as:

$$\pi_\Theta(a|s) := \mathcal{N}\left(a; {\omega_\mu}^T \phi(s), {\omega_\sigma}^T \phi(s)\right)^2, \tag{5.9}$$

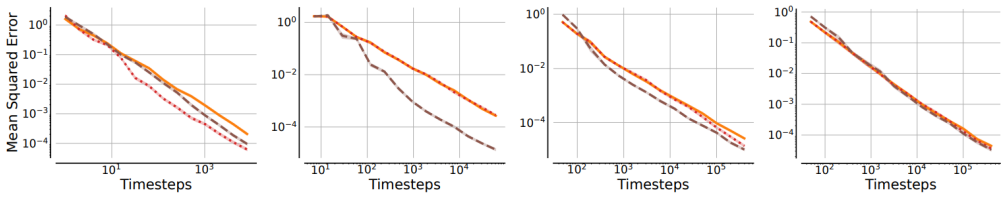
Here, ϕ is a function of the state determined by a feed-forward neural network. The policy parameters, denoted as Θ, encompass all parameters associated with the policy. For Bandit and GridWorld, Θ consists solely of the vectors ω_a, while in the cases of CartPole and ContinuousCartPole, Θ also incorporates the weights and biases of the neural network. When ϕ is defined as a neural network, it is structured to include a batch normalization layer as the first layer, succeeded by two hidden layers. Each of these hidden layers contains 64 hidden units and employs the ReLU activation function to enhance learning effectiveness. We leverage PyTorch for our neural network implementations, and utilize NumPy for performing necessary linear algebra computations.

In all domains, we apply the REINFORCE algorithm to train the policy model, and we select a policy snapshot during the training phase to act as the evaluation policy. This evaluation policy achieves returns that are greater than those produced by a uniformly random policy, although it still remains substantially far from convergence. To compute $v(\pi_e)$, we employ on-policy sampling to gather a total of 10^6 trajectories, from which we subsequently compute the Monte Carlo estimate of $v(\pi_e)$. This robust approach ensures a comprehensive evaluation of the policy's performance across diverse domains, facilitating a nuanced understanding of its effectiveness and guiding future enhancements.

Policy Evaluation without Initial Data

We first conduct a series of experiments in a setting devoid of initial data, wherein all data is gathered entirely from scratch. To facilitate our analysis, we denote T as the average length of a trajectory within each specific domain. Throughout our experiments, we accumulate a total of $2^{12}T$ environment steps using each of the methods under investigation. Moreover, we compute relevant metrics at intervals of $2^1, 2^2, \ldots, 2^{12}$ trajectories. It is important to emphasize that we specify the number of environment steps rather than the number of trajectories in our empirical results to ensure clarity in our methodology. For the Bandit domain, we find that $T = 1$; for GridWorld, $T = 7.42$; for CartPole, $T = 48.48$; and for CartPoleContinuous, $T = 49.56$. The hyper-parameter settings employed across all experiments are detailed in Appendix E for reference.

In the initial phase of our analysis, we aim to verify that the Reinforcement Learning with ROS approach effectively reduces sampling error when compared to traditional on-policy sampling methods. To quantify sampling error, we utilize the KL between the estimated policy π_e and a parametric maximum likelihood estimate of the target policy π_D derived from the observed data. A comprehensive definition of this measure, along with an alternative metric that yields qualitatively similar results. Due to constraints on space, we present this outcome exclusively for the GridWorld domain in Fig. **??**(a); however, the results obtained for other domains exhibit qualitatively similar trends and can be examined. As illustrated in Fig. **??**(a), it is evident that when employing ROS, the sampling error diminishes at a faster rate than with OS. Additionally, it is not surprising to observe that the Behavior Policy Gradient (BPG) method results in an increase in sampling error. This outcome arises because BPG is inherently an off-policy method, which adjusts the behavior policy away from π_e. Collectively, these results contribute to addressing our first empirical inquiry.



(a) Bandit (b) GridWorld (c) CartPole (d) CartPoleContinuous

Fig. 5.2: MSE of policy evaluation in the context of the 'without initial data' setting. In this scenario, policy evaluation is conducted on the data that has been collected from each strategy employed. These resulting curves effectively illustrate the MSE of the estimates, where a lower value is preferred as it indicates better performance. The vertical axis represents the MSE values, while the horizontal axis indicates the number of environment steps taken, with both axes being log-scaled for clarity and better visualization. Additionally, the shaded areas on the graph signify one standard error, providing a sense of the variability and reliability of the estimates presented [50].

These results answer our first empirical question and confirm our theoretical claim that non-i.i.d. off-policy sampling can cause the empirical distribution of data to converge to the expected on-policy distribution at a faster rate than previously observed. Ultimately, this paper focuses primarily on reducing sampling error to achieve lower MSE in policy evaluation. Fig. 5.2 clearly illustrates that the proposed method, referred to as ROS, significantly lowers MSE when compared to both the conventional OS method and the BPG across all evaluated domains. These findings not only address our second empirical question but also robustly support the claim that effectively reducing sampling error leads to a notable decrease in the MSE of the Monte Carlo estimator used for policy evaluation. This advancement has important implications for enhancing the overall accuracy and reliability of policy evaluation methods in various applications.

Policy Evaluation with Initial Data

Our next set of experiments considers a setting with initial data, wherein a comprehensive set of 100 trajectories has already been gathered. Our objective is to leverage these trajectories to enhance our policy value estimates. These trajectories were collected through independent and identically distributed (i.i.d.) off-policy sampling, utilizing a behavior policy that exhibits slight deviations from the target policy, denoted as π_e. This experimental framework is designed to mirror a scenario where π_e has recently undergone an update from a preceding policy. In such a case, it becomes essential to exploit the off-policy data that was accumulated from the older policy while simultaneously integrating new data that will be collected in the ongoing experimental phase.

In addition to the off-policy data (OPD), we will gather an extra $2 * 12T$ steps of environment interaction through each method employed. It is important to note that we do not include the initial 100 trajectories in the overall data tally. For the Reinforcement Off-policy Sampling (ROFFS) method, we will utilize the OPD to initialize the gradient $\nabla_\Theta \mathcal{L}$. Our expectation is that ROS will effectively gather new data, which, when combined with the OPD, will yield an aggregate dataset that appears as though it was originally collected using the policy π_e.

To evaluate the performance of ROS, we will compare it against several baseline methods. The first baseline, denoted as (OPD + OS)-MC, involves collecting additional data with the Off-policy Sampling method and employing the Monte Carlo estimator across the entire dataset. The second baseline, (OPD + OS)-(WIS + MC), implements WIS to derive an estimate from the OPD and integrates this WIS estimate with a Monte Carlo estimate that relies on on-policy data. The third baseline, (OPD + BPG)-OIS, collects supplementary data using the BPG method and utilizes ordinary importance sampling as the estimator across all available data.

Lastly, the (OS - MC) method substitutes the initial 100 trajectories with those gathered using OS, subsequently collecting the remainder of the data with OS and relying on the Monte Carlo estimator. In the scenario of (OS - MC), it is crucial to mention that the 100 initial trajectories will be counted towards the total data collected. By conducting these experiments, we aim to gain valuable insights into

the effectiveness of various approaches in leveraging both off-policy and on-policy data to enhance policy value estimates.

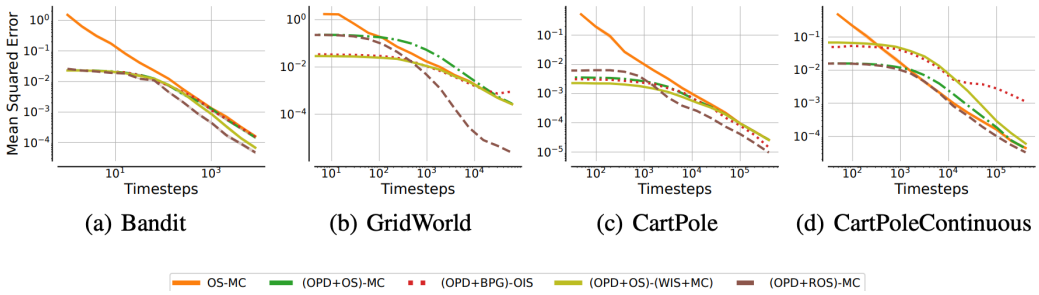


Fig. 5.3: The MSE of policy evaluation is calculated in the context of the initial data setting. In this process, policy evaluation is performed using the data that has been collected from each specific strategy, alongside a small, representative set of initial data that has been gathered off-policy. It is important to note that the axes and confidence intervals for this analysis are consistent with those presented in Fig. 5.3, ensuring comparability and clarity in the results. This approach allows for a comprehensive assessment of policy performance based on both on-policy and off-policy data [50].

Fig. 5.1(b) illustrates that the sampling error decreases most rapidly for the Reinforcement Learning with ROS method as additional data is progressively collected. To assess policy evaluation more comprehensively, we present the MSE for varying amounts of data in Fig. 5.3 and include detailed numerical values for the final MSE. Our observations reveal that the initial data tends to result in an immediate reduction in MSE; however, this comes at the cost of injecting bias into the estimates. Notably, the (OPD+OS)-Monte Carlo (MC) method struggles to mitigate this bias effectively, whereas the (OPD+ROS)-MC method can successfully address it through strategic data collection practices. Overall, this result underscores the efficacy of ROS in gathering additional data that not only reduces sampling error in the aggregate dataset but also yields lower MSE estimates when compared to other data collection approaches.

Intuitively, the Off-Policy Sampling (OS) method requires a significantly greater number of samples to dilute the bias that arises from using the Off-Policy Distribution (OPD) in the Monte Carlo estimator. In contrast, ROS possesses the capability to correct the empirical off-policy distribution, aligning it more closely with the expected on-policy distribution, thereby allowing the Monte Carlo estimator to function without necessitating any off-policy corrections. The comparison with OS-MC further highlights the potential of ROS for effectively correcting an off-policy empirical distribution to better reflect the expected on-policy distribution. It is important to note that OS-MC utilizes 100 fewer trajectories than the other baseline methods. Nevertheless, even when factoring in the initial 100 off-policy trajectories in the total data for all methods, ROS ultimately achieves a lower MSE

in comparison to OS-MC. In this regard, ROS has adeptly transformed an initially biased dataset by selectively collecting the appropriate trajectories, thereby creating the appearance that the evaluation policy had collected all trajectories from the outset.

Summary

From the experiment results, it's safe to conclude: 1. ROS effectively reduces sampling error within finite datasets, and 2. as a consequence, it significantly lowers the MSE of policy value estimates in comparison to conventional i.i.d. on-policy sampling techniques. This advancement in policy evaluation methodology has the potential to enhance the efficiency and accuracy of RL applications across various domains.

5.2.2 Off-Policy Evaluation

Off-policy evaluation (OFFPE) methods Evaluate and predict the performance of a reinforcement learning policy given historical data that may be have been generated by a different policy. The ability of off-policy evaluation is important for applications where the deployment of a bad policy can be dangerous and costly. As examples, a deployed policy for determining which advertisement to show to a user visiting a website, or for determining which medical treatment to suggest for a patient, or for suggesting a personalized curriculum for a student. In this examples, the quality of the deloped policy makes a difference in the serving quality of the RL systems and can be fatal sometimes when these systems provide services relate to the life. So, it is important to predict how well a new policy will perform without having to deploy it, as well as evaluate how existing policies perform on changing environments.

The problem of off-policy policy evaluation (OFFPE) is a significant challenge in reinforcement learning and can be defined in the following manner. We start with an evaluation policy, denoted as π_e, alongside a set of historical data, represented as D, and an approximate model of the system being studied. Our primary objective is to produce an estimator, denoted $\hat{v}(D)$, which aims to approximate the value function $v(\pi_e)$ associated with the evaluation policy. To ensure the efficacy of our estimator, we seek to minimize the mean squared error (MSE) between our estimator and the true value function. This can be mathematically expressed as $MSE(\hat{v}(D), v(\pi_e)) := E[\hat{v}(D) - v(\pi_e)]^2$. In this context, capital letters are utilized to signify random variables, meaning that all random components within expected values are consistently represented by capitalized letters (for instance, D is treated as a random variable).

Furthermore, we operate under the assumption that the underlying process generating states, actions, and rewards conforms to the structure of a Markov Decision Process (MDP). However, it is crucial to note that the initial state distribution, transition function, and reward function remain unknown. Despite this uncertainty, we assume that the evaluation policy, π_e, along with the behavior policies π_i for

i in the set $1, ..., n$, and the discount parameter γ, are known. This foundational understanding sets the stage for addressing the complexities inherent in off-policy evaluation, enabling us to explore various methodologies and algorithms that can effectively derive accurate estimates based on historical data and the specified evaluation policy. By leveraging the available information, researchers aim to enhance decision-making processes in a variety of applications, from robotics to adaptive control systems.

5.2.2.1 Doubly Robust Estimator

The doubly robust (DR) estimator [55], represents a significant advancement in the field of statistical estimation, particularly in the context of MDPs. This innovative estimator is designed to provide an unbiased estimate of the value function $v(\pi_e)$, and it has demonstrated both promising empirical and theoretical results. The DR estimator achieves this by leveraging an approximate model of an MDP, which effectively reduces the variance associated with the unbiased estimates generated through ordinary importance sampling techniques.

The term "doubly robust" refers to the estimator's ability to deliver reliable estimates under two distinct scenarios: first, if the model used in the estimation process is accurate, and second, if the behavior policies are known. The robustness of the estimator is noteworthy—if the model is inaccurate, the estimator maintains its unbiased nature, although it may exhibit high variance, leading to an increased mean squared error. Conversely, if the behavior policies are not known but the model itself is accurate, the doubly robust estimator tends to yield lower error rates. This dual reliability is one of the reasons why doubly robust estimators have gained traction in the statistical community since their introduction by Rotnitzky and Robins in 1995.

The foundational work that brought forth the DR estimator for MDPs was built upon the earlier developments of doubly robust estimators for bandit problems. This connection may explain why the original derivation of the DR estimator was confined to the finite horizon setting, where the total time horizon is predetermined and known (with L being the maximum steps taken, where L is finite). This limitation resulted in a recursive formulation of the DR estimator, which can be somewhat challenging to interpret and apply in practical scenarios.

In light of these challenges, we have approached the derivation of the DR estimator for MDPs from a fresh perspective, presenting it as an application of control variates. Our new derivation is notable for its flexibility, as it does not impose any constraints on the horizon. Furthermore, it provides a more intuitive, non-recursive definition of the estimator, expressed simply as ${w_t}^i = \frac{{\rho^i}_t}{n}$. This reinterpretation enhances the accessibility and usability of the DR estimator in various applications, paving the way for broader implementation in research and practice. By refining the understanding and application of the DR estimator, we contribute to the ongoing efforts to improve statistical methodologies in decision-making frameworks.

Assuming π_i is a set of known policies, H_i is the trajectory generated with policy π_i. Let $D := (H_i, \pi_i)_{i=1}^n$ be the experience dataset consists of generated experience

trajectories. It's allowed that $\pi_i == \pi_j$. Define the importance weight ρ_t as the ratio between the probability of the first t steps of H under the evaluation policy π_e and its probability under the behavior policy π_b:

$$\rho_t(H, \pi_e, \pi_b) := \Pi_{i=0}{}^{t} \frac{\pi_e(A_{H_i}|S_{H_i})}{\pi_b(A_{H_i}|S_{H_i})}, \tag{5.10}$$

In other words, the same known policy can generate multiple experience trajectories. The doubly robust estimator on experience dataset D is defined as:

$$DR(D) := \sum_{i=1}^{n} \sum_{t=0}^{\infty} \gamma^t \omega_t{}^i R_{tH_i} \tag{5.11}$$

$$- \sum_{i=1}^{n} \sum_{t=0}^{\infty} \gamma^t (\omega_t{}^i \hat{q}_{\pi_e}(S_{t,H_i}, A_{t,H_i} - \omega_{t-1}^i \hat{v}_{\pi_e}(S_{t,H_i}))), \tag{5.12}$$

where $\omega_t{}^i = \frac{\rho_t{}^i}{n}$ is the step-length normalized imporatnce sampling weight.

Although unbiasedness might seem like a desirable property of an estimator, when the primary goal is to minimize the MSE, it often does not hold the same significance. The MSE of an estimator, denoted as $\hat{\Theta}$, of a statistic Θ can be decomposed into its variance and its squared bias. This relationship can be expressed mathematically as follows:

$$MSE(\hat{\Theta}, \Theta) = E[(\Theta - \hat{\Theta})^2] = Var(\hat{\Theta}) + Bias(\hat{\Theta})^2, \tag{5.13}$$

where $Bias(\hat{\Theta}) := E[\hat{\Theta}] - \Theta$. In practice, the optimal estimator in terms of minimizing MSE is often one that carefully balances this bias-variance trade-off rather than simply striving for zero bias. This means that an estimator may possess some bias but can still achieve a lower overall MSE by having a significantly reduced variance.

Consequently, in the context of minimizing MSE, it becomes apparent that strong asymptotic consistency is a more desirable property than mere unbiasedness. Strong asymptotic consistency necessitates that the MSE of an estimator converges almost surely to zero as the amount of available data increases. This property ensures that as we gather more data, our estimator becomes increasingly reliable, reinforcing the idea that sometimes allowing for a small amount of bias can lead to better performance in real-world applications. In summary, while unbiasedness is an important consideration, it is the nuanced understanding of the bias-variance trade-off that ultimately guides effective estimation strategies.

5.2.2.2 Weighted Doubly Robust Estimator

The weighted doubly robust (WDR) estimator arises from the application of a straightforward yet well-established extension to importance sampling estimators, which is then integrated into the DR estimator framework. This innovative approach results in a new guided importance sampling method that enhances the traditional techniques. While this extension does not directly target the optimization of the bias-variance trade-off, it possesses a remarkable ability to better balance these competing aspects, all while ensuring that asymptotic consistency is preserved. More specifically, the WDR method relies on weighted importance sampling, which contrasts with the conventional approach of ordinary importance sampling. This distinction is crucial, as weighted importance sampling often yields improved efficiency and robustness in estimation, particularly in scenarios where data may be sparse or unevenly distributed. For those interested in a deeper understanding of the advantages of weighted importance sampling compared to ordinary importance sampling, further discussion and research are available, highlighting the myriad benefits that this approach can provide in statistical estimation and inference. Formally, WDR is defined similar to DR, except the importance sampling weight ${\omega_t}^i = \frac{{\rho_t}^i}{\sum_{j=1}^{n} {\rho_t}^j}$. By the law of large numbers the denominator of ${\omega_t}^i$ will converge to n. We can see if there is a single behavior policy π_b, the expected value of WDR shifts from $v\pi_b$ towards $v(\pi_e)$ as the number of trajectories increases.

5.2.2.3 Model and Guided Importance Sampling Combining Estimator

Model and Guided Importance Sampling Combining (MAGIC) Estimator

5.2.2.4 The ModelFail Domain

ModelFail domain was constructed so that the behavior policies fail to generate experience trajectories that converge to true MDP. One way that this can happen is if the model uses function approximation, so that it cannot represent the true MDP. Another way that this can happen is if there is some degree of partial observability present in the system, which is quite common in real-world applications across various domains and contexts. This limitation can significantly impact the accuracy and effectiveness of the outcomes.

The example MDP used by ModelFail in the experiment is in Fig 5.4. The MDP has three regular states plus the terminal absorbing state, the agent does not observe which state it is in, for instance, it only sees a single state. The agent begins its journey in the left-most state, where it has two distinct actions available to choose from. The first action consistently takes it to the upper state, while the second action always leads it to the lower state. In both scenarios, the agent receives no reward, which may impact its decision-making process. At time $t = 1$, the agent finds itself in either the upper or lower state (although it lacks the ability to differentiate between them and the

initial state), and it must make a critical choice between two possible actions. Both actions always have the same effect—the agent transitions to the terminal absorbing state. However, if the agent was in the upper state, it receives a reward of R1 = 1, while if it was in the lower state, $R1 = -1$. The horizon is set to $L = 2$, indicating a limited timeframe for decision-making. The behavior policy selects action a1 with a probability of approximately 0.88 and action a2 with a probability of approximately 0.12 (these probabilities were chosen arbitrarily by utilizing weights of 1 and -1 with softmax action selection, and were not optimized). In contrast, the evaluation policy operates in the opposite manner—it selects action a1 with a probability of approximately 0.12 and action a2 with a probability of approximately 0.88, creating a dynamic and challenging decision environment for the agent.

Consider what happens when we attempt to model this MDP based solely on the observations generated by executing the behavior policy to create an infinite number of trajectories, without attempting to infer anything about the true underlying structure of the MDP itself. It is important to recall that we can only observe a single state throughout this process. First, let us analyze the transition dynamics: in this scenario, half of the time either action taken results in a transition back to the single state, while the other half of the time, the agent transitions to the absorbing state, which effectively terminates the episode. Next, we need to examine the rewards associated with these actions: half of the time, the agent receives no reward at all. Additionally, with a probability of $0.88/2$, the agent receives a reward of 1, and with a probability of $0.12/2$, it receives a reward of -1. Notably, these rewards appear to be completely uncorrelated with the specific action that was selected, since non-zero rewards manifest at time $t = 1$ and the action A1 has no influence on either the rewards or the state transitions that follow. Thus, from the model's perspective, the actions taken have no discernible impact on the resulting state transitions or rewards. Consequently, every policy is evaluated as equally effective and will yield an expected return of 0.38, even though, in reality, an optimal policy will yield an expected return of 0.5, while a pessimal policy will produce an expected return of -0.5. To ensure the model's validity, we provided it with the true horizon, $L = 2$, which allows its predictions of R_t to be zero for any time t that is greater than or equal to 2. This setup underscores the challenges and limitations inherent in modeling MDPs when relying solely on limited observations and not incorporating the full context of the decision-making environment.

5.2.2.5 Experiments

Experiments are conducted to compare the performance of WDR estimator and several previous ones (IS, PDIS, WIS, CWPDIS, DR, and AM). The experiments are performed using three domains: ModelFail, ModelWin, and a gridworld. All three domains have a finite horizon and use $\gamma = 1.0$. We first describe each domain, then describe the experimental setup, and finally present empirical results.

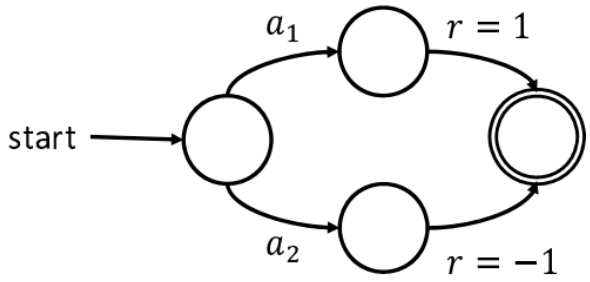


Fig. 5.4: Example ModelFail MDP [55].

The ModelWin Domain

This domain was carefully constructed so that the approximate model of the MDP would quickly converge to the true MDP, while importance sampling-based approaches like DR and WDR would continue to exhibit high variance. Recall from our comprehensive discussion in Section 6 that both DR and WDR will effectively reduce to a simple model-based approach if the approximate MDP is perfect and both state transitions and rewards are deterministic. To avoid this simplification, the ModelWin domain includes stochastic state transitions that ensure the control variable used by DR and WDR does not perfectly cancel with the PDIS term.

The ModelWin MDP is visually depicted in Fig. 5.5. Unlike the ModelFail domain, here the agent has access to the true underlying states of the ModelWin MDP, which consist of three distinct states plus an additional terminal absorbing state that is not pictured. The agent consistently begins its journey in state $s1$, where it must make a decision between two possible actions. The first action, labeled a1, allows the agent to transition to state s2 with a probability of 0.4 and to state s3 with a probability of 0.6. Conversely, the second action, a2, produces the opposite effect: transitioning to state s2 with a probability of 0.6 and to state s3 with a probability of 0.4. If the agent successfully transitions to state $s2$, it receives a reward of 1, while a transition to state $s3$ yields a reward of -1. In states $s2$ and $s3$, the agent has two available actions, but both actions always result in a reward of zero and a deterministic transition back to the initial state $s1$. The horizon for this model is set to $L = 20$, ensuring that $S20$ remains infinite, thus presenting a unique challenge for the agent to navigate through its decision-making process.

The behavior and evaluation policies both select actions uniformly at random in states $s2$ and $s3$. However, in state $s1$, the behavior policy exhibits a preference for action $a1$, taking it with a probability of approximately 0.73, while action $a2$ is chosen with a lower probability of approximately 0.27. In contrast, the evaluation policy operates inversely; it selects action $a1$ with a probability of approximately 0.27 and action $a2$ with a significantly higher probability of about 0.73. These specific probabilities arise from employing softmax action selection, which utilizes weights of 1 for action $a1$ and 0 for action $a2$. This weight assignment effectively skews the selection process. As seen in the ModelFail domain, a similar approach

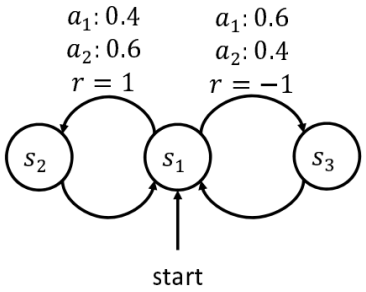


Fig. 5.5: Example ModelWin MDP [55].

was taken in the ModelWin domain. Here, we provided the approximate model with the true horizon of the MDP, set at $L = 20$. This ensures that the model's predictions of the reward function, R_t, are zero for all time steps t that are greater than or equal to 20, effectively aligning the model's understanding of future rewards with the actual structure of the MDP. This careful consideration of probabilities and horizons allows for more accurate modeling and evaluation of decision-making processes.

The Gridworld Domain

The third domain that the experiments utilized was the gridworld domain, which was specifically developed for the purpose of evaluating OFFPE algorithms. This gridworld consists of a $4x4$ layout, featuring four distinct actions available to the agent, and operates under a fixed horizon of $L = 100$. The environment is characterized by deterministic transition and reward functions, making it a controlled setting for assessing various OFFPE methods. The original research on this gridworld proposed five distinct policies that can effectively serve as both behavior and evaluation policies. While the gridworld was designed for robust OFFPE evaluations, it was not originally intended to accommodate newer methodologies like Direct Reinforcement (DR) and Weighted Direct Reinforcement (WDR), which were introduced in subsequent studies. This limitation arises from the deterministic nature of the state-transition and reward functions; when the model is accurately specified, AM, DR, and WDR will exhibit similar performance levels.

To address these limitations, experiments were conducted with two variants of the gridworld. In the first variant, the approximate model was provided with the correct horizon, $L = 100$. Conversely, the second variant introduced a level of partial observability by supplying the model with an incorrect horizon of $L = 101$. This adjustment has a considerable impact on value predictions, particularly as the agent approaches the end of a trajectory, as the model inaccurately forecasts when rewards will inevitably diminish to zero. To differentiate these setups, we refer to them as Gridworld-TH and Gridworld-FH, where the former denotes the gridworld in which the agent is given the true horizon and the latter indicates the scenario with the false horizon. The implications of these two configurations are significant, as

they allow for a comprehensive analysis of how variations in horizon perception can affect the performance of OFFPE methods in practical applications.

Experiment Settings

For each domain, we generated n trajectories (for various values of n) and computed the sample mean squared error between the predictions of the various Off-Policy Evaluation (OFFPE) methods and the true performance of the evaluation policy. This true performance was estimated using a large number of on-policy Monte-Carlo rollouts, ensuring a robust comparison. For every value of n and each OFFPE algorithm, we conducted this experiment 128 times. We then reported the average sample mean squared error over these 128 trials to provide a comprehensive view of the algorithms' performance. All resulting plots include standard error bars to illustrate the variability of the results, and we employed logarithmic scales for both the horizontal and vertical axes to enhance clarity and interpretation of the data.

Perhaps surprisingly, determining a fair comparison between the different OFFPE algorithms proves to be challenging. Clearly, Importance Sampling (IS), Per-Decision Importance Sampling (PDIS), Weighted Importance Sampling (WIS), and Conditional Weighted Per-Decision Importance Sampling (CWPDIS) should utilize all of the trajectories in dataset D, as they do not necessitate an approximate model. The definitions of these four importance sampling can be found in [50]. Similarly, the Approximate Model (AM) method should leverage the entirety of the data available to construct its approximate model effectively. However, a pertinent question arises: how should the available data be appropriately partitioned for Double Robust (DR), Weighted Double Robust (WDR), and the MAGIC estimators? We propose at least three reasonable approaches to address this issue:

- DR, WDR, and MAGIC should be provided with additional trajectories that are not accessible to IS, PDIS, WIS, and CWPDIS; these extra trajectories should be employed to construct an approximate model. This setup would closely emulate a scenario where prior domain knowledge, which may not necessarily consist of trajectories, can be harnessed to build an effective approximate model, something that IS, PDIS, WIS, and CWPDIS do not take into account.
- An alternative approach is that DR, WDR, and MAGIC should utilize the complete dataset D to construct an approximate model. They would subsequently reuse this same dataset to compute their estimates. While this method is reasonable, it is important to note that the reuse of data may invalidate our theoretical guarantees. Nevertheless, we have empirically observed that this strategy enables DR, WDR, and MAGIC to perform at their optimal levels.
- Lastly, we suggest that DR, WDR, and MAGIC should partition the dataset D into two distinct sets. The first subset should be designated for constructing the approximate model, while the second subset is reserved for computing the DR, WDR, and MAGIC estimates using the previously constructed approximate model. This partitioning could ensure that the estimates remain unbiased and

provide a fair evaluation of the different OFFPE methods, allowing us to draw more reliable conclusions from our experiments.

Since there is not necessarily a singularly "correct" answer to the question of which method of performing experiments is optimal, we have chosen to present our results using both the second and third approaches to provide a comprehensive view. For each domain considered, the "full-data" variant employs the second approach, while the "half-data" variant utilizes the third approach, wherein the dataset D is partitioned into two sets of equal size for analysis. Given that all of the domains we focus on possess finite state and action sets, we apply a straightforward maximum-likelihood approximate model to our evaluations. Specifically, we predict that the probability of transitioning from state s to state s', given action a, is calculated by taking the number of times this particular transition was observed and dividing it by the total number of times action a was executed in state s. In situations where D contains no recorded instances of action a being taken in state s, we adopt the assumption that taking action a in state s invariably leads to a transition into the terminal absorbing state.

Experiment Results

We present empirical results drawn from four previously established importance sampling methods: IS, per-decision importance sampling (PDIS), WIS, and consistent weighted per-decision importance sampling (CWPDIS). Additionally, we provide results for the guided importance sampling methods, specifically DR and WDR, as well as the purely model-based method, AM. The legend utilized by all of the plots included in this appendix is clearly provided in Fig. 5.6, ensuring that readers can easily interpret the results presented.

Fig. 5.6: The legend used by all plots of experiment results [55].

ModelFail Results In Fig. 5.7, we reproduce this experiment in a full-data setting, which allows for a comprehensive evaluation of the various importance sampling methods employed. Within this context, the weighted importance sampling methods, specifically WIS and Control Weighted Partial Data Importance Sampling (CWPDIS), appear to be obscured by the curve representing the performance of Weighted Doubly Robust (WDR) estimators. Conversely, the unweighted importance sampling methods, including IS and Partial Data Importance Sampling (PDIS), are similarly obscured by the curve associated with the DR estimators. A notable observation is that WDR significantly outperforms the Augmented Model (AM) by orders of magnitude, while it surpasses the performance of the DR method by approximately an order of magnitude as well.

Additionally, it is important to note that despite the inaccuracy of the approximate model utilized in this setting, which implies that the control variates employed by both DR and WDR may not be optimal, the performance of the DR and WDR estimators does not diminish below that of PDIS and CWPDIS, respectively. This suggests a robustness in their performance even with less-than-ideal models.

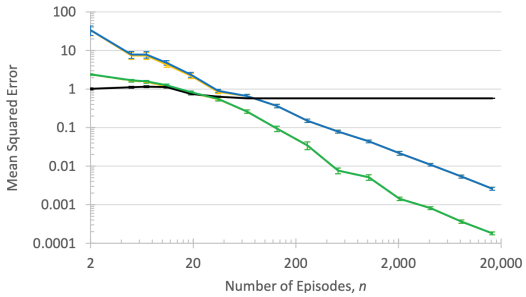


Fig. 5.7: ModelFail, full-data [55].

Moving on to Fig. 5.8, we replicate this experiment in a half-data setting, which introduces a different dynamic. Since the AM method does not leverage any data for importance sampling, its performance remains identical in both the half-data and full-data scenarios. Similarly, IS, PDIS, WIS, and CWPDIS do not utilize an approximate model, thus relying on the full dataset and maintaining consistent performance across both experimental settings. However, the situation differs for DR and WDR, as these methods utilize half of the available data to construct their approximate model while employing the remaining half to compute their respective estimates. Consequently, this division of data results in a potentially inferior approximate model for both DR and WDR, which in turn leads to a slight upward shift in their performance curves. Nevertheless, the overarching trends remain clear: WDR continues to outperform AM by significant margins, and DR does the same when compared to AM, albeit by a smaller order. This consistency in performance across different data settings highlights the reliability of the WDR approach in various scenarios.

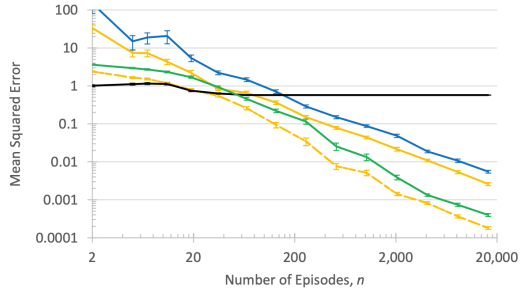


Fig. 5.8: ModelFail, half-data [55].

Fig. **??** provides a comprehensive depiction of the results obtained from implementing both importance sampling and guided importance sampling methods, alongside the approximate model estimator, within the context of the ModelWin experimental setup under a full-data scenario. Notably, the Adaptive Method (AM) demonstrates an approximately an order of magnitude lower MSE compared to all other methodologies examined, including the Weighted Dynamic Regression (WDR) approach. This significant performance advantage served as a key motivator for our innovative strategy of combining AM with WDR through the use of Bayesian Inference Methods (BIM).

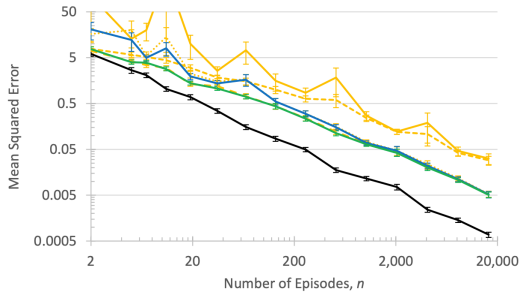


Fig. 5.9: ModelWin, full-data [55].

In Fig. **??**, we replicate this experimental analysis in a half-data setting, which offers additional insights into the performance dynamics of the methods. Consistent with our findings from the ModelWin setup, we observe that the reduction in available data adversely affects the performance of the Doubly Robust (DR) and WDR methods. Particularly, when the number of trajectories is limited, the impact seems to be more pronounced on the DR methodology compared to WDR. However, it is essential to consider that this observation may be influenced by the presence of noise, as evidenced by the substantial standard error bars associated with the DR curve when the sample size (n) is small.

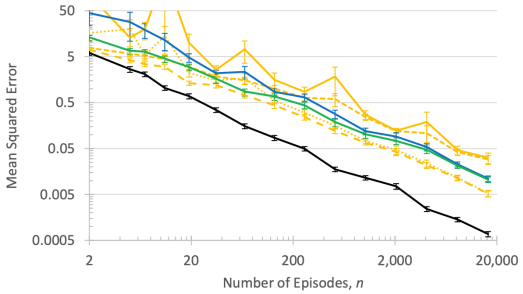
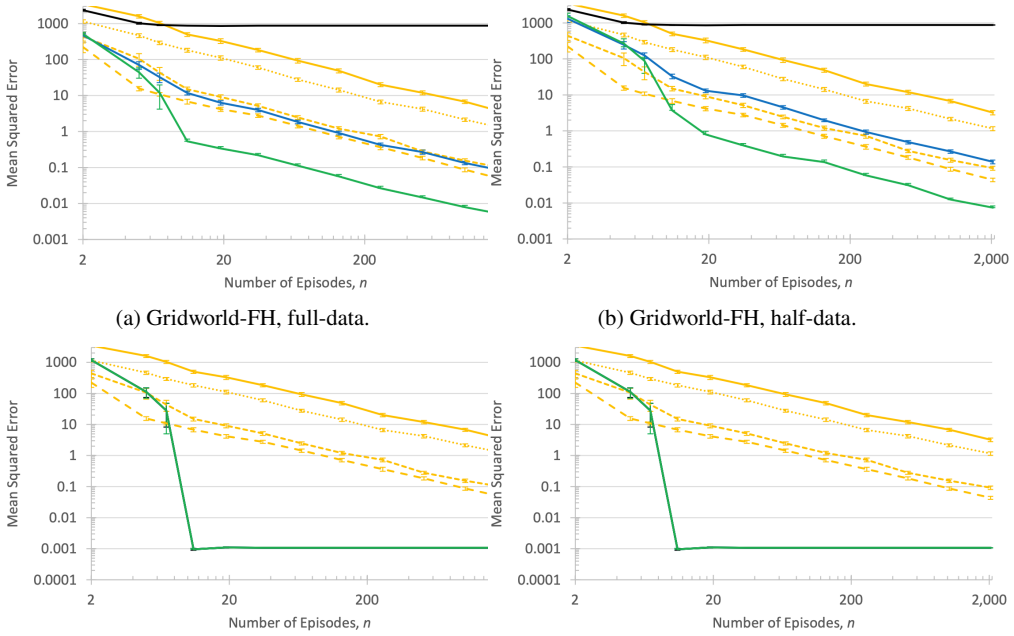


Fig. 5.10: ModelWin, half-data [55].

Fig. 5.11a provides a detailed depiction of the results obtained from utilizing the fourth gridworld policy, denoted as π_4, functioning as the behavior policy, alongside

the fifth policy, π_5, serving as the evaluation policy for the Gridworld-FH domain under a full-data setting. It is noteworthy that the WDR method significantly outperforms all other competing methods by at least an order of magnitude, showcasing its effectiveness and reliability in this context. In Fig. 5.11b, we replicate this experiment within a half-data setting. As anticipated, there is minimal change observed, with the exception that both the DR and WDR curves exhibit an upward shift. Remarkably, WDR continues to be the top-performing estimator, maintaining its superiority by approximately an order of magnitude over others.

Subsequently, we performed a reproduction of Fig. 5.11a and Fig. 5.11b, this time focusing on the Gridworld-TH scenario as opposed to Gridworld-FH. The results are illustrated in Fig. 5.11d and Fig. 5.11d, respectively. It is essential to highlight that when the true horizon is accurately provided, the Adaptive Method (AM) demonstrates exceptional performance. In the full-data setting, both DR and WDR align closely with the curve for AM, which is logical given that the transition function and reward function in this context are deterministic. Consequently, the way we constructed our approximate model leads both methods to converge precisely to AM. However, in the half-data setting, DR and WDR lag slightly behind AM's curve, primarily due to the limitation of utilizing only half as much data.



(a) Gridworld-FH, full-data.

(b) Gridworld-FH, half-data.

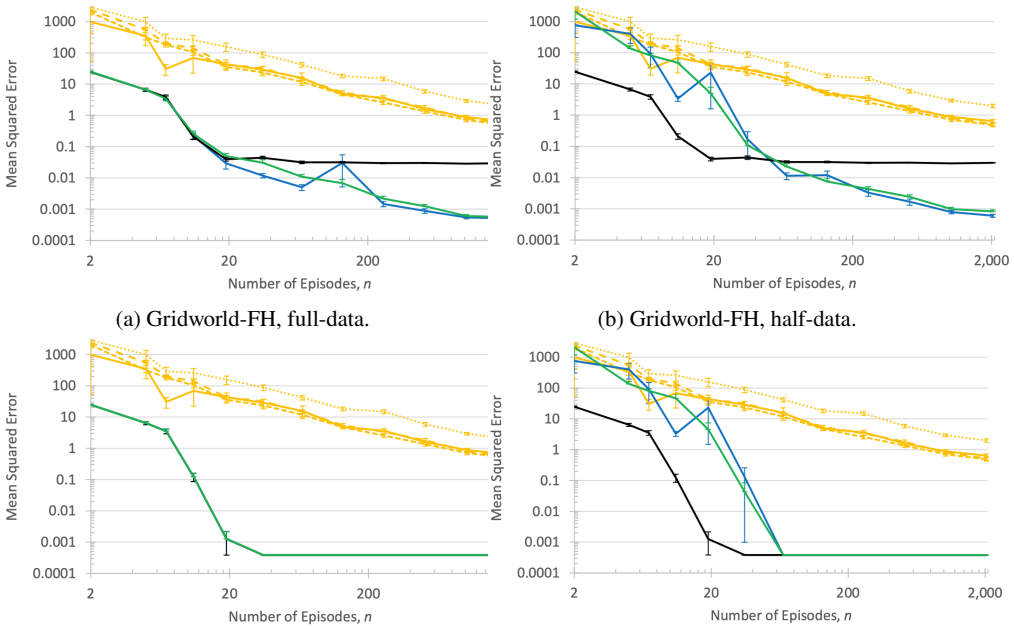
(c) Gridworld-TH, full-data.

(d) Gridworld-TH, half-data.

Fig. 5.11: The Gridworld Domain; π_4 behavior policy, π_5 evaluation policy

Furthermore, we replicated these four figures employing the first gridworld policy, π_1, as the behavior policy and the second, π_2, as the evaluation policy. In contrast to the deterministic nature of π_4 and π_5, which generate longer trajectories, π_1 and π_2 are significantly less deterministic and tend to produce shorter trajectories. Notably,

the behavior policy, π_1, selects actions in a uniformly random manner, creating a distinctly different setting for OFFPE. The outcomes from this alternative configuration are presented in Fig. 5.12. In this particular example, both DR and WDR perform comparably well, showing significant improvement over the importance sampling algorithms such as IS, PDIS, WIS, and CWPDIS, and achieving marginally better results than AM when sufficient data is available. Additionally, when provided with the true horizon, both DR and WDR again converge towards the performance of AM, underscoring the robustness of these methods in various data scenarios.



(a) Gridworld-FH, full-data.

(b) Gridworld-FH, half-data.

(c) Gridworld-TH, full-data.

(d) Gridworld-TH, half-data.

Fig. 5.12: The Gridworld Domain; π_1 behavior policy, π_2 evaluation policy

The key takeaways from these comprehensive experiments indicate that the WDR method tends to outperform other importance sampling estimators, including IS, PDIS, WIS, and CWPDIS, as well as the guided importance sampling technique known as DR. Notably, none of these competing methods managed to achieve mean squared errors that were within an order of magnitude of WDR's performance across all of our diverse experimental conditions. This significant disparity highlights the robustness and effectiveness of WDR as a guided importance sampling method. However, it is crucial to note that WDR did not consistently emerge as the superior approach; in the ModelFail setting, for instance, Adaptive Method (AM) outperformed WDR by a considerable order of magnitude. Similar observations have been documented in previous studies. For instance, in the experiments conducted, AM consistently outperformed DR, although it is worth mentioning that they did not compare their results to WDR since it had not yet been introduced at that time. This gap in the literature motivated our introduction of the Blended Importance Sampling

(BIM) estimator, designed specifically to combine the strengths of both WDR and AM. It is also important to recognize that if the transition function and reward function are deterministic and there is an absence of partial observability—such as in the gridworld experiments utilizing the true horizon—then, based on the construction of our approximate model, both DR and WDR effectively degenerate to AM. This degeneration is not inherently detrimental; rather, it suggests that importance sampling methods may not be necessary under certain conditions. However, this degeneration would not take place if the approximate model employed function approximation techniques. Lastly, it is noteworthy that DR and WDR exhibited superior performance in the full-data setting compared to the half-data setting. This finding implies that, in practical applications, one should utilize all available data to create an approximate model and to compute the DR and WDR estimates. Although this approach may violate the assumptions underlying our theoretical guarantees, it does not imply that methods like MAGIC will not remain strongly consistent estimators for the given application context.

5.2.3 Simulation-Based Evaluation

Simulation-based evaluation is an essential technique in the field of RL that involves the creation and interaction with simulated environments to comprehensively assess the performance of RL agents. This innovative approach provides several significant advantages over direct evaluation methods conducted in real-world settings. These advantages include:

- Safety: By utilizing simulations, researchers can avoid potential risks or damages that could arise during real-world experiments, ensuring the well-being of both the agents and the environment.
- Control: Simulated environments allow for precise manipulation of parameters and conditions, enabling researchers to systematically study their effects on agent performance and decision-making processes.
- Efficiency: Simulation-based evaluation accelerates the experimentation process, allowing for rapid data gathering and analysis, which is invaluable in optimizing and refining RL algorithms.
- Flexibility: This approach facilitates the exploration of a wide range of scenarios and counterfactuals, enabling researchers to test the robustness of their agents in various hypothetical situations and improve their adaptability.

Overall, simulation-based evaluation is a powerful tool that enhances the development and understanding of reinforcement learning algorithms, paving the way for more effective and reliable AI systems. For instance, simulation-based evaluation can be effectively utilized to assess agents that have been trained through the process of transfer learning. This method involves leveraging the knowledge acquired from training on an offline dataset, which is instrumental in enhancing performance in a new, yet similar, environment. In many situations, particularly when experimentation

could be costly or impractical, real environments are often not accessible for direct testing.

The main components of simulation-based evaluation encompass several critical elements, including evaluation metrics, environment simulation, and agent interaction. Environment simulation typically consists of two essential processes: model construction and environment parameterization. Model construction involves designing a detailed digital representation of the real-world environment, capturing its dynamics, states, actions, and associated rewards. In some cases, model construction may be optional if evaluation models can be directly derived from real physical models or training models related to the RL problems being investigated. On the other hand, environment parameterization is responsible for establishing the properties and rules that govern the environment. One of the primary algorithm properties that must be rigorously tested is its robustness to environment initialization. If the performance of the agent is significantly influenced by the specific state in which the simulation begins, it is generally safe to conclude that the learning algorithm lacks robustness. This indicates that the agent's ability to generalize its learning across various scenarios may be limited, which is a crucial aspect to address in order to enhance the effectiveness and reliability of the simulation-based evaluation process. Agent interaction involves several critical components, including policy execution, environment updates, and the collection of experiences for evaluation purposes. The RL agent's policy plays a pivotal role, as it determines the actions the agent takes based on the current state of the environment. Once the agent performs its actions, these actions lead to transitions between states and generate reward signals that indicate the success or failure of those actions. To facilitate effective learning, the agent meticulously stores its interactions—comprising state, action, reward, and next state—allowing for thorough evaluation. This evaluation process is typically conducted at regular intervals to ensure the agent's learning and performance are continuously assessed and improved.

Simulation-based evaluation methods frequently use synthetic environments in environemnt simulation to test the learned policies or trained agents. Synthetic environments develop and create highly advanced simulated environments that closely mimic the complexities and nuances of the real-world environment. These meticulously crafted simulations enable a comprehensive evaluation of the agent's performance and behavior, providing invaluable insights into how well it functions under various conditions and scenarios, ultimately enhancing its effectiveness and adaptability. There are multiple RL softwares that provide synthetic environments for different purposes that can be used as synthetic environments for RL performance evaluation. Android Env [57] and MuJoCo [56] are widely utilized for simulating physical systems and robotic tasks. OpenAI [9] and TFAgents [16] provide the capability to develop cuatomized environments as well as sophisticated implementation of families of popular environments.

5.2.4 Benchmarking

Benchmarking is a critical component of RL evaluation, providing a standardized framework to compare the performance of different algorithms and agents effectively. By establishing common evaluation criteria and utilizing diverse datasets, benchmarking enables researchers and practitioners to systematically assess the progress of RL research over time and identify specific areas for improvement, ensuring that advancements in the field are both measurable and reproducible. This process is essential for fostering innovation and driving further developments in RL applications across various domains.

The key benefits of Benchmarking include standardized environment, standardized evaluation metrics, baseline algorithms which are usually available in mainstream RL softwares, and flexible environment setup which is provided by sophisticated RL softwares. These benefits make it comparatively easy for performance comparison with existing popular algorithms.

Multiple popular benchmark suites are available

It's worth to note that due to the current status of RL development, environments available in these suites are usually simple environments or simplified models of real-world environments.

- OpenAI Gym Benchmarking: A comprehensive collection of standardized benchmarks designed specifically for evaluating performance across a variety of reinforcement learning tasks. This framework facilitates comparison and aids researchers in developing more effective RL algorithms.
- DeepMind Control Suite Benchmark: An advanced benchmark tailored for continuous control tasks, providing a range of challenging environments that enable the assessment of algorithmic performance in dynamic settings.
- Roboschool: A sophisticated physics-based simulator developed for benchmarking robotic control algorithms, offering realistic environments that allow for rigorous testing and evaluation of robotic learning approaches.
  Meta-World: A specialized benchmark for meta-reinforcement learning, emphasizing the importance of generalization and adaptation across different tasks, which is crucial for developing intelligent agents capable of learning efficiently in diverse scenarios.

5.3 Online Performance Evaluation

Online performance evaluation of RL systems plays a crucial role in assessing an agent's performance as it interacts with its environment in real-time. This dynamic evaluation approach offers valuable insights into the agent's behavior, learning progress, and overall effectiveness in achieving designated goals. Key metrics utilized in this evaluation process include reward, success rate, episode length, and learning curve. Each of these metrics serves to quantify the agent's ability to learn and adapt over time.

Several techniques for online evaluation are frequently employed, such as real-time monitoring, A/B testing, and human evaluation. Real-time monitoring allows for the continuous tracking of an agent's performance during training, enabling adjustments to the learning process as necessary. A/B testing serves to compare the real-time performance of different RL agents or algorithms, helping to identify which approaches yield superior results. Human evaluation, on the other hand, involves expert assessments of the agent's performance, providing qualitative feedback that can enhance the learning process.

However, online performance evaluation of RL systems comes with its own set of unique challenges. The exploration-exploitation trade-off is a prominent concern; RL agents must balance the need to explore new actions with the desire to exploit known successful actions. While online evaluation can shed light on how effectively an agent manages this balance, ensuring that the exploration-exploitation dynamics align between training and testing phases is paramount for obtaining an accurate assessment.

Moreover, online evaluation facilitates immediate feedback on the agent's performance, allowing for timely adjustments to the learning process. However, this responsiveness often incurs the drawback of delayed agent behaviors in the primary procedure, which can complicate the evaluation process. Lastly, it is important to note that online evaluation can be computationally expensive, particularly in complex environments or large-scale RL problems, potentially limiting its practicality in certain scenarios. Balancing these various factors is essential for optimizing the evaluation process and ensuring effective training outcomes.

5.3.1 Real-time Monitoring

Real-time monitoring is a critical aspect of RL that involves continuously tracking and analyzing an agent's performance as it interacts with the environment. This enables timely adjustments to the learning process and helps to identify potential issues early on.

Generally, there are five key components of real-time monitoring: performance metrics, visualization, logging, alerts, and debugging tools. Performance metrics track key metrics such as reward, success rate, episode length, and learning curve to assess the agent's progress. Visualization uses visualizations like plots, graphs, and animations to provide a clear and intuitive understanding of the agent's behavior. Logging records relevant data points for later analysis and debugging. Alerts sets up alerts to notify you of significant events or deviations from expected behavior. Debugging tools utilize debugging tools to inspect the agent's state, actions, and rewards at different points in time.

Common techniques for real-time monitoring include tensorboard, custom dashboards, logging frameworks, and remote access. TensorBoard is a popular visualization tool for tracking various metrics and visualizing neural network models. Custom dashboards create custom dashboards using libraries like Plotly or Mat-

plotlib to visualize specific metrics and visualizations. Logging frameworks use logging frameworks like Python's built-in logging module or specialized RL logging libraries to record data for later analysis. Remote access sets up remote access to the training environment to monitor the agent's performance from anywhere.

There are multiple benefits of using real-time monitoring as the online performance evaluation method. Early Detection of Issues: Identify problems such as divergence, instability, or suboptimal performance before they become significant. Improved Learning Efficiency: Make adjustments to the learning process based on real-time feedback, leading to faster convergence and better performance. Enhanced Understanding: Gain a deeper understanding of the agent's behavior and decision-making process. Facilitated Debugging: Use monitoring tools to pinpoint and resolve issues more efficiently.

By effectively implementing real-time monitoring, you can gain valuable insights into your RL agent's performance, identify and address issues promptly, and ultimately improve the overall learning process. To implement an effective real-time monitoring module, we must choose relevant metrics. That is to select metrics that are most informative for your specific reinforcement learning task but not all general metrics. Furthermore, visionalization should be designed properly so that the metric plots and drawings are easy to understand and interpret. Alerts must be configured for critical events or deviations from expected behavior, so that abnormal performance can be notified and diagnized in a timely manner. Important data must be logged informatively, so that they can be used for debugging and analysis. More importantly, the logged data can be informative feedback and be used for performance improvements. Finally, debugging tools, which usually enable human interruption and investigation to some extent, can be useful to inspect the real-time performance of the system. Thus, it's useful helpful to leverage debugging tools to inspect the agent's state and behavior.

5.3.2 A/B Testing

A/B testing is a powerful technique used to compare the performance of different reinforcement learning agents or algorithms. By simultaneously running multiple agents on the same environment and evaluating their performance, A/B testing can help you identify the most effective approach for a given task.

Generally, there are five key steps in A/B testing for reinforcement learning: define the experiment, set up the environment, run the agents, collect data, and analyze results. A/B testing starts with defining the experiment. It clearly specifies the goal of the experiment, the RL algorithms or agents to be compared, and the metrics that will be used to evaluate performance. Next, we need to set Up the Environment. That is to create a controlled environment where both agents can interact and learn. This environment should be consistent to ensure a fair comparison. Then, we can run the agents to allow both agents to train and interact with the environment for a specified duration. During or after the testing run, we gather data on the performance of each

agent, including metrics like reward, success rate, episode length, and learning curve. Analysis of results is carried out to compare the performance of the agents based on the collected data. It uses statistical analysis to determine if the differences are significant. Finally, based on the results, we iterate on the experiment by modifying the agents or algorithms and repeating the process.

With A/B testing, we can accurately identify the most effective RL algorithm or agent for a given task. This improved performance may not available for offline performance evaluation because of its limited access to online data that serve customers in a live way. Besides performance evaluation, A/B testing reduces the risk of deploying an underperforming agent by evaluating multiple options. It enables informed decision making about RL agent design and implementation based on empirical evidence. Iterative A/B testing can also accelerate the development process by providing insights into what works and what doesn't. By effectively implementing A/B testing, you can gain valuable insights into the performance of your RL agents and make data-driven decisions to optimize their development and deployment.

Obtaining effective performance evaluation from A/B testing faces several challenges and considerations. Firstly, Ensure that the differences in performance observed between the agents are statistically significant. We can use hypothesis testing or confidence intervals to assess significance. It's also important to determine the appropriate duration for the experiment based on the complexity of the task and the desired level of confidence in the results. Furthermore, A/B testing must be run long enough to collect sufficient data to ensure reliable and accurate comparisons. Randomization usually help ensure the effectiveness of performance evaluation. The experiments randomly assign agents to different environments or starting conditions to minimize bias. Control on influential variables can make a difference in metric quality. A/B testing should set up proper control for variables that could affect performance, such as random seeds or hardware differences. Finally, if the agents are being evaluated in a real-world setting, the design and implementation usually need take into account related ethical implications and potential risks .

5.3.3 Testing Interleaving

Testing Interleaving in reinforcement learning involves evaluating the effectiveness of this technique in improving the agent's performance and generalization. Some of testing interleaving techniques overlap with those of A/B testing, but we can see that the main goals of the two online performance evaluation methods are very different. We list four common approaches to testing interleaving.

- Direct Comparison
  - Control Group: Train a baseline agent without interleaving on the same tasks or environments.
  - Experimental Group: Train an agent with interleaving on the same tasks or environments.

  - Performance Metrics: Compare the performance of both agents using relevant metrics such as reward, success rate, episode length, and learning curve.

- Generalization Testing
  - Novel Tasks: Evaluate the agent's ability to perform well on new tasks that were not seen during training.
  - Transfer Learning: Assess how well the agent can transfer knowledge from previously learned tasks to new tasks.
  - Domain Randomization: Test the agent's robustness to variations in the environment by randomly changing task parameters or conditions.
- Ablation Studies
  - Varying Interleaving Frequency: Experiment with different frequencies of task switching to determine the optimal interleaving rate.
  - Task Similarity: Evaluate the impact of task similarity on interleaving effectiveness.
  - Curriculum Learning: Compare the performance of interleaving with curriculum learning, where tasks are gradually introduced in increasing difficulty.
- Visualization
  - Learning Curves: Plot learning curves for both the interleaved and non-interleaved agents to compare their convergence rates.
  - Policy Visualization: Visualize the agent's learned policies to understand how interleaving affects their structure and generalization.
  - Task Representation: Analyze how interleaving affects the agent's representation of different tasks.

To design and implement Testing Interleaving effectively and efficiently, several considerations should be taken into account: statistical significant, experimental design, data collection, and ethical considerations. Firstly, we need to ensure that the observed differences in performance between the interleaved and non-interleaved agents are statistically significant. Hypothesis testing and confidence intervals are frequently used to assess significance. The experiments should be carefully designed to control for variables that could affect performance, such as random seeds or hardware differences. Furthermore, data sufficient is also important to ensure reliable and accurate comparisons. Finally, ethical considerations may become critical if the agents are being evaluated in a real-world setting. In such cases, it's usually beneficial to consider ethical implications and potential risks especially for large-scale or long-term testings.

By conducting thorough testing and analysis, we can gain valuable insights into the effectiveness of interleaving in reinforcement learning applications and make informed decisions about its implementation.

5.4 Performance Emulator

Performance simulator or emulator is frequently used for performance evaluation. These simulator and emulator mimic the real environments and simulate the behavior of a reinforcement learning agent in various environments, allowing for the evaluation of its performance without the need for real-world interaction. This is particularly useful when the real-world environment is complex, expensive, or dangerous to interact with. Different from offline performance evaluation methods which may also generate trajectories offline for testing or evaluation purposes, the design and implementation of performance emulators often need more effort and they are used for performance evealuation in an online manner.

The key components of a performance emulator include environment model, agent simulator, and performance metrics. An environment model is a representation of the real-world environment, including its dynamics, states, actions, and rewards. The agent simulator is a system that can execute the RL agent's policy within the simulated environment. The performance metrics are a set of metrics to evaluate the agent's performance, such as reward, success rate, episode length, and learning curve.

Generally, there are three types of performance emulators: physical-based emulators, rule-based emulators, and data-driven emulators. Physics-based emulators use physics engines to simulate the dynamics of real-world objects and environments. Rule-based emulators use predefined rules or models to simulate the behavior of the environment. Data-driven emulators use historical data or machine learning techniques to learn the dynamics of the environment.

The benefits of using a performance emulator over traditional offline and online performance evaluation methods include cost-effective, safety, controllablity, and scalability. Firstly, emulators can significantly reduce the cost of experimentation by eliminating the need for physical hardware or real-world resources. And emulators can be used to test agents in dangerous or risky scenarios without putting humans or equipment at risk. Furthermore, emulators allow for precise control over the environment, making it easier to isolate variables and conduct controlled experiments. Finally, emulators can be scaled to simulate large-scale environments or multiple agents simultaneously. This is usually infeasible if offline the performance evaluation method is used, and it's cost inefficient to use the online performance evaluation method in such cases due to the large resource consumption.

While performance emulators are applied in different application areas of reinforcement learning, they are used more frequently in areas where the accurate simulation of the reinforcement learnign system is possible and much more cost-efficient thatn running the real-world systems. Example of such areas include Robot, Autonomous Vehicle, Game AI, and Healthcare. In robot industry, robot simulation is used to simulate the behavior of robots in various environments to test their control algorithms and safety measures. Autonomous vehicle testing evaluates the performance of self-driving cars in different driving scenarios without putting humans at risk. In game industry, game AI emulators develop and test AI agents for

video games in simulated environments. In healthcare industry, healthcare emulator simulates medical procedures or patient interactions to train AI agents for healthcare tasks. By using a performance emulator, RL researchers and engineers can efficiently evaluate the performance of their agents in a controlled and scalable environment, accelerating the development of intelligent systems.

Fig. 5.13 shows example performance simulation results using game emulators. On each game, there is a wide range of learnign rates with random initializations. Such performance evaluation is simply not feasible and at best cost-inefficient if feasible when offline and online performance evaluation methods are used. Compared to offline and online performance evaluation methods, evaluation through simulators or emulators can usually obtain more smoothed and stable performance metrics due to its flexible parameter adjustiment and cost-efficiency.

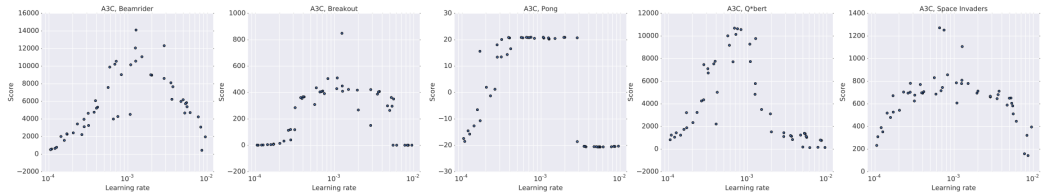


Fig. 5.13: Sample plots of scores obtained by A3C through game emulators of five games (Beamrider, Breakout, Pong, Q*bert, Space Invaders) for 50 different learning rates and random initialization. The results demonstrate the robustness of A3C to different learning rates and initial random weights [34].

5.5 Challenges and Best Practices

Challenges in Performance Evaluation Non-Stationarity: RL environments can be non-stationary, meaning that the dynamics of the environment can change over time. This can make it difficult to evaluate an agent's performance consistently. Stochasticity: RL environments are often stochastic, meaning that the outcomes of actions are not deterministic. This can make it challenging to compare the performance of different agents. Multiple Objectives: In some cases, RL agents may have multiple objectives to achieve. This can make it difficult to define a single performance metric. Best Practices for Performance Evaluation Experiment Design: Carefully design experiments to ensure that they are fair and unbiased. Baseline Comparison: Compare the performance of your RL agent to a baseline, such as a random policy or a simple heuristic. Statistical Significance: Use statistical tests to determine whether the observed differences in performance are statistically significant. Visualization: Use visualizations to help understand the agent's behavior and identify areas for improvement. Ablation Studies: Conduct ablation studies to evaluate the impact of different components of your RL algorithm.

5.6 System-wise Performance Evaluation

System-wise reinforcement learning evaluation encompasses a variety of critical components, including performance evaluation, recovery testing, security evaluation, stress testing, and deployment testing. Performance evaluation focuses on assessing the run-time performance of reinforcement learning (RL) systems, examining how well they function under various operating conditions. Recovery testing is crucial for verifying the RL system's ability to recover from failures or disruptions, ensuring resilience in its operation. Security evaluation is essential as it verifies the effectiveness of the system's protection mechanisms aimed at preventing unauthorized access, penetration, or manipulation by malicious entities. Stress testing plays a vital role in evaluating how well the system copes with abnormal resource demands, particularly when the number of active agents becomes significantly large, which can lead to potential performance bottlenecks. Deployment testing assesses the system's capability to upgrade its software and hardware resources, ensuring that it remains adaptable to changing requirements. While all these aspects are integral to a comprehensive system-wise evaluation of RL systems and warrant detailed exploration, we will narrow our focus to emphasize system-wise performance evaluation in the context of RL foundations.

Performance evaluation involves a thorough and systematic assessment of the performance of an RL agent within the broader framework of a larger system architecture. This evaluation considers a multitude of factors, including integration with existing systems, scalability to accommodate growth, and the myriad challenges associated with real-world deployment scenarios. Such a meticulous evaluation approach is vital for gaining meaningful insights into the practical implications of RL technology. It ensures that RL agents can be effectively utilized in real-world scenarios, where conditions tend to be more complex and variable than those encountered in controlled environments. By concentrating on performance evaluation, we can better understand how to improve RL systems to meet the demands of practical applications, thereby maximizing their effectiveness and reliability.

Key considerations for system-wise performance evaluation encompass a range of critical factors that can influence the overall success of an RL deployment. These factors include not only the technical aspects of integration and scalability but also the inherent challenges that come with real-world applications. Moreover, safety and security concerns must be addressed to preemptively identify any risks associated with deploying the RL agent. Ethical implications also play a significant role; understanding issues such as bias and discrimination is essential in ensuring that the technology is applied responsibly. Lastly, a thorough cost-benefit analysis is key to determining whether the advantages of deploying the RL agent justify the associated costs and resource investments. In summary, a holistic approach to evaluating RL agents within the framework of larger systems is necessary to ensure their effective application in real-world settings. This multi-faceted evaluation process aids in identifying potential pitfalls and optimizing the design and implementation of RL solutions for various applications.

In addition to the common performance evaluation metrics of RL algorithms, system-wide evaluation emphasizes unique metrics that may not be readily accessible through stand-alone or algorithm-specific performance evaluations. These distinct metrics are crucial in understanding the holistic impact of the RL agent on the overall system. The main metrics for system-wise evaluation encompass several key aspects: incremental system performance, cost-effectiveness, user experience, reliability, and adaptability.

- Incremental System Performance: This metric assesses how the overall performance of the system improves with the implementation of the RL agent. It examines not only the efficiency gains but also how the agent contributes to achieving the system's objectives.
- Cost-Effectiveness: This evaluation criterion focuses on whether deploying and maintaining the RL agent is cost-effective. It explores the balance between the financial investment required and the benefits derived from enhanced performance.
- User Experience: An essential aspect of any system is its interaction with users. This metric investigates how the presence of the RL agent influences user satisfaction, engagement, and overall experience, considering both positive and negative impacts.
- Reliability: This metric evaluates how dependable the RL agent is under real-world conditions, considering factors such as consistency of performance and the ability to function correctly over time.
- Adaptability: Finally, adaptability examines the RL agent's capacity to adjust to changes in the environment, including shifting user needs or evolving operational conditions. This flexibility is vital for ensuring long-term success in dynamic settings, thus enhancing the system's robustness.

System-wise evaluation faces multiple real-world challenges that complicate the assessment of complex RL systems. In the first place, evaluating a complex RL system can be particularly daunting due to the intricate interdependence of its various components. Each component often interacts with others in unpredictable ways, leading to emergent behaviors that are difficult to assess. This complexity is the paramount reason why such evaluations may not be practical, especially when relying on basic evaluation methods designed to test the performance of RL algorithms. These methods may fail to capture the intricacies involved in real-world applications. In the second place, real-world environments frequently present constraints and variables that are not present in simulated environments. These discrepancies can significantly impact the differences observed between performance evaluations conducted via system simulations and those undertaken in actual RL systems deployed in the field. Finally, addressing ethical concerns in reinforcement learning can be exceptionally complex and requires thorough and careful consideration. For instance, the evaluation of customer-facing RL systems typically involves sensitive private data concerning customers' behaviors and preferences. Consequently, privacy preservation becomes one of the main ethical concerns, necessitating robust data protection measures and ethical frameworks to ensure that customer data is handled responsibly and transparently. Addressing these challenges is crucial for

advancing the field of RL and ensuring its responsible application in real-world scenarios.

To design and implement effective and efficient system-wise evaluation procedures, best practices include adopting a holistic approach, conducting real-world testing, performing ethical assessments, engaging in continuous monitoring, and actively seeking user feedback.

- Holistic Approach: It is essential to consider the entire system, encompassing hardware, software, and human factors. This comprehensive viewpoint ensures that all components interact seamlessly, leading to a better understanding of the system's overall performance.
- Real-World Testing: Testing the reinforcement learning (RL) agent in real-world conditions is crucial for accurately assessing its performance. This type of testing helps identify strengths and weaknesses that may not be evident in controlled environments, providing valuable insights into how the agent will behave when deployed.
- Ethical Assessment: Conducting an ethical assessment is vital to identify and address potential risks associated with the RL agent. This process helps ensure that the system aligns with ethical standards and addresses concerns such as bias, privacy, and fairness.
- Continuous Monitoring: Ongoing monitoring of the RL agent's performance over time allows organizations to identify and address emerging issues proactively. By tracking key performance indicators, organizations can make timely adjustments to improve the agent's effectiveness and reliability.
- User Feedback: Gathering feedback from users is critical for understanding the RL agent's impact on their experience. Engaging with users helps uncover insights that can lead to enhancements in the system, ensuring it meets their needs and expectations effectively.

By diligently following these guidelines, organizations can effectively evaluate the performance of RL agents within the broader context of their systems, ultimately leading to more informed decisions regarding their deployment and ongoing optimization. This multi-faceted approach fosters a deeper understanding of how these agents operate and interact with various elements of the system, ensuring their success in real-world applications.

References

[1] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29(13):1608–1639, 2010.

[2] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Ng. An application of reinforcement learning to aerobatic helicopter flight. *Advances in neural information processing systems*, 19, 2006.

[3] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.

[4] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems*, 13:341–379, 2003.

[5] Sarah Bechtle, Artem Molchanov, Yevgen Chebotar, Edward Grefenstette, Ludovic Righetti, Gaurav Sukhatme, and Franziska Meier. Meta learning via learned loss. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4161–4168. IEEE, 2021.

[6] Jacob Beck, Risto Vuorio, Evan Zheran Liu, Zheng Xiong, Luisa Zintgraf, Chelsea Finn, and Shimon Whiteson. A survey of meta-reinforcement learning. *arXiv preprint arXiv:2301.08028*, 2023.

[7] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International conference on machine learning*, pages 449–458. PMLR, 2017.

[8] Léon Bottou. Online algorithms and stochastic approximations. *Online learning in neural networks*, 1998.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[10] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[11] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. *Innovations in multi-agent systems and applications-1*, pages 183–221, 2010.

[12] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198, 2016.

[13] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. *Advances in neural information processing systems*, 29, 2016.

[14] G Dulac-Arnold, N Levine, DJ Mankowitz, and etc. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110:2419–2468, 2021.

[15] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala:

Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

[16] Sergio Guadarrama etc. TF-Agents: A library for reinforcement learning in tensorflow. https://github.com/tensorflow/agents, 2018. [Online; accessed 25-June-2019].

[17] Maor Gaon and Ronen Brafman. Reinforcement learning with non-markovian rewards. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3980–3987, 2020.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[19] Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. *Advances in neural information processing systems*, 31, 2018.

[20] Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *AAAI*, volume 4, pages 709–715, 2004.

[21] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado Van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3796–3803, 2019.

[22] Todd Andrew Hester, Evan Jarman Fisher, and Piyush Khandelwal. Predictively controlling an environmental control system, January 16 2018. US Patent 9,869,484.

[23] Rein Houthooft, Yuhua Chen, Phillip Isola, Bradly Stadie, Filip Wolski, OpenAI Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. *Advances in Neural Information Processing Systems*, 31, 2018.

[24] Wenyi Huang and Jack W Stokes. Mtnet: a multi-task neural network for dynamic malware classification. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*, pages 399–418. Springer, 2016.

[25] Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *ICLR*, 2019.

[26] Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[27] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd, 2018.

[28] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[29] Frank L Lewis and Kyriakos G Vamvoudakis. Reinforcement learning for partially observable dynamic processes: Adaptive dynamic programming using

measured output data. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(1):14–25, 2010.

[30] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[31] Robert Loftin, Aadirupa Saha, Sam Devlin, and Katja Hofmann. Strategically efficient exploration in competitive multi-agent reinforcement learning. In *Uncertainty in Artificial Intelligence*, pages 1587–1596. PMLR, 2021.

[32] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[33] IA Luchnikov, SV Vintskevich, DA Grigoriev, and SN Filippov. Machine learning non-markovian quantum dynamics. *Physical review letters*, 124(14):140502, 2020.

[34] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[35] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

[37] Daniel Neider, Jean-Raphael Gaglione, Ivan Gavran, Ufuk Topcu, Bo Wu, and Zhe Xu. Advice-guided reinforcement learning in a non-markovian environment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9073–9080, 2021.

[38] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *ICML*, volume 1, page 2, 2000.

[39] Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado P van Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms. *Advances in Neural Information Processing Systems*, 33:1060–1070, 2020.

[40] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[41] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. *Advances in neural information processing systems*, 10, 1997.

[42] Pascal Poupart, Aarti Malhotra, Pei Pei, Kee-Eung Kim, Bongseok Goh, and Michael Bowling. Approximate linear programming for constrained partially observable markov decision processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[43] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *Machine learning: ECML 2005: 16th European conference on machine learning, Porto, Portugal, October 3-7, 2005. proceedings 16*, pages 317–328. Springer, 2005.

[44] Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[45] Nicolas Schweighofer and Kenji Doya. Meta-learning in reinforcement learning. *Neural Networks*, 16(1):5–9, 2003.

[46] David Silver, Nicolas Heess, Lukas Schäfer, Stefano Albrecht, and Josiah Hanna. Robust on-policy sampling for data-efficient policy evaluation in reinforcement learning. *Advances in Neural Information Processing Systems*, 38:37376–37388, 2022.

[47] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[48] Joseph Suarez, Yilun Du, Phillip Isola, and Igor Mordatch. Neural mmo: A massively multiagent game environment for training and evaluating intelligent agents. *arXiv preprint arXiv:1903.00784*, 2019.

[49] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

[50] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[51] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[52] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[53] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[54] Yee Teh, Victor Bapst, Wojciech M Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[55] Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 2139–2148. PMLR, 2016.

[56] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[57] Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. Androidenv: A reinforcement learning platform for android. *arXiv preprint arXiv:2105.13231*, 2021.

[58] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30-1, 2016.

[59] Wikipedia contributors. Stochastic gradient descent — Wikipedia, the free encyclopedia, 2024. [Online].

[60] Haiyan Yin and Sinno Pan. Knowledge transfer for deep reinforcement learning with hierarchical experience replay. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[61] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1095. PMLR, 2020.

[62] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pages 321–384, 2021.

[63] Zeyu Zheng, Junhyuk Oh, Matteo Hessel, Zhongwen Xu, Manuel Kroiss, Hado Van Hasselt, David Silver, and Satinder Singh. What can learned intrinsic rewards capture? In *International Conference on Machine Learning*, pages 11436–11446. PMLR, 2020.