Institute for Artificial Intelligence Systems - MLS

University of Stuttgart
Universitätsstraße 32
D–70569 Stuttgart

Masterarbeit

Model-Based Reinforcement Learning under Sparse Rewards

Ravi Akash

Study program:	Information Technology
Examiner:	Prof. Dr. Mathias Niepert
Supervisors:	Carlos E. Luis, M.Sc, Dr. Ing. Felix Berkenkamp (Bosch Center for Artificial Intelligence)
Commenced:	February 15, 2023
Completed:	August 14, 2023

Abstract

Reinforcement Learning (RL) has recently seen significant advances over the last decade in simulated and controlled environments. RL has shown impressive results in difficult decision-making problems such as playing video games or controlling robot arms, especially in industrial applications where most methods require many interactions with the system in order to achieve good performance, which can be costly and time-consuming. Model-Based Reinforcement Learning (MBRL) promises to close this gap by leveraging learned environment models and using them for data generation and/or planning and, at the same time trying to be sample efficient. However, Learning with sparse rewards remains a significant challenge in the field of RL. In order to promote efficient learning the sparsity of rewards must be addressed. This thesis work tries to study individual components of MBRL algorithms under sparse reward settings and investigate different design choices made to measure the impact on learning efficiency. Suitable Integral Probability Metrics (IPM) are introduced to understand the model's reward and observation space distribution during training. These design combinations will be evaluated on continuous control tasks with established benchmarks.

Kurzfassung

RL hat in den letzten zehn Jahren bedeutende Fortschritte in simulierten und kontrollierten Umgebungen verzeichnet. RL hat beeindruckende Ergebnisse bei schwierigen Entscheidungsproblemen erzielt, wie zum Beispiel das Spielen von Videospielen oder die Steuerung von Roboterarmen, insbesondere in industriellen Anwendungen, bei denen die meisten Methoden viele Interaktionen mit dem System erfordern, um eine gute Leistung zu erzielen. Dies kann kostspielig und zeitaufwändig sein. MBRL verspricht, diese Lücke zu schließen, indem gelernte Umgebungsmodelle genutzt werden, um Daten zu generieren und/oder zu planen, und gleichzeitig versucht wird, eine hohe Probeneffizienz zu erreichen. Die Herausforderung des Lernens mit spärlichen Belohnungen bleibt jedoch ein bedeutendes Problem im Bereich des RL. Um ein effizientes Lernen zu fördern, muss die Spärlichkeit der Belohnungen angegangen werden. Diese Masterarbeit versucht, einzelne Komponenten von MBRL-Algorithmen unter Bedingungen mit spärlichen Belohnungen zu untersuchen und verschiedene Designentscheidungen zu untersuchen, um ihre Auswirkungen auf die Lerneffizienz zu messen. Geeignete IPM werden eingeführt, um das Belohnungs- und Beobachtungsraumverteilung des Modells während des Trainings zu verstehen. Diese Designkombinationen werden anhand von kontinuierlichen Steuerungsaufgaben mit etablierten Benchmarktests ausgewertet.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

A2C Advantage Actor-Critic. 25

BLR Bayesian Linear Regression. 18

CEM Cross Entropy Method. 27

CPC Contrastive Predictive Coding. 15

CPU Central Processing Unit. 39

DDP Differential Dynamic Programming. 18

DL Deep Learning. 21

DQN Deep Q-Networks. 15

DRL Deep Reinforcement Learning. 21

EMD Earth Mover Distance. 33

GAN Generative Adversarial Networks. 15

GP Gaussian Process. 13

GPU Graphical processing units. 39

HER Hindsight Experience Replay. 17

HPC High Performance Cluster. 39

ICM Intrinsic Curiosity Model. 16

IPM Integral Probability Metrics. 3

MBPO Model Based Policy Optimization. 20

MBPO-C Model Based Policy Optimization-Consistent. 31

MBRL Model-Based Reinforcement Learning. 3

MCTS Monte Carlo Tree Search. 27

MDP Markov Decison Process. 14

MFRL Model-Free Reinforcement Learning. 20

MMD Maximum Mean Discrepancy. 33

MPC Model Predictive Control. 18

MVE Model Value Expansion. 27

NN Neural Networks. 13

PPO Proximal Policy Optimization. 25

PSRL Posterior Sampling for Reinforcement Learning. 18

RKHS Reproducing Kernel Hilbert Space. 35

RL Reinforcement Learning. 3

RND Random Network Distillation. 17

SAC Soft Actor Critic. 17

SLBO Stochastic Lower Bound Optimization. 28

STEVE Stochastic Ensemble Value Expansion. 27

TRPO Trust Region Policy Optimization. 25

WD Wasserstein Distance. 33

1 Introduction

RL has become a popular paradigm of Machine Learning over the last decade, where intelligent agents learn to act in some environment with the objective of maximizing cumulative rewards. The agents learn optimal strategies directly by interacting with an *a priori* unknown dynamical system [SB20]. RL has garnered a lot of attention and popularity in the research community due to the success of mastering the game of Go [SSS+17] than any other human player could, winning in most of the Atari games it has been deployed to play and most recently Deepmind's Starcraft II [VEB+17]. RL algorithms are categorized mainly into two classes MFRL and MBRL. The key difference between them is that MFRL learns a policy by directly interacting with the environment, while in MBRL, the agent learns an approximate model of the environment's dynamics and uses this model for policy optimization or planning. The key advantage of MBRL over MFRL is that the agent can first learn the model of the environment and then can make informed decisions when interacting with the actual environment. This helps MBRL to be more sample-efficient learning and faster convergence to optimal policies. The learned model can also be used for planning and exploration, allowing the agent to explore various scenarios and hypothetical actions without risking real interactions with the environment.

A fundamental challenge in RL is dealing with sparse rewards, It is crucial to study sparse reward problems, since they often appear in real-world tasks and are easy to design without domain knowledge. The sparse reward can be specified as long as there is a defined state-based criterion for success (e.g., a goal location is reached) since there are no rewards elsewhere except in the area of the state space that meets the success criterion. In most real-world cases it would be hard to specify dense rewards since we only have limited knowledge about the system [DMH19]. Most of the traditional Rl and MFRL algorithms might fail to solve the sparse reward problems because of the absence of intermediate rewards which enable the agent to drive exploration. MBRL can be particularly helpful in solving sparse reward tasks since it offers the promise to leverage the uncertainty of the learned dynamics model to drive the exploration toward interesting regions of the state space.

One of the pioneers from the field of reinforcement learning shared his view on MBRL and how it certainly plays an essential role in shaping the research ahead:

> *"The next big step forward in AI will be systems that actually understand their worlds. The world is only accessed through the lens of experience, so to understand the world means to be able to predict and control your experience, and your sensitive data, with some accuracy and flexibility. In other words, understanding means forming a predictive model of the world and using it to get what you want. This is model-based reinforcement learning."*
>
> *-Richard S. Sutton*

The asymptotic performance of MBRL algorithms was historically lower than that of model-free methods, but the gap has been closing in recent years [JFZL21] as a lot of effort has been directed towards scaling RL for real-world environments and also dealing with sparse rewards [WEH+22]. MBRL has shown great success in terms of sample efficiency [DR11], [BHT+19], improving planning [JFZL21] and being robust enough for distributional shifts[FVM+22].

The major motivation behind studying sparse reward problems is that it requires close to no domain knowledge to design a reward function for a given task/environment. For example, in a continuous control task where a robotic arm task is required to stack an object on top of other objects, we can assign a sparse reward of 1 if the robotic arm successfully stacks the object and 0 otherwise. Designing intermediate rewards in this scenario can be challenging and in most cases not practical. The sparse reward problem has been addressed in previous state-of-the-art literature in a plethora of ways, in which Gaussian Process (GP) [DR11] and Neural Networks (NN) [CCML18] were used as a typical model for representing the one-step dynamics of the RL environment. Ensembles of probabilistic NN are a common paradigm used by a plethora of MBRL methods leveraging uncertainty estimates to improve performance [BHT+19],[ZZW+19]. The learned models are then used, for instance, for planning as done by [CCML18]. [FVM+22] present one of the most recent studies where classic RL benchmarks are modified for different reward sparsity levels. However, research in MBRL under a sparse reward setting has not answered some key questions such as how accurate and diverse the model should be, how aware the model should be about uncertainty, and how long the rollout trajectory should be. While it is fairly common for authors to regularly experiment and provide various algorithms for solving sparse reward tasks, it still remains an open question as to what the key ingredients and design choices which are essential for solving sparse reward tasks. The study carried out in this thesis helps us understand why some hyperparameter design choices might be working better than others through the lens of the dataset which we use to train the agent.

1.1 Goal of the Thesis

The primary focus of this study revolves around understanding the essential elements within the Model-Based Reinforcement Learning (MBRL) algorithm that effectively tackle the challenge of sparse rewards. The objective of this thesis is to thoroughly examine the current state-of-the-art MBRL algorithm, namely Model-Based Policy Optimization (MBPO), and investigate the impact of various design decisions in the algorithm's design when dealing with sparse rewards scenarios. Modifications suggested in the literature are used as a starting point for benchmarking performance under sparse rewards in a customized sparse inverted pendulum [CBK20], mountain car continuous [BCP+16a] and a complex DeepMind continuous control environment cheetah run [TMD+20], which are stochastic in terms of their initial state and are well suited for testing continuous control tasks.

1.2 Thesis Overview

The remainder of this work is organized as follows. Chapter 2 introduces prior research from the literature that is pertinent to our approach. Subsequently, Chapter 3 delves into the necessary technical background, encompassing topics such as MBRL, Markov Decison Process (MDP), and more. Chapter 4, we discuss the details of the design of experiments conducted and their particulars. Chapter 5 encompasses the details of the experimental setup and hardware specifications. Moving forward to Chapter 6, we present the assessments and in-depth analysis of the outcomes derived from the simulated experiments. Transitioning to Chapter 7, we provide a succinct summary of our findings while also offering glimpses into potential avenues for future research.

2 Related Work

This section summarizes prior state-of-the-art literature on solving sparse reward problems. An overview of the literature is provided, as well as a brief discussion of the methodologies employed.

The sparse reward problem is a common and difficult challenge that many RL algorithms face in real-world scenarios. The sparse reward problem is particularly challenging due to the fact that there is little signal to learn from and also rely on accurate models of environments to make effective decisions. Moreover, this sparse reward setting makes the learning process slower and less efficient. In order to overcome the difficult challenges posed by the sparse reward problem, a wide range of novel ideas in deep reinforcement learning research have emerged.

[VHS+18] proposed a model-free approach using demonstrations collected from a human demonstrator to tackle the sparse reward problem on a high-dimensional robotic control problem. The purpose of using demonstrations is to replace carefully engineered rewards and reduce the exploration burden in sparse reward settings.

A hybrid approach using Deep Q-Networks (DQN) was suggested by [GL19], which combines both model-free and model-based approaches that help in exploring least likely seen states and learning environments with sparse rewards more efficiently.

Reward shaping is an effective strategy to help the RL algorithms learn more efficiently where carefully engineered rewards are introduced in the environment to guide the algorithm towards convergence. [LTA20] propose an reward shaping methodology through Contrastive Predictive Coding (CPC) by learning predictive representations offline. CPC learns self-supervised representations by predicting the future in latent space with the help of auto-regressive models. With their algorithm, long-horizon tasks can be addressed with shaped reward signals. Another parallel literature work proposed by [WWWZ20], introduces a non-expert helpers algorithm, where a prior control policy(non-expert helper) plays an important role in guiding the agent in exploring the state space by dynamically reshaping the learning objectives over time.

[CM20] propose PlanGAN a model-based algorithm for addressing the multi-goal task in the presence of sparse rewards. The collected experience by an RL algorithm is used to train an ensemble of Generative Adversarial Networks (GAN) to generate multiple plausible experiences, where these experiences are combined into a novel planning algorithm that helps in achieving efficient learning. [KCM20] frame policy search as a multi-objective problem, where the objectives are optimized using a Pareto-based multi-objective optimization problem. The proposed approach was able to solve sparse reward tasks within a few episodes.

[LWZZ21] take advantage of model error as an extra reward which aids in increasing reward density in sparse reward settings in order to drive the exploration. [NHM21] introduce a novel computationally light model-based approach to tackle the sparse reward problem, which encourages the agent to explore uncertain states by considering a prior model of their goal behavior. To tackle sparse reward problems in continuous action setting [WWW21] deployed a particle swarm optimization planner as an actor in actor-critic architecture which helped to regulate the exploration rate. This aided the RL algorithm to identify rewards in non-interesting regions in state space. Motivated by the recent success of value-based for approximating state-action values [RYH+21] utilize radial basis value function for addressing continuous control robotic manipulation multi-task sparse reward settings.

An effective alternative for reward shaping was proposed by [DDHB22], where model predictive control is utilized as an experienced source for training RL algorithms in sparse reward environments. Algorithm complexity increases significantly when demonstrations are involved as there is more hyperparameters introduced and tuned. As an effective alternative to this approach [WBD+22] introduce a parameter-free modification to standard actor-critic algorithms which computes a modified Q-value and a Monte Carlo estimate of the reward-to-go as a result increasing learning efficiency in sparse reward tasks. [LWRG22] leverage long-term Q-Values and provide richer feedback signals to the actions taken to improve learning efficiency. [SGK22] propose a novel approach of redistributing local and shared global rewards across multiple agents thereby boosting exploration in multi-agent sparse reward setting.

In the following sections, various approaches are broadly categorized into standard methodologies which are utilized in addressing the sparse reward problem are examined.

2.1 Curiosity-Driven Exploration

The main idea behind curiosity-driven methods is that the RL algorithm is encouraged to visit unseen states in the environment, The main intuition behind this approach is that an agent that is curious about its environment will be more likely to discover novel and informative experiences, which can ultimately lead to better performance in the long run.

[HCD+16] proposes an exploration strategy that maximizes information gain about the algorithm's belief of environment dynamics, which has significant performance in tackling continuous control tasks with sparse rewards. [PAED17] propose Intrinsic Curiosity Model (ICM) where curiosity is formulated as an error in the algorithm's ability to predict the consequence of its own actions in a visual feature space learned by self-supervised inverse dynamics model. The algorithm is composed of two subsystems, one which outputs a curiosity-driven intrinsic reward signal and a policy that outputs a sequence of actions to maximize the reward signal. While the previous approach used a model-free agent to solve the environments [SRD+20] showed that the idea of curiosity can be combined with MBRL to create an agent that efficiently explores and solves sparse reward tasks. However, blindly pursuing novel states renders the previous methods' sample inefficient and may also fail to converge to useful behavior in some cases. To overcome this issue, [LLL+20] formulate a goal-oriented curiosity-driven exploration method and dynamic initial state selection mechanism which achieves a much higher success rate and leads to faster convergence.

2.2 Random Network Distillation

The main idea behind curiosity-driven learning is to build a reward function that is intrinsic to the algorithm but has serious drawbacks due to the "noisy TV problem". This is a common issue in RL where the observations are noisy or incomplete due to environmental factors such as sensor noise, occlusions, or partial observability. In this scenario, the algorithm must learn to distinguish between the relevant and irrelevant parts of the observation in order to make accurate decisions. Imagine that an RL algorithm is rewarded with seeking novel experience, and RL algorithms are distracted forever by stochastic elements in the environment. So at every timestep, the curiosity reward will be much higher and pushes the algorithm to pursue these noisy states. Random Network Distillation (RND) addresses this problem by calculating curiosity but is not attracted by the stochastic elements of an environment. RND consists of a non-trained target network with fixed random weights and a predictor network that tries to predict the target network's output. The target network feature representation for a given state will be the same. The predictor network predicts the target network's output for the next state. Therefore the next state is propagated into both networks and the prediction network is trained to minimize the mean square error between the output of two networks. This process distills a randomly initialized neural network (target) into a trained (predictor) network. To improve exploration in several hard Atari games, [BESK18] included RND bonus combined with flexible integration of intrinsic and extrinsic rewards helped in achieving better than human average performance. An ensemble-free alternative approach was suggested by [NKTK23] for quantifying uncertainty by making use of RND combined with Soft Actor Critic (SAC) which helped deliver performance comparable to ensemble-based methods.

2.3 Hindsight Experience Replay

Curiosity driven methods try to maximize the statistical novelty of states in comparison with previously experienced states. There is a potential pitfall in this method, the variation among states explored is observed only when there is a significant amount of diversity present among the states. When the explored states lack diversity in large state space ICM will fail to explore the state space efficiently. In a sparse reward environment, the algorithm finds it difficult to appropriately explore the environment and learn the sequence of steps required to achieve the desired goal.

Hindsight Experience Replay (HER) [AWR+17] consists of a buffer that stores a copy of each trajectory experienced and replaces the actual rewards with rewards calculated assuming the goals are the steps achieved at the end of the trajectories. As a result, the algorithm will be able to explore more effectively and learn intermediate goals that build up to the actual goal. [ZZLL20] leverage demonstrations to accelerate training and propose a new experience replay mechanism to address the sparse reward problems in robotic tasks. Sequential object manipulation robotic tasks are extremely difficult in sparse reward settings where, [LWD+22] proposes relay HER, where the huge sequential task is decomposed into multiple sub-tasks. The decomposed sub-tasks are efficiently learned with the help of HER. The concept of HER is applied to model-based methods to solve multi-goal tasks in sparse reward settings. [MR21] propose Imaginary-HER which incorporates imaginary data into policy updates to improve exploration using HER and is endowed with curiosity-based intrinsic rewards. This imaginary data is replaced each time the model is updated. Another parallel work proposed by [YFH+21] introduces model-based HER which exploits experiences by leveraging environmental dynamics to generate imaginary virtual goals. These generated virtual goals allows the algorithm to reinterpret its actions using a different goal as per the latest policy.

2.4 Policy Optimization

Policy optimization methods are centered around the policy, a function that maps the agent's state to its next action. These methods view reinforcement learning as a numerical optimization problem where the expected rewards are optimized with respect to the policy's parameters. [LK13] proposed a guided policy search algorithm that has the capability of learning complex policies by incorporating guiding samples into the policy search. A model-free initial policy search is guided by a model-based Differential Dynamic Programming (DDP) which samples from a distribution of high reward trajectories, is helpful in optimizing the policy without requiring new on-policy samples.

In this line of work of policy optimization, [JFZL21] propose a simple procedure of making limited use of the model. In particular, unclasping the model goal and the original task goal by querying the model only short rollouts. [CBK20] introduce an optimistic exploration algorithm that augments the control space of the agent, which helps to control the agent's epistemic uncertainty for short transition dynamics. They try to address the problem of greedy and insufficient exploration of MBRL algorithms with probabilistic dynamical problems by leveraging the model uncertainty to optimize the policy.

As model-based methods often struggle with errors in learned models, [ZLW20] deduces an upper bound for the uncertainty in Q-values in order to achieve asymptotic performance as model-free methods. Furthermore, [QPC20] propose an uncertainty-aware trust region policy optimization algorithm that optimizes the policy conservatively thereby increasing performance and reducing overfitting to inaccurate models by optimizing conservatively.

[CVLH19] formulate model-free optimistic actor-critic which facilitates deep exploration by leveraging the predictive uncertainty of the policy performance during policy optimization. They achieve this by using a bootstrap method to estimate the epistemic uncertainty of the Q-function so that they can adjust the upper confidence bound for critics based on the principle of optimism in the face of uncertainty. In this similar line of work, [FM21] propose a model-based version by implementing Posterior Sampling for Reinforcement Learning (PSRL) with function approximation and making use of Bayesian Linear Regression (BLR) when fitting transition and reward models. In the end, the authors implement Model Predictive Control (MPC) to optimize policy under the sampled models in each episode.

[FLZB22] introduce on-policy corrections methodology which uses on-policy transition data accompanied by a learned model in order to make accurate long-term predictions for MBRL. The authors show that when generating trajectories on-policy with the model true state distribution can be recovered by improving by means of policy improvement bound.

The idea of policy optimization is further extended to Offline RL by [YTY+20]. In this paper, the authors present an offline MBRL algorithm that optimizes a policy in an uncertainty-penalized MDP, the proposed algorithm penalizes states with high model uncertainty with a policy that maximizes the MDP. The algorithm ensures that the traps of behavioral distribution can be mitigated while avoiding the risk of making mistakes.

3 Technical Background

In this chapter, we briefly introduce the standard formalisms and techniques on which the thesis is based, as well as their significance in the machine learning and reinforcement learning communities. In Section 3.1, an introduction to reinforcement learning and the framework in which it operates is provided. This aids in framing the issue of sparse rewards in Section 3.1.7. In section 3.2, a brief introduction is given to Model-Free Reinforcement Learning (MFRL) and in section 3.3 to MBRL. Further, a brief discussion of a model-based algorithm, Model Based Policy Optimization (MBPO) method is provided. This method serves as a baseline upon which the rest of the thesis investigation is carried out.

3.1 Reinforcement Learning

Reinforcement learning, is one of the three fundamental machine learning paradigms, along with supervised and unsupervised learning. RL is involved in making sequences of decisions where it considers an intelligent agent situated in an unknown environment. At each timestep, the agent takes an action and receives an observation and reward. The primary goal of the RL algorithm is to maximize the notion of cumulative reward, given an unknown environment, through a trial-and-error learning process. Section 3.1.4 onwards, provides a more detailed description of the mathematical formulation of RL.

In the Figure shown below Figure 3.1, depicts an extremely general RL problem that involves a reward-maximizing agent. There have been numerous applications of RL algorithms to various fields, including robot control [KBP13], [CER20] applied RL techniques to economics, game theory, operation research and finance, and the recent technological trend of large language models [Ope23] where RL is used to fine-tune the model's behavior with human feedback, to produce responses which are better aligned with the user's intent.

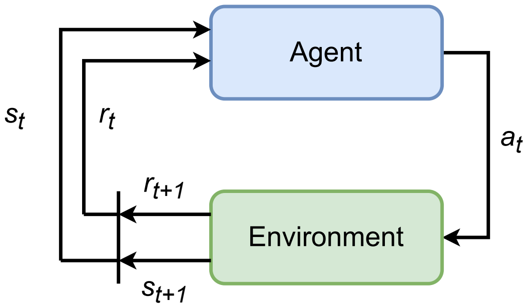


Figure 3.1: Interaction loop between an RL agent and environment. The reward and the state resulting from taking an action serve as the input for the next iteration [SB18]

3.1.1 Deep Reinforcement Learning

Most of the modern machine learning methods are primarily concerned with learning functions from data. Deep Learning (DL) is a sub-field of machine learning that uses artificial neural networks to model and solve complex problems. DL involves a loss function, a non-linear function approximator, and utilizes the gradient descent method to optimize parameters between nodes in order to minimize the difference between the predicted output and the actual output. DL has been applied to various fields and has produced some groundbreaking results in computer vision [SPP18], autonomous driving [HC20] and natural language processing [TSK+21]. DL transforms a learning problem to an optimization problem in a straightforward fashion in supervised learning scenarios, but this reduction is less straightforward in RL.

The two main caveats involved in RL with respect to supervised learning are that the dataset is non-stationary (changing over time) and the data is not independently and identically distributed, instead the data is strongly correlated. Moreover, the input data to the agent strongly depends on how it behaves in the unknown environment which makes it difficult to develop straightforward algorithms to solve the task. In RL we have various choices of approximating such as policies, value functions, dynamics models, and what kind of function approximators to utilize. all the above-mentioned can also choose in various combinations when solving a RL problem.

Deep Reinforcement Learning (DRL) is a subfield of machine learning which studies reinforcement learning using neural networks as non-linear function approximators. DRL incorporates DL which helps the RL agent to make decisions from unstructured input data without the need for manual engineering of the state space. DRL algorithms have the capacity to take ingest large datasets to maximize the cumulative reward. DRL has been applied to diverse sets of applications where [SSS+17] mastered the game of Go to play better than most human experts. They utilized a combination of supervised learning, several RL steps to train deep neural networks combined with a Monte-Carlo tree search algorithm. [MKS+13] present DQN a classical example of DRL, since it scales up conventional Q-learning to perform tasks with more complex observation space. Since then, there has been an explosion of state-of-the-art DRL algorithms applied in various domains. In the upcoming subsections, individual components which contribute to the DRL are discussed in detail.

3.1.2 Markov Decision Process

We consider a RL agent that interacts with a MDP. An MDP is described by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mu, \gamma \rangle$, $\mathcal{P}(s'|s,a)$ defines a Markovian transition probability density between the current state s and the next state s' under action a. The initial state distribution is defined as $\mu : \mathcal{S} \to [0, 1]$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and a discount factor $\gamma \in [0, 1)$. Given an MDP, the goal of RL agent is to learn a specific behavior in an unknown environment. That is, we want to learn an action selection policy that maximizes the expected cumulative reward

3.1.3 Policy

The policy π determines the agent's behavior by selecting an action given its current state and is formally defined as a mapping from a state s to an action a, $\pi : \mathcal{S} \longrightarrow \mathcal{A}$. It is possible for the policy to be a deterministic function, which means it will always return the same action regardless of the state. Policies can also be stochastic and be defined by a distribution $\pi : \mathcal{S} \times \mathcal{A} \longrightarrow [0, 1]$ where $\mathcal{A}$ a probability is assigned to each action a given a state s.

When an agent is present in a state it has a wide variety of actions to choose from and we need a performance measure to evaluate a given policy. The performance of a policy is quantified by means of its return R. This return is defined as the sum of all rewards r_t received within an episode starting at $t = 0$ and terminating at $t = T$.

$$R = \sum_{t=0}^{T} r_t \tag{3.1}$$

In the finite horizon setting, the return is calculated over a finite number of timesteps and is therefore bounded. However, to deal with infinite horizons tasks a discount factor, $\gamma \in [0, 1]$ is introduced to weigh down the contribution of distant future rewards. This discounted return is further made used in calculations of value functions.

$$R = \sum_{t=0}^{\infty} \gamma^t r_t \tag{3.2}$$

3.1.4 Value Function

The RL agent needs to find a policy $\pi(a|s)$ to take an appropriate action a when the agent is in state s. There are multiple actions which the agent can choose from a particular state and this is given by a so-called state value function or state-action value function. The state-action value function indicates how beneficial it is to be in a certain state s and perform a certain action a. Formally, the state value function can be defined as the expected return when starting in state s and selecting actions according to the policy π:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right]. \tag{3.3}$$

Correspondingly, the state-action value function is given by

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a\right], \tag{3.4}$$

When we have access to the reward function and the transition function, the state-action value function can be calculated recursively:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{\pi}\left[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \mid s_0 = s, a_0 = a\right] \\ &= \mathbb{E}_{\pi}\left[r_0 \mid s_0 = s, a_0 = a\right] + \gamma \mathbb{E}_{\pi}\left[r_1 + \gamma r_2 + \ldots \mid s_0 = s, a_0 = a\right] \\ &= r(s,a) + \gamma \sum_{s'} P(s' \mid a, s) \mathbb{E}\left[r_1 + \gamma r_2 + \ldots \mid s_1 = s'\right] \\ &= r(s,a) + \gamma \sum_{s'} P(s' \mid a, s) Q^{\pi}(s', \pi(s')) \end{aligned} \tag{3.5}$$

The above equation is called the Bellman equation for the state-action value function. When we consider two policies π and π', where $\pi \geq \pi'$ if and only if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in \mathcal{S}$.

Analogously, we can calculate a similar Bellman equation for the state value function which is given by:

$$V^{\pi}(s) = r(s, \pi(s)) + \gamma \sum_{s'} P(s' \mid \pi(s), s) V^{\pi}(s') \tag{3.6}$$

Therefore, we can assess policies given their value functions and derive an optimal policy π^* based on value functions. The value function that defines the optimal policy π^* is called the optimal state-action value function $Q^*(s,a)$ which is given by:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a), \quad \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{3.7}$$

An optimal state-value function $V^*(s)$ is given by:

$$V^*(s) = \max_{\pi} V^{\pi}(s), \quad \forall s \in \mathcal{S} \tag{3.8}$$

And we can define the Bellman optimality equation for the state-action value function as:

$$Q^*(s,a) = r(s,a) + \gamma \sum_{s'} P(s' \mid a, s) Q^{\pi}(s', \pi(s')) \tag{3.9}$$

Similarly, we can define the Bellman optimality equation for the state value function:

$$V^*(s) = \max_{a} \left[r(s,a) + \gamma \sum_{s'} P(s' \mid a, s) V^*(s) \right] \tag{3.10}$$

And finally the optimal policy π^* is given as:

$$\pi^*(s) = \operatorname*{argmax}_{a} \left[r(s,a) + \gamma \sum_{s'} P(s' \mid a, s) Q^*(s,a)) \right] \tag{3.11}$$

We can write the above equation in a simplified form:

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q^*(s, a) \tag{3.12}$$

3.1.5 Sparse Rewards

Reward functions play an important role in helping the agent to learn a better policy. In sparse reward problems, the reward function returns a reward signal which is typically not informative towards learning an optimal policy. Instead, only a small region of state-action space is well-defined to provide valuable feedback for learning. Learning becomes very difficult in these scenarios because the algorithm must internally realize when the agent made a mistake during an episode, which led to negative rewards after termination. Similarly, after a successful episode, the agent needs to know what action led to a successful episode. Ideally RL algorithms are applied to simulated environments where it is possible to define rewards at each time step and help the agent to reach a goal state. Unfortunately, for many problems in real-world settings, this is not the case.

Let us consider a state space of a previously considered example of the robotic arm stack task to be represented by a tuple (S, A, P, R, γ) where, S is the set of possible states in the environment, A is the set of possible actions that the agent can take, $P(s' \mid s, a)$ represents the transition probability from state s to state s' when action a is taken $R(s, a, s')$ is the reward received by the agent when transitioning from state s to s' by taking action a and γ is the discount factor that determines the importance of future rewards.

The reward can be defined as $R(s, a, s')$ sparse, that is rewards are only achieved when the robotic arm is in a specific small region of defined reward space. This makes the agent learning an optimal policy more difficult. The agent may receive a positive reward of +1 only when it reaches the goal state and zero rewards for all other state-action transitions. In this case, the agent has to rely on sparse positive rewards to learn an effective policy. Mathematically, this can be represented as,

$$R(s, a, s') = \begin{cases} +1, & \text{if } s' \text{ is goal state} \\ 0, & \text{otherwise} \end{cases}$$

3.2 Model-Free Reinforcement Learning

MFRL is a group of algorithms that learn optimal policies through trail-and-error interactions with an environment without the need for explicitly modeling the environment's dynamics i.e. the agent tries to learn from experience, rather than building a model of the environment's dynamics and utilizing it to plan ahead.

In the MFRL setting, the agent explores the environment and receives observations that represent the state of the environment at a given time. Agents choose an action based on observations in the environment and receive a reward as feedback on the quality of their decisions. The goal of the agent is to learn a policy that maximizes the expected cumulative reward over time. It is possible to achieve this goal by using value-based or policy-based algorithms, which are two main classifications of MFRL algorithms [Sil15]. There are a plethora of RL algorithms, which are classified as On-Policy or Off-Policy methods, and in the context of this thesis, we currently focus on the category of off-policy learning and the actor-critic framework.

On-Policy methods update the policy based on the data the agent collects using a current policy i.e. the data used for learning is from the interactions with the environment utilizing the current policy. As a result, the agent can only update the policy it is currently following. On the other hand, Off-Policy methods learn from data collected by following a different policy. The agent has a flexible choice of choosing either a random policy or a previously learned policy. This flexibility allows the agent to learn more efficiently by reusing past experience.

The above-mentioned data collection methods can be applied to both Value-based and Policy-based methods. Value-based methods typically estimate the value of each state or state-action pair and utilize this value to select actions and in turn, maximize the expected cumulative reward. Policy-based methods on the other hand, directly learn a policy, mapping each state to a probability distribution over actions. The policy is updated based on the observed rewards, and it involves computing the gradient of the policy's performance objective. Policy-based methods can be further classified as Gradient-free and Gradient-based methods.

Gradient-free methods do not rely on gradient information for policy optimization. Instead, their approach consists of searching the policy space directly for optimal policies, utilizing evolutionary algorithms or other search-based optimization techniques such as Particle Swarm Optimization and the cross-entropy method. These methods explore the policy space through trial and error, gradually improving policies over iterations.

Gradient-based methods utilize gradients to optimize the policy directly. The goal of this method is to find a policy that maximizes the expected cumulative reward by iteratively updating the policy in the direction of the steepest ascent. Commonly used Gradient-based methods include state-of-the-art policy optimization algorithm Proximal Policy Optimization (PPO) [SWD+17], Trust Region Policy Optimization (TRPO) [SLM+17].

There are hybrid methods that combine the advantages of both Policy-based and Value-based methods such as Advantage Actor-Critic (A2C) which employs an actor-critic framework. The critic network is used to estimate a value function and the policy gradient is computed based on an advantage function which is utilized to guide policy updates to select corresponding actions. The actor is responsible for selecting actions based on the current state which then adjusts its policy based on the feedback from the critic to improve future actions [SML+18].

Extending this line work, [HZAL18] proposed state-of-the-art SAC which incorporates stochastic policies, allowing the agent to explore and generate a diverse set of actions. The exploration is encouraged by an entropy regularizer term and discourages overly deterministic policies. By maximizing entropy, SAC seeks a policy that strikes a balance between exploration and exploitation which builds the base for the rest of the thesis and is discussed in detail in section 3.2.1.

3.2.1 Soft Actor-Critic

Soft Actor-Critic algorithm is a model-free, off-policy, actor-critic reinforcement learning method. While PPO, A2C and TRPO algorithms learn in an on-policy manner, i.e., they need new sets of samples for each policy update, SAC learns in an off-policy way, i.e. by using experience replay buffers to learn from past data. SAC is primarily applied to continuous action space environment. By incorporating a maximum entropy objective, the SAC algorithm addresses the exploration-exploitation trade-off. It is used as a regularization term in policy optimization to measure the degree of uncertainty in a policy distribution. Additionally, the entropy quantity prevents convergence to sub-optimal policies. The entropy H for a policy is defined as

$$H(\pi) = \mathbb{E}_\pi \left[-\log(\pi(\cdot \mid s_t)) \right]. \tag{3.13}$$

Maximum entropy term is incorporated into the standard RL objective, where the agent receives an additional bonus reward at each timestep proportional to the entropy of the policy at that timestep. As a result the modified RL objective [HZAL18] is given as,

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t, a_t) + \alpha H(\pi(\cdot \mid s_t)) \right) \right]. \tag{3.14}$$

The temperature parameter α controls the stochasticity of the optimal policy and is used to balance entropy regularization terms. As a result of adding the entropy parameter in the modified objective, the state value and state-action value functions are defined as follows,

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t \left(r(s_t, a_t) + \alpha H(\pi(\cdot \mid s_t)) \right) \right], \tag{3.15}$$

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) + \alpha \sum_{t=1}^{\infty} \gamma^t H(\pi(\cdot \mid s_t)) \right]. \tag{3.16}$$

Finally, the modified Bellman equation for the state-action value function is given as,

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[r(s, a) + \gamma(Q^\pi(s', a') + \alpha H(\pi(\cdot \mid s'))) \right]. \tag{3.17}$$

SAC simultaneously learns a policy and two Q-functions to improve policy optimization. Two Q-functions are learned independently and the minimum value between the Q-functions is utilized for both improved policy update and the accurate target value estimation. Also, the two Q-functions provide multiple value estimates, which is helpful in guiding policy learning, and entropy regularization which encourages policy to explore diverse sets of actions, thereby promoting exploration. SAC utilizes clipped double-Q trick [FHM18] to reduce overestimation of biases for the two Q-functions. A minimum Q-value estimate between two Q-functions is used by SAC when estimating the target value for updating the Q-functions. As a result, the overestimation bias is mitigated and accurate value estimates are provided. During the policy update, SAC utilizes the maximum of Q-function estimates, encouraging the policy to choose actions with higher Q-values.

SAC trains the Q-network by minimizing the mean squared error between Q-value estimate $Q_{\theta_i}(s, a)$ which is given as

$$J_Q(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[(Q_\theta(s,a) - \hat{Q}(s,a))^2\right] \tag{3.18}$$

with the entropy regularized Bellman update target is given as

$$\hat{Q}(s,a) = r(s,a) + \gamma(\min_{\theta_{1,2}} Q_{\theta_i'}(s', a') - \alpha \log \pi(\cdot \mid s')) \tag{3.19}$$

where $a' \sim \pi(\cdot \mid s')$ and $\log \pi(\cdot \mid s')$ term approximates the entropy of the policy, and $\mathcal{D}$ is the replay buffer storing samples of the agent during training. The term $\min_{\theta_{1,2}} Q_{\theta_i'}(s', a')$ takes the minimum of Q-value network estimate between the two target Q-value networks $Q_{\theta_1'}$ and $Q_{\theta_2'}$ for the next state and action pair.

Given the stochastic nature of the policy in SAC, the policy objective is formulated so as to maximize the likelihood of actions $a \sim \pi(\cdot \mid s)$ that would result in high Q-value estimate $Q(s, a)$. The policy's objective function can thus be defined as

$$\max J_\pi(\phi) = \mathbb{E}_{s\sim\mathcal{D}}\left[\min_{\theta_{1,2}} Q_{\theta_i}(s,a) - \alpha \log \pi_\phi(a \mid s)\right] \tag{3.20}$$

3.3 Model-Based Reinforcement Learning

Following the discussion of MFRL algorithms in the previous sections, this section discusses MBRL upon which the rest of the thesis is built upon. In MBRL approaches, an agent explicitly learns a probabilistic model of the environment's dynamics. Various techniques can be used to build the model, such as stochastic models, regression models, and neural networks. This learned model can later be leveraged to solve various RL problems. [CCML18] propose a probabilistic ensemble with trajectory sampling, where Cross Entropy Method (CEM) method is utilized to sample actions from a distribution that is similar to previous action samples that yielded high rewards. [SHM+16] uses this Monte Carlo Tree Search (MCTS) methodology in the game of AplhaGo. Each node in the tree corresponds to a state which will be evaluated based on the returns with the help of a random policy.

An alternate strategy is that model can be used to generate simulated data for policy learning or value approximation. In this line of work, [FWS+18] proposed Model Value Expansion (MVE) which utilizes n-step temporal difference targets and depends on a fixed horizon. [BHT+19] proposed Stochastic Ensemble Value Expansion (STEVE) which interpolates between different horizons based on the uncertainty calculated using the ensemble networks.

The model can be used for data augmentation to improve the policy. [LXL+21] proposed Stochastic Lower Bound Optimization (SLBO) which uses a multi-step L2-norm loss to train the dynamics. SLBO uses the model to rollout full-length trajectories from the start state. However, long rollouts typically result in compounding prediction errors, which ultimately hinder the policy optimization process. [JFZL21] introduced MBPO, which makes use of shorter rollout lengths to avoid large compounding errors. MBPO begins to roll out from states which are sampled from the real environment and run n-steps according to the policy and the learned model. MBPO uses the model-generated data as a replay buffer to train a standard SAC agent. As such, MBPO can be considered a natural extension of SAC that incorporates model rollouts. SAC is utilized as an agent to update the policy with mixed data from both real and learned models. MBPO is discussed in detail in section 3.3.1 and its respective hyperparameters are in section 3.3.2.

3.3.1 Model-Based Policy Optimization

The natural choice of this model-based algorithm to further investigate is that it is a natural extension of model-free counterpart SAC. In this approach, the learned model is used to gather imaginary data to train a policy. To gather this imaginary data, a bootstrap ensemble of dynamics model is utilized where each member of the ensemble is a probabilistic neural network. Using probabilistic neural networks, aleatoric uncertainty, or noise in outputs relative to inputs, is captured while bootstrapping ensures to capture of epistemic uncertainty or uncertainty in model parameters. To generate a prediction from the ensemble, a random model is sampled from the ensemble of the dynamics model to test different transitions along a single model rollout.

Policy optimization is performed using SAC which is adopted from open-repository mbrl-lib [PAZ+21]. SAC alternates between a policy evaluation step, which estimates the state-action value function, and a policy improvement step. For all the experiments, we use the automatic entropy tuning flag that adaptively modifies the entropy gain based on the stochasticity of the policy.

Performing extended model rollouts leads to compounding error problems. To mitigate this problem, the MBPO algorithm begins propagating short model rollouts from state distribution of a different policy under the true environment dynamics. As a result, a few long trajectories with large errors can be traded for many short trajectories with smaller errors. The Figure 3.2 shows the architecture of a MBPO agent. It consists of two replay buffers for storing real experiences from the stochastic environment and generated experiences from the predictive model. Off-policy SAC is utilized as a policy optimization algorithm. SAC is designed using actor-critic framework which is constructed using neural networks. The actor-network consists of 2 hidden layers of 64 units each with a Tanh activation function. Similarly, the critic network also consists of 2 hidden layers of 256 units each with a Tanh activation function. The predictive model is represented as an ensemble of probabilistic neural networks whose outputs parameterize a Gaussian distribution. Each individual network of the ensemble consists of 4 hidden layers of 200 units each with SiLU activation [EUD17].

Figure 3.3 shows the data collection process in MBPO algorithm. Random models are uniformly sampled for a single model rollout step from the ensemble to generate a transition prediction. These transition predictions are later stored in the model replay buffer which are further utilized as inputs for the SAC algorithm for policy optimization.

Algorithm 3.1 shows the working of the Model-Based Policy Optimization algorithm.

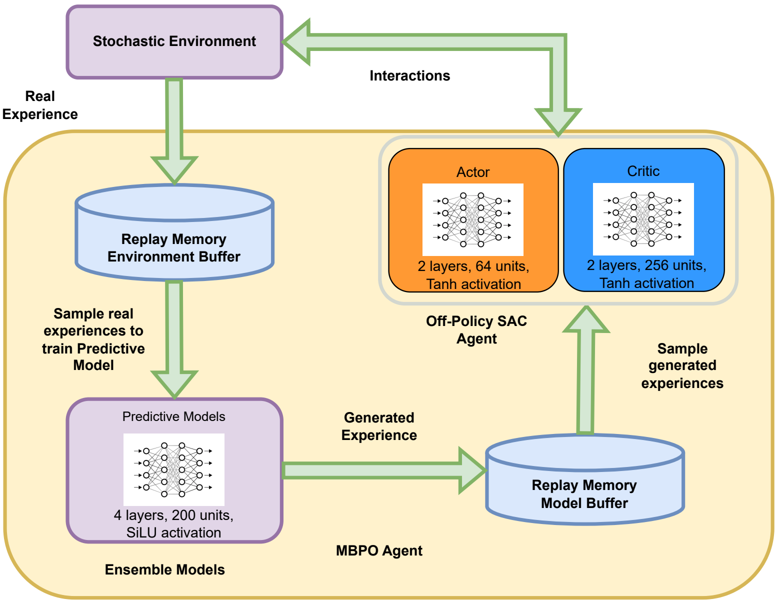

Stochastic Environment
Interactions
Real Experience
Replay Memory Environment Buffer
Actor
2 layers, 64 units, Tanh activation
Critic
2 layers, 256 units, Tanh activation
Off-Policy SAC Agent
Sample real experiences to train Predictive Model
Sample generated experiences
Predictive Models
4 layers, 200 units, SiLU activation
Generated Experience
Replay Memory Model Buffer
MBPO Agent
Ensemble Models

Figure 3.2: Detailed architecture depicting the individual components of an MBPO agent, which utilizes SAC as a policy optimization algorithm. The Figure depicts the interaction of the SAC agent with the stochastic environment to collect real experiences, which are later utilized to train a predictive model to generate a simulated experience. This simulated experience is later utilized as input to the SAC algorithm for policy optimization.

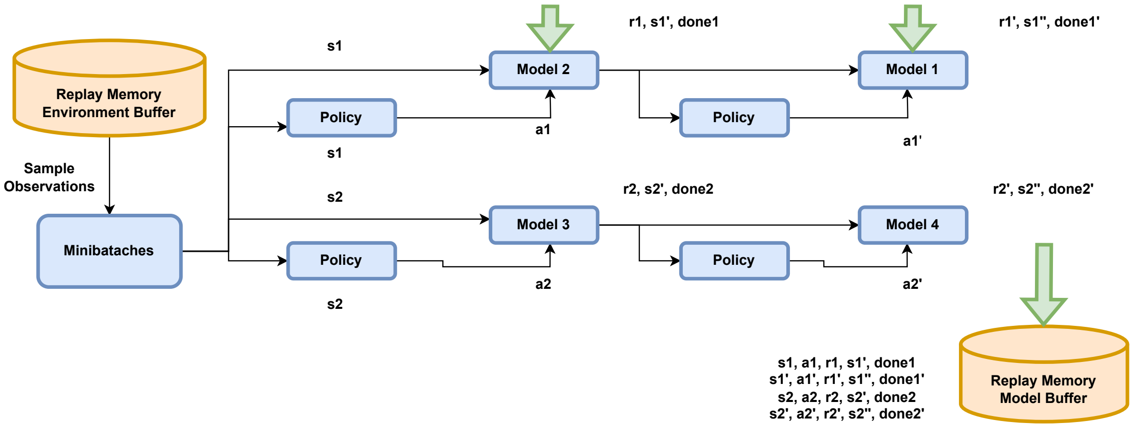


Figure 3.3: Experiences collected in the environment replay buffer are sampled as mini-batches. To generate a transition prediction, a model is uniformly sampled at random from the ensemble and the predicted experiences are stored inside the model replay buffer.

Algorithm 3.1 Model Based Policy Optimization algorithm

```
Initialize policy $\pi_\phi$, predictive model $p_\theta$, critic ensemble $\{Q_i\}_{i=1}^2$, environment buffer $\mathcal{D}_t$,
model buffer $\mathcal{D}_{model}$
$globalstep \leftarrow 0$
for episode $t = 0, ..., T - 1$ do
    for $E$ steps do
        if global step $\%F == 0$ then
            Train model $p_\theta$ on $\mathcal{D}_t$ via maximum likelihood
            for $M$ model rollouts do
                Perform $k$-step model rollouts starting from $s \sim \mathcal{D}_t$; add to $\mathcal{D}_{model}$
        Take action in environment according to $\pi_\phi$; add to $\mathcal{D}_t$
        for $G$ gradient updates do
            update $\{Q_i\}_{i=1}^2$ with mini-batches from $\mathcal{D}_{model}$, via SGD (11)
            update $\pi_\phi$ with mini-batches from $\mathcal{D}_{model}$, via SGA on optimistic Q-values of (9)
    $globalstep \leftarrow globalstep + 1$
```

3.3.2 Hyperparameters

Table 3.1 provides the main hyperparameters utilized for the continuous learning environments. This section further discusses specific hyperparameters which play an important role in defining the capacity of model-generated buffers and are also later used to conduct further ablation studies.

Model rollout length k, refers to the number of steps taken by the agent during the rollout phase. A rollout is a simulated sequence of actions taken by the agent, starting from an initial state and following a particular policy until a terminal state or a predefined time horizon is reached. Rollout length determines the length of the prediction horizon generated by the model to estimate the expected returns of various policies.

Model rollouts per step M, refers to the number of rollouts generated by a model per training step. This hyperparameter controls the number of rollouts executed under the model, it has a direct control on the amount of data generated by the model. It gives us the control of gathering the number of real environment rollouts and simulated rollouts. The increasing number of rollouts per step allows for gathering more simulated experiences, thereby improving the quality of data and the accuracy of the agent's policy.

Frequency of model retrain F, refers to how frequently the model is retrained during the training process. This hyperparameter determines the frequency at which the model parameters are updated to improve the performance of the model. Higher frequency retraining suggests that the model is retrained more frequently to adapt to the distribution of states observed in the true environment to improve the model's accuracy.

Updates to retain buffer R, refers to the number of model updates that are retained in the replay buffer. The replay buffer contains past experiences of the agent, typically consisting of state, action, reward, next-state, and done flags stored as tuples. The higher the value of this hyperparameter the more off-policy (old) data can be stored and sampled for training. Table 3.1 shows the list of important hyperparameters used in MBPO style learning in continuous control experiments.

Ensemble Size N, refers to the number of models or dynamics functions used in the ensemble. Each model present in the ensemble is a probabilistic neural network whose outputs parameterize a Gaussian distribution [JFZL21]

$$p_\theta^5 \left(s_{t+1}, r \mid s_t, a_t\right) = \mathcal{N}\left(\mu_\theta^5 \left(s_t, a_t\right), \Sigma_\theta^5 \left(s_t, a_t\right)\right)\Big)$$

, Ensemble models are responsible for capturing both aleatoric and epistemic uncertainties.

Trainer Patience p, refers to a parameter used for early stopping during model training. If the validation performance does not improve for consecutive epochs, training is halted to prevent further training on potentially overfitting the model.

Model-generated buffers are created by using a predictive model to generate additional training data for reinforcement learning algorithms. Model-generated buffers are used to augment the environment dataset and improve the agent's learning performance. This augmented dataset is utilized to simulate imaginary actions and outcomes, expanding the range of collected experiences. As a result, the capacity of the model-generated buffers $\mathcal{D}_{model}$ is computed as $k \times M \times F \times R$. The capacity of the model replay buffer is defined in such a manner as to hold R iterations of data generation.

Table 3.1: Hyperparameters for MBPO style learning

Hyperparameters	Acronyms
# episodes	T
# steps per episode	E
policy updates per step	G
# model rollouts per step	M
frequency of model retrain (#steps)	F
# updates to retain buffer	R
ensemble size	N
rollout length	k
Trainer Patience	p

3.3.3 MBPO-C Method

This section introduces Model Based Policy Optimization-Consistent (MBPO-C), a variant of the MBPO approach. Following the suggestion presented in [CCML18], instead of having a model being sampled at random for a rollout, MBPO-C uses a fixed model for the entire rollout and randomly samples states. Therefore, the rollouts are model consistent and we obtain a mean of the value distribution equal to the mean of the true value distribution. The key idea behind this approach is to leverage the diversity of multiple models to improve overall performance. By using different fixed models for each rollout, we can benefit from the strengths and weaknesses of each model. As a result, the ensemble of models helps reduce bias and increase the robustness of the estimated value. Figure 3.4 shows modified training loop of MBPO by having model consistent rollouts.

Algorithm 3.2 shows the working of the Model-Based Policy Optimization-Consistent algorithm.

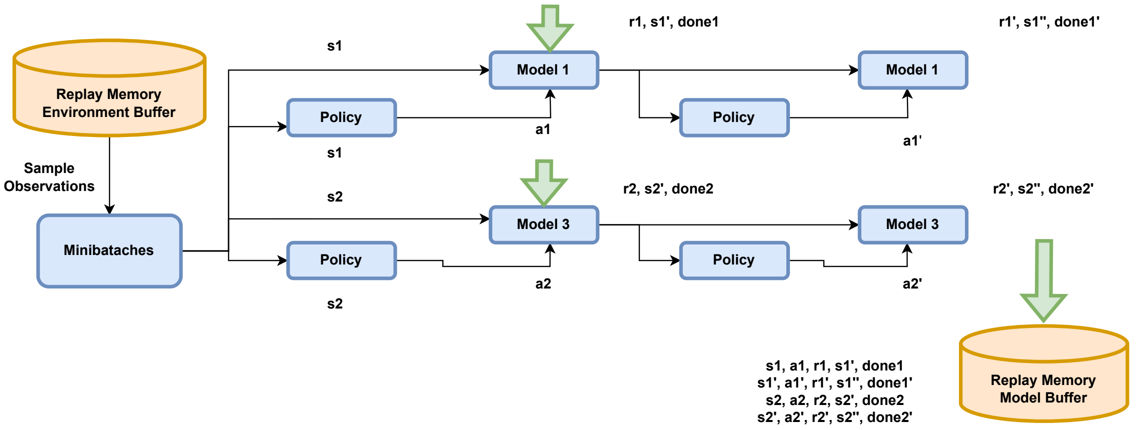


Figure 3.4: Experiences collected in the environment replay buffer are sampled as mini-batches. In comparison to the MBPO-styled rollout in Figure 3.3, A transition prediction is generated using a model which is uniformly sampled at random from the ensemble, and the same model is utilized throughout the entire rollout to generate predicted experiences. The predicted experiences using the model are stored inside the model replay buffer.

Algorithm 3.2 Model Based Policy Optimization-Consistent algorithm

```
Initialize policy $\pi_\phi$, predictive model $p_\theta$, critic ensemble $\{Q_i\}_{i=1}^2$, environment buffer $\mathcal{D}_t$,
model buffer $\mathcal{D}_{model}$
$globalstep \leftarrow 0$
for episode $t = 0, ..., T-1$ do
    for $E$ steps do
        if global step $\%F == 0$ then
            Train model $p_\theta$ on $\mathcal{D}_t$ via maximum likelihood
            Using fixed model M, perform $k$-step model rollouts starting from $s \sim \mathcal{D}_t$; add to
$\mathcal{D}_{model}$
        Take action in environment according to $\pi_\phi$; add to $\mathcal{D}_t$
        for $G$ gradient updates do
            update $\{Q_i\}_{i=1}^2$ with mini-batches from $\mathcal{D}_{model}$, via SGD (11)
            update $\pi_\phi$ with mini-batches from $\mathcal{D}_{model}$, via SGA on optimistic Q-values of (9)
    $globalstep \leftarrow globalstep + 1$
```

3.4 Integral Probability Metrics

This section gives a detailed introduction to a class of statistical metrics named IPM, which quantifies the discrepancy between two probability distributions. In this study, IPM's are used to compare the discrepancies between distributions of environment rewards and model imagined rewards, as well as empirical state distributions of both environment and the model. The main idea of incorporating the metrics is to study the impact of the different model-based hyperparameters on the model-generated data and measure the discrepancy between the model-generated distribution and the environment distribution of observations and rewards.

IPM provide a unified framework to compare distributions without making specific assumptions about their parametric form. For example, Let us consider two empirical distributions x and y over a common space and a distance function D which is an IPM. This distance function is considered a metric when properties such as non-negativity, positive definiteness, and symmetry are satisfied. The conditions are given below:

$$D(x, y) \geq 0 \;\; \text{(non-negativity)}, \tag{3.21}$$

$$D(x, y) = 0 \;\; \text{(3.21 and 3.22 together produce positive definiteness)}, \tag{3.22}$$

$$D(x, y) = d(y, x) \;\; \text{(symmetry)}, \tag{3.23}$$

This study introduces two IPM metrics [SFG+09] namely Wasserstein Distance (WD), which measures the discrepancy between distributions of environment and model rewards, and Maximum Mean Discrepancy (MMD), which measures the discrepancy between empirical state distributions of the environment and the model.

3.4.1 Wasserstein Distance

Wasserstein distance, also known as the Earth Mover Distance (EMD), is a distance metric that measures the dissimilarity between two probability distributions. The metric quantifies the minimum cost which is required to transform one distribution into another. The cost of the metric is defined as the amount of mass that needs to be moved from each point in one distribution to its corresponding point in the other distribution. Mathematically, If P is an empirical probability distribution of rewards from environment dataset $X_1, ..., X_n$ and Q is an empirical probability distribution of rewards from model dataset $Y_1, ..., Y_n$ of the same size, then the Wasserstein distance between the two distributions is calculated as follows:

$$W_p(P, Q) = \left[\sum_{i=1}^{n} ||X_i - Y_i||^p \right]^{(1/p)} \tag{3.24}$$

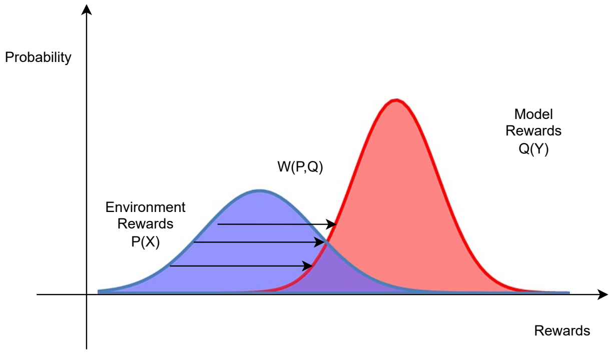


Figure 3.5: The plot depicts the Wasserstein distance calculation between environment reward distribution P(X) and Model reward distribution Q(Y). W(P,Q) quantifies the minimum cost which is required to transform one distribution into another.

In this study, the Wasserstein distance metric is particularly used for comparing reward distributions between the model and the environment. The reward distributions are one-dimensional and the metric provides a way to capture the underlying structure and spatial relationships between two types of reward distributions. The distance metric is calculated using the Python open-source library, SciPy [VGO+20]. The Figure 3.5 shows a visual representation of how the Wasserstein distance is calculated between two reward distributions.

The Wasserstein distance metric can be extended to higher dimensional state space, by considering the amount of mass that needs to be moved in a multidimensional space. The distance metric measures the minimum cost required to transform one distribution into another. It is done by moving the appropriate amount of mass in a way that preserves the overall mass and as a result, minimizes the total distance. In practice, computing the exact Wasserstein distance for higher dimensional distributions can be computationally intensive due to the optimization involved. As the dimensionality increases, the number of required samples to accurately estimate the Wasserstein distance also grows exponentially. High-dimensional spaces suffer from the curse of dimensionality, where the volume of the space increases exponentially with the number of dimensions. This makes it more challenging to sample from and represent distributions accurately, potentially leading to distorted distance measures. To address the problem of measuring distance in higher dimensional spaces, subsection 3.4.2 introduces MMD distance metric to measure the distance between higher dimensional distributions.

3.4.2 Maximum Mean Discrepancy Distance

MMD is a kernel-based metric, that quantifies the discrepancy between distributions as distances between mean embeddings of features. Moreover, under certain conditions, MMD can be zero if the two distributions are identical. MMD metric does not assume any specific parametric form of the distribution being compared. MMD metric maps the samples from each distribution into a higher-dimensional space using a feature map and compares the maximum means of the distributions. A smaller MMD value indicates a higher similarity between the distributions and vice-versa.

Given two probability distributions, if P is an empirical probability distribution of state space from environment dataset $X_1, ..., X_n$ and Q is an empirical probability distribution of observation space from model dataset $Y_1, ..., Y_n$, and assuming we have samples drawn from each distribution, mathematically the MMD between P and Q is computed as follows:

First, MMD is defined by a feature map for environment observation space $\psi : \mathcal{X} \rightarrow \mathcal{H}$, where $\mathcal{H}$ is called as Reproducing Kernel Hilbert Space (RKHS) [Gre13]. MMD starts by mapping the samples from the original input space to a higher dimensional feature space. These feature maps are spaces of functions that satisfy the reproducing property: $\langle f, \psi(x) \rangle_{\mathcal{H}} = f(x)$ for any $f \in \mathcal{H}$

Second, a kernel function is defined to measure the similarity between pairs of samples in the observation space. The kernel function compares the representations of the mapped samples. This combined kernel function for both environment and model observations can be represented as $K(x, y) = \langle \psi(x), \psi(y) \rangle_{\mathcal{H}}$. Common kernel functions include the Gaussian kernel, polynomial kernel, and Laplacian kernel. The Gaussian kernel and Laplacian kernel are defined respectively in equations 3.7a and 3.7b,

$$K(x, y) = e^{-\frac{||x-y||^2}{2\sigma^2}}, \quad \sigma > 0, \tag{3.25}$$

$$F(x, y) = e^{-\frac{||x-y||}{2\sigma^2}}, \quad \sigma > 0, \tag{3.26}$$

Figure 3.7 shows a comparative study using Gaussian and Laplacian kernel function for calculating the MMD distance. The study was performed between the model and environment observation dataset from the Sparse Inverted Pendulum environment. Both types of kernel functions can be utilized since both kernel functions return an unbounded distance value. In general, the Gaussian kernel results in smaller distance values. Since the squared Euclidean distance in the numerator leads to faster decay as the distance between data points increases. On the other hand, the Laplacian kernel with the absolute value of the Euclidean distance decays more slowly, resulting in larger distance values. As a result Gaussian kernel is utilized in the MMD distance metric to compute the similarity between mapped representations.

Finally, the MMD distance is computed based on the differences and similarities between samples within each distribution and across the distributions.

The first term of the MMD, captures the similarity within the environment state space The term calculates the average pairwise kernel similarity between samples belonging to environment state space. The goal is to capture the inherent structures and patterns in each distribution individually. Similarly, the second term of the MMD captures the similarity within the model state space. The first and second terms are correspondingly given as follows:

$$\langle \mathbb{E}_{X\sim P}\psi(X), \mathbb{E}_{X'\sim P}\psi(X')\rangle_{\mathcal{H}}, \tag{3.27}$$

$$\langle \mathbb{E}_{Y\sim Q}\psi(Y), \mathbb{E}_{Y'\sim Q}\psi(Y')\rangle_{\mathcal{H}}, \tag{3.28}$$

The third term measures the dissimilarity between samples across the environment and model state space distributions. This term captures the differences or discrepancies between the distributions and is given as follows:

$$\langle \mathbb{E}_{X\sim P}\psi(X), \mathbb{E}_{Y\sim Q}\psi(Y)\rangle_{\mathcal{H}}, \tag{3.29}$$

Finally, the MMD distance is calculated by combining equations 4.6, 4.7, and 4.8 which gives an overall measure of the difference between the distributions. The weights in front of each term ensure that the contributions from each term are properly balanced and finally, the combined equation is given as follows:

$$\begin{aligned} MMD^2(P,Q) = {} & \langle \mathbb{E}_{X\sim P}\psi(X), \mathbb{E}_{X'\sim P}\psi(X')\rangle_{\mathcal{H}} + \langle \mathbb{E}_{Y\sim Q}\psi(Y), \mathbb{E}_{Y'\sim Q}\psi(Y')\rangle_{\mathcal{H}} \\ & - 2\langle \mathbb{E}_{X\sim P}\psi(X), \mathbb{E}_{Y\sim Q}\psi(Y)\rangle_{\mathcal{H}}, \end{aligned} \tag{3.30}$$

The above equation can be simplified as follows

$$MMD^2(P,Q) = ||\mathbb{E}_{X\sim P}\psi(X) - \mathbb{E}_{Y\sim Q}\psi(Y)||^2_{\mathcal{H}} \tag{3.31}$$

Figure 3.6 shows a visual representation of how the MMD is calculated by transforming the observations to RKHS.

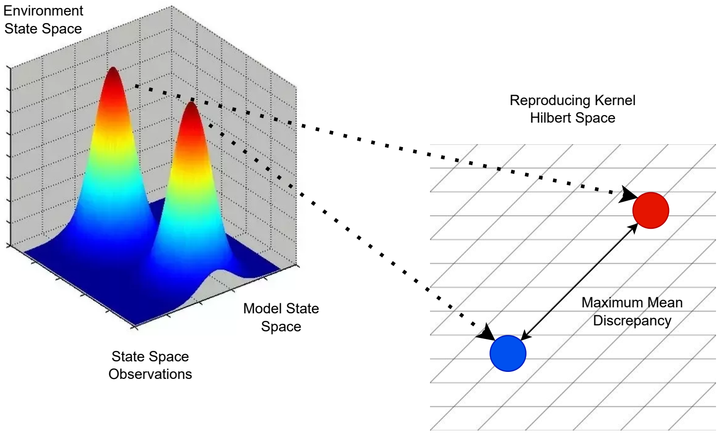
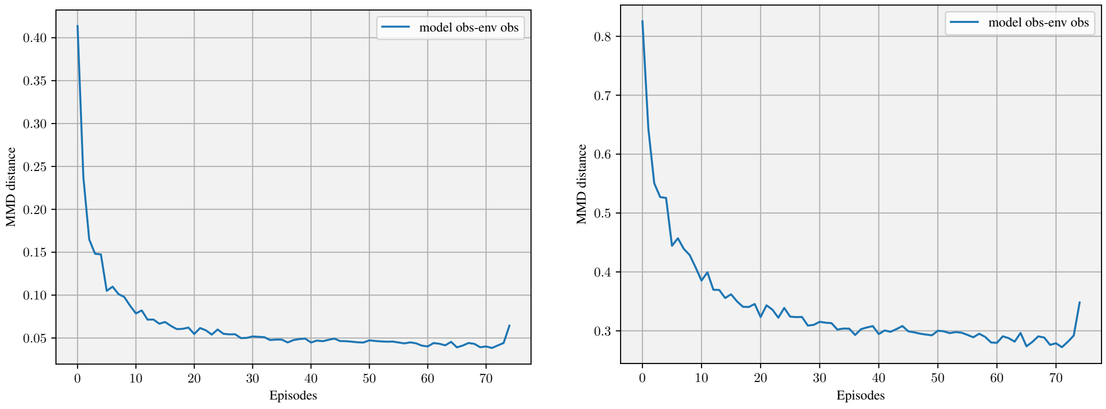


Figure 3.6: The figure depicts MMD distance calculation between Environment state space and Model state space. MMD metric maps the samples from each distribution into a higher-dimensional Reproducing Kernel Hilbert Space (RKHS) using a feature map and compares the maximum means of the distributions in the RKHS space.



(a) Gaussian Kernel

(b) Laplacian Kernel

Figure 3.7: Comparison of Gaussian and Laplacian kernel function applied to MMD distance metric for comparing the disparity between model observed states and environment states of the sparse pendulum environment.

4 Methodology

In the preceding chapters, we elucidated the elements of reinforcement learning, encompassing their classification into model-free and model-based approaches. Additionally, we provided an in-depth exploration of the various hyperparameters employed in the model-based setting. Furthermore, a comprehensive introduction to IPMs, which are subsequently utilized in this study to quantify the disparity between the distribution of observations and rewards in the model and the actual environment.

Following a thorough examination of existing literature, the primary research objective of this thesis is to identify the core components of MBRL algorithms and design specific experiments to understand the impact of such components in isolation under sparse reward scenarios. A non-exhaustive list of MBRL methodologies design choices includes the diversity of model-generated data, training the model for one-step or multi-step prediction, amount of model-generated data, frequency of retraining the model, length of rollouts and amount of off-policy data stored inside the model buffer. For this purpose, MBPO is chosen as the choice of the model-based algorithm for extensive investigation in this study. The primary benefit of opting for this model-based algorithm is its capacity to yield superior data efficiency and greater flexibility in adjusting hyperparameters, and the ability to operate in a manner that closely resembles a model-free setting. This analysis of design decisions aims to establish guiding principles for addressing challenges posed by sparse reward scenarios through the utilization of model-based approaches.

Integral probability metrics are employed to measure the impact of design choices. This is achieved by evaluating the discrepancy between the distribution of observations and rewards perceived by the model and the state and reward distributions in real-world environment. Specifically, the Wasserstein distance metric is employed to quantify the discrepancy between rewards observed by the model and the actual rewards distribution present in the environment. The maximum mean discrepancy distance metric is utilized to assess the dissimilarity between the state distribution as observed by the model and the state distribution in the actual environment. These established metrics assist in illuminating the model's interpretation of rewards and states when considering different components of the MBRL algorithm.

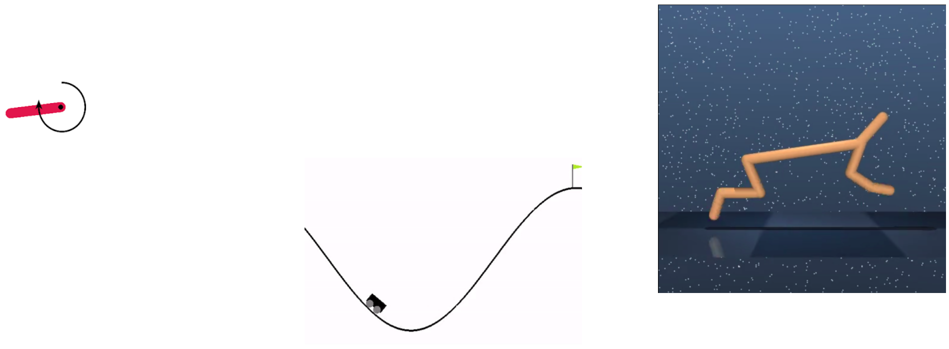
Moving forward, chapter 5 of this study outlines the experimental configuration, while chapter 6 presents a comprehensive examination and analysis of the experiments carried out to explore various design choices.

5 Experimental Setup

This section gives a brief introduction to continuous control environments, where the effect of hyperparameters is evaluated in continuous state-action spaces. In contrary to discrete control environments, where the agent has the option of choosing actions from a finite set of possibilities, continuous control environments allow the agent to choose from a broader set of action possibilities. In continuous control environments, actions are represented as continuous variables such as real numbers or vectors. Examples of continuous control environments which are used to train RL agents include OpenAI Gym [BCP+16a], and DeepMind Control Suite [TMD+20].

5.1 Hardware Specifications

The experiments designed are carried out in-house High Performance Cluster (HPC). The hardware configuration is as follows, Central Processing Unit (CPU) is Intel Xeon Gold 6148 and has the specification of the processor with 20 cores and 3.70 GHz at turbo performance and 2.40 GHz nominal performance. Graphical processing units (GPU) is NVIDIA Tesla V100 SXM and has a specification of 32 GB RAM and a memory bandwidth of 900 GB/s. The GPU consists of 5120 CUDA cores and 640 NVIDIA Tensor cores.



(a) Inverted Pendulum **(b)** Mountain Car Continuous **(c)** Cheetah Run

Figure 5.1: Illustration of continuous control environments where different design choices of MBPO algorithm are tested upon.

5.2 Inverted Pendulum

The Inverted Pendulum environment in which rewards are set up as sparse was proposed by [CBK20]. In this task, the pendulum is always initialized in the downward position with zero velocity and the goal of the RL agent is to apply appropriate torque to swing the pendulum to the upwards position. Figure 5.1a shows the inverted pendulum environment.

5.2.1 State Space

The state space of a sparse inverted pendulum environment is continuous and consists of the angular position of the pendulum given by the sine and cosine angles and its angular velocity. Table 5.3 shows the elements of the state space.

Table 5.1: State Space specifications of Sparse Inverted Pendulum environment

States	Min	Max
x = cos(angle)	-18.19	18.19
y = sin(angle)	-71.81	71.81
Angular Velocity	-0.5	0.5

5.2.2 Action Space

The action space in a sparse inverted pendulum environment is continuous. The actions of this environment represent the torque applied at the base of the pendulum. The action values range from -1.0 to 1.0, where -1.0 corresponds to maximum torque in one direction, 1.0 corresponds to maximum torque in the opposite direction, and 0.0 corresponds to no torque applied.

5.2.3 Rewards

The inverted pendulum environment is considered sparse because the agent is rewarded with a positive value given as $cosine\_angle\_tolerance \times velocity\_tolerance$ when the pendulum remains within both angle and velocity threshold. To complicate the problem, an action cost of value 0.1 is added, which is a cost proportional to the applied torque added to the state reward. This action cost counteracts the effect of exploration signals as a result the agent receives a penalized reward if the pendulum falls or deviates from the upright position. Action cost can be referred to as the cost associated with taking specific actions. The action cost is incorporated into the reward function thereby influencing the agent's behavior by encouraging actions that minimize the effect of the overall cost. This sparse pendulum setup along with action cost, makes the task challenging, as the agent needs to explore effective policies despite the negative signal coming from the action cost. The episode is terminated when the agent reaches 400 steps.

5.3 Mountain Car Continuous

The Mountain Car Continuous environment is part of classical control environments from OpenAI Gym [BCP+16b]. A car is stochastically placed at the bottom of a valley. The goal of the agent is to help the car reach the flag at the top of the hill by applying appropriate acceleration. Figure 5.1b shows the mountain car's continuous environment.

5.3.1 State Space

The state space of a mountain car's continuous environment consists of two variables: the position of the car along the x-axis and velocity along the x-axis. Table 5.2 shows the elements of the state space.

Table 5.2: State Space specifications of Continuous Mountain Car environment

States	Min	Max
Position of the car along the x-axis	-1.2	0.6
Velocity of the car	-0.07	0.07

5.3.2 Action Space

The action space of the Mountain Car environment is continuous and the agent can apply acceleration in the action value ranging from -1.0 to 1.0, where a value of -1.0 corresponds to full acceleration in the left direction, 1.0 corresponds to full acceleration in the right direction, and 0.0 corresponds to no acceleration. The action value is finally multiplied by a power of 0.0015

5.3.3 Rewards

The mountain car continuous task is considered solved when the car reaches a position greater than or equal to 0.45 at the top of the hill and receives a sparse reward of value 100. However, the agent receives a negative reward of $-0.1 \times action^2$, encouraging it to reach the goal with the minimum amount of effort. Each episode lasts 1000 steps.

$$r_t = \begin{cases} -0.1 \times action^2, & (position < 0.45), \\ 100, & (position \geq 0.45) \end{cases}$$

5.4 Cheetah Run

The Cheetah Run environment is a continuous control environment, where the goal is to train the agent to make the cheetah attain a forward maximum velocity and the task is to optimize the control policy to achieve high-speed running. This environment is a dense reward environment and a parallel study is conducted on this environment to observe if the same set of hyperparameters are responsible for the performance of the model-based agent in the dense reward setting.

5.4.1 State Space

The state space of the cheetah run consists of 17 dimensions, high dimensional in comparison with the previously introduced environment. The body position represents the overall position of the cheetah and velocity represents the rate at which the cheetah is moving in the environment. The position and the velocity of the cheetah are unbounded. Table Table 5.3, shows the elements of the state space.

Table 5.3: State Space specification of Cheetah Run environment

States
Body position of the Cheetah
Body Velocity of the Cheetah

5.4.2 Action Space

The action space is also continuous and 6 dimensional in size. The goal of the RL agent is to apply torque to actuate the cheetah's joints and make the cheetah run forward with the maximum velocity possible.

5.4.3 Rewards

The cheetah run task is completed when the agent is able to make the cheetah achieve a velocity of value greater than or equal to 10 and receives a reward of 1. If otherwise, the agent receives a reward of value between 0 and 1.

$$r_t = \begin{cases} 0.1 \times velocity, & (0 \leq velocity < 10), \\ 1, & (velocity \geq 10) \end{cases}$$

6 Evaluations and Analysis

In this section, we conduct several ablation studies to analyze the behavior of MBPO and, the experimental results are discussed in detail. Each experiment carried out was conducted with 20 random seeds to be able to properly evaluate the results. The experimentation carried out in this section was performed using SAC [HZAL18] as baseline model-free algorithm and MBPO [JFZL21] as baseline model-based algorithm. The evaluation return plots presented in this section are computed by collecting the evaluation returns across several runs, calculating the mean score and finally smoothening with a moving average filter of window size 5. The confidence interval regions are produced by calculating the standard error of the mean for the smoothened returns and plotting the mean with ±1 standard error. By calculating the confidence interval value, we can estimate the range of values within which the true population mean is likely to fall with a certain level of confidence. All the algorithms were implemented using MBRL-Lib [PAZ+21].

In section 6.1, we conduct experiments to evaluate the performance of SAC and MBPO agents in both $SparsePendulum - v0$ and $MountainCarContinuous - v0$ sparse reward environments. This study is conducted to compare how a model-free SAC agent and a model-based MBPO agent behave in sparse reward environments. In section 6.2, experiments with hyperparameter ablations were conducted on rollout length (k), model rollouts per step (M), updates to retain buffer (R), ensemble size (N), frequency retrain (F) and trainer patience. The goal is to examine why some hyperparameters work better than others through the lens of the dataset that is used for the training of the agent. The IPMs are utilized to better analyze the behavior of the agent during training. The hyperparameter ablation study was parallelly conducted on the MBPO-C agent, to evaluate the performance of this MBPO variant in sparse reward scenarios.

In section 6.3, a hyperparameter ablation study was performed in a dense reward $CheetahRun - v0$ environment to compare the behavior of the MBPO agent in higher dimensional dense reward settings. Finally, we conduct an experiment by specifically making the MBPO agent produce a performance close to the SAC agent in sparse reward scenarios. After reducing the performance, we conduct a specific hyperparameter study to understand which specific hyperparameter or a combination of hyperparameters is responsible for contributing to the performance boost in sparse reward scenarios.

6.1 SAC vs MBPO Performance

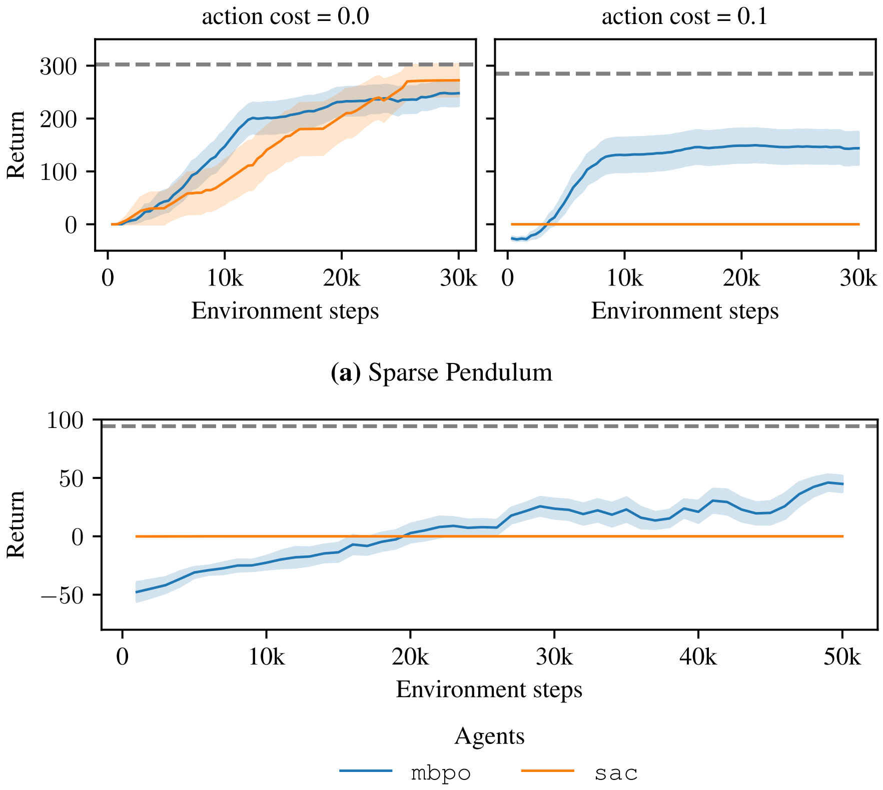
This section examines the behavior of model-free SAC and model-based MBPO agents in $SparsePendulum - v0$ and $MountainCarContinuous - v0$ sparse reward environments. In the aforementioned experiments, the MBPO agent uses the hyperparameter settings presented in Table 6.1. Model-free SAC agent uses the same configurations of policy network, Q-Network, and model network as the agent presented in Table 6.1, except for the model-based hyperparameters. We perform a comparative study to observe the effect of introducing the action cost and how both model-free and model-based agents manage to solve the sparse reward task in the environment.

The goal of introducing action cost in a sparse pendulum environment is to make the sparse reward problem more difficult to solve. An action cost refers to a penalty or negative value associated with taking a specific action in a particular state of the environment. Action costs are often used in scenarios where taking certain actions might have drawbacks that the agent should consider when making decisions. These costs can influence the agent's policy by encouraging it to choose actions that minimize the cumulative cost over time. A similar action cost was not introduced in a continuous mountain car environment since the environment has an action cost already implemented. The results of the experiment are seen in Figure 6.1b.

In figure 6.1a, In the plot where no action cost is applied, i.e. there is no penalty for exploring diverse actions that are not helpful in finding the reward. Model-free SAC and model-based MBPO agents were able to find the sparse reward and converge to an effective policy across a majority of the seeds. Another key observation from the plot is that a model-based agent uses less number of environmental steps to converge to an optimal policy, on the other hand, the model-free agent is less sample efficient.

In stark contrast, When action cost is applied to the sparse pendulum problem, i.e. awarding a penalty for performing explorations that are not helpful in finding the sparse reward. The SAC algorithm incorporates unstructured exploration to encourage the agent to explore new states and actions in the environment. Unstructured exploration in this context, refers to exploring state/action space in a more random and undirected manner, rather than relying solely on the policy optimization process. Through the combination of entropy regularization and exploration noise, SAC strikes a balance between structured exploration (encouraging diverse action selection) and unstructured exploration (encouraging random exploration in the action space. The results show us that the state-of-the-art model-free SAC algorithm struggles to explore and find the optimal or near-optimal policies in sparse reward scenarios because of the unstructured exploration.

In contrast, the model-based MBPO agent manages to find sparse rewards in nine of the twenty seeds launched. By utilizing the learned model, MBPO can explore via model rollouts which helps in effective exploration and convergence to optimal policies. A similar kind of comparative study was performed between model-free and model-based agents, on the challenging continuous version of the mountain car problem. It is evident from this study that the state-of-the-art SAC algorithm has limitations since it performs unstructured exploration by injecting noise into action selection, and that is insufficient for tasks such as continuous mountain cars and sparse pendulum, which require sustained exploration to find the solution [RKS21] [EHPM22]. On the other hand, MBPO solves the problem roughly for half of the seeds.



(a) Sparse Pendulum

(b) Mountain Car

Figure 6.1: (a) Learning curves of SAC and MBPO agents with and without action cost in sparse pendulum environment. (b) Learning curves of SAC and MBPO agents in a sparse continuous mountain car environment. Results presented show average returns over 20 random seeds which are smoothened by a moving average filter and we report the mean (solid lines) and standard error (shaded regions) and carried out using hyperparameter settings in Table 6.1.

6.2 Hyperparameters Ablation Study

In this section, MBPO hyperparameter ablation study was conducted for both $SparsePendulum-v0$ and $MountainCarContinuous-v0$ sparse reward environments. All the experiments were carried out by applying action cost = 0.1, and the results presented show average returns over 20 random seeds which are smoothened by a moving average filter and report the mean (solid lines) and standard error (shaded regions). The results of the ablation study along with the metrics are discussed in further subsections. We utilize the hyperparameter settings from Table 6.1, which were used to conduct the initial experiment in the sparse pendulum as shown in Figure 6.1a and the mountain car as shown in Figure 6.1b. We use the average return in both the sparse pendulum environment with action cost as a baseline for the remainder of the hyperparameter ablation study, and similarly in the case of the mountain car environment.

Table 6.1: MBPO Hyperparameter settings for continuous control experiments

Hyperparameter	Sparse Pendulum	Mountain Car	Cheetah Run
# environment steps	30e3	50e3	250e3
T - # episodes	75	50	250
E - # steps per episode	400	1000	
G - policy updates per step	20		10
M - # model rollouts per step	10		
F - frequency of model retrain (#steps)	400		250
R - # updates to retain buffer	1		10
N - ensemble size	5		
k - rollout length	10		5
trainer patience	1		5
learning rate	3e-4		1e-3
Policy network	2 layers, 64 units, Tanh activation		2 layers, 128 units, Tanh activation
Q network	2 layers, 256 units, Tanh activation		
Model network	4 layers, 200 units, SiLU activation		

The MMD distance metric plots presented in the hyperparameter ablation study indicate the distance between the distributions of environment and model state space. A higher MMD value indicates that the distributions are dissimilar, while an MMD value closer to zero indicates they are similar. Similarly, The Wasserstein distance metric plot shows the distance between the distributions of environment and model rewards. A higher distance value indicates the two distributions are different, and a value closer to zero indicates the distributions are similar. All the distance metric plots presented in the hyperparameter ablation study are clipped to a region where the distinction between the curves is clearly observable and as the training progresses all the values more or less remain constant.

6.2.1 Rollout Length Ablation

In this subsection, we present the results of an ablation study conducted on rollout length hyperparameter. The study was conducted using MBPO and MBPO-C style methods in the sparse pendulum and mountain car environments. We performed the experiments with $k = \{1, 3, 7 ,10\}$ and kept the rest of the hyperparameters fixed to values mentioned in Table 6.1.

Rollout length determines the length of the prediction horizon generated by the model to estimate the expected returns of various policies [BTZ22]. As a result, longer rollout length emphasizes the increased exploration of unfamiliar states and may lead to better policies but The accuracy of predictions typically decreases for longer rollouts due to distributional shift, i.e., we query the learned model out-of-distribution. On the other hand, a shorter rollout length emphasizes exploitation, which indicates that the agent focuses on immediate rewards and makes decisions based on its current state.

The results presented in Figure 6.2, the top row shows the rollout length ablation study performed in the sparse pendulum environment and the bottom row corresponds to the ablation study performed in the mountain car environment. We observe from the learning curves that a higher value for rollout length in both the sparse reward environments resulted in higher returns when compared to the returns of smaller values of rollout length. A rollout length of value 1 indicates that the model takes a single action, observes the resulting state and reward, and then makes the next decision. This leads to a lack of exploration because, in continuous control environments, the model often has numerous potential actions available for any given state. By focusing on just one action, the agent could overlook opportunities to explore alternative actions that might yield superior results over time. Furthermore, the agent could become trapped in local optima, a situation where it makes a narrow-minded choice that enhances its immediate reward but hinders its progress towards a more favorable state in the future.

We confirmed this hypothesis in the mountain car environment, where the learning curve with k=1 did not yield better returns compared to other rollout lengths with higher values. Interestingly, the sparse pendulum environment showed positive returns with a rollout length of 1. However, the hypothesis is still supported, as higher rollout lengths led to even greater returns. This suggests that other factors are at play in contributing to the positive returns observed in the sparse pendulum environment.

In the W-distance metric plots (center column), In the sparse pendulum environment, we can observe that the difference between the reward distributions generated by the model and the environment is more pronounced at the start of training. As training advances, the Wasserstein distance for various rollout lengths gradually converges. The reported results are truncated because this convergence behavior persisted throughout the remaining environment steps. In the mountain car environment, we notice that a greater rollout length led to the most significant disparity between model-generated and environment reward distributions. This consequently led to a higher overall Wasserstein distance. This trend mirrors the observation made in the sparse pendulum environment. It can be attributed to the fact that at the beginning of the training the model visits unfamiliar states with longer rollouts and receives rewards based on the action taken from that unfamiliar states and hence the discrepancy between the model-generated and environment distribution at the beginning of the training and the eventual convergence at the end of the training.

The results observed from the MMD distance metric plots (right column) indicate that higher rollout lengths produced maximum discrepancy between model-generated and environment observation state distributions for both sparse reward environments. As the training progressed the MMD distance value decreases, which indicates the distribution of states generated under the learned model approaches the true distribution of states prescribed by the environment dynamics. In both environments, a rollout length of 1 demonstrates an interesting behavior, wherein the MMD distance between the model-generated and environment observation state distributions increases as training progresses. The MMD distance value for k=1 represents a very small value, indicating that the model is operating in an on-policy manner, as the states visited by the model closely resemble the states visited in the environment.

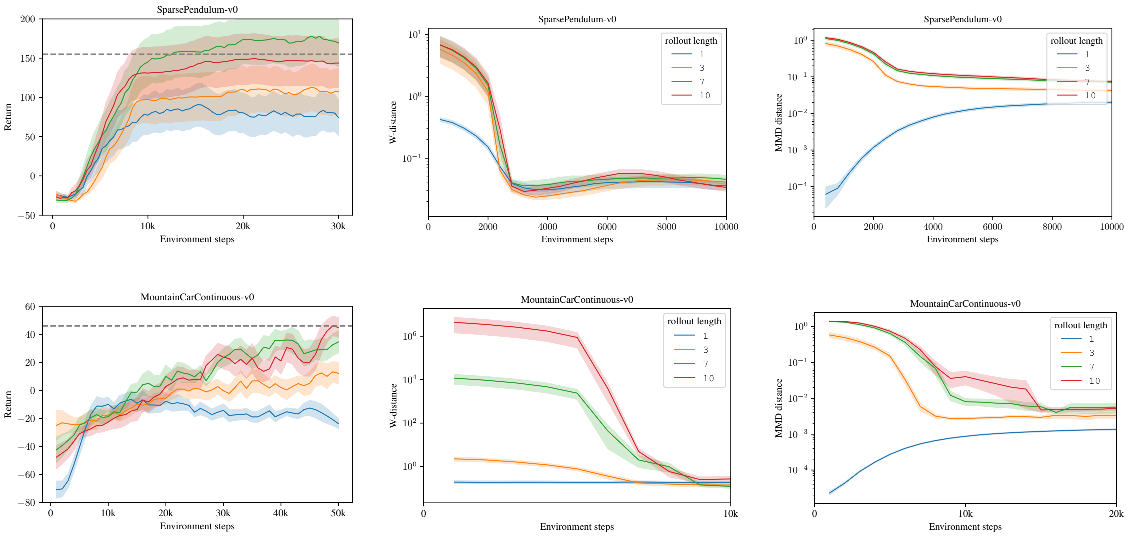


Figure 6.2: Ablation study over rollout length hyperparameter conducted using MBPO agent in $SparsePendulumv0$ (Top row) and $MountainCarContinuousv0$ (Bottom row) environment. (Left column) We report the learning curves, (Center column) corresponds to the Wasserstein distance metric, and (Right column) corresponds to the MMD distance metric of different rollout lengths. Results presented show average returns over 20 random seeds which are smoothened by a moving average filter and report the mean (Solid lines) and standard error (Shaded regions).

We extended the rollout length hyperparameter ablation study using the MBPO-C method, and Figure 6.3 presents the results. We observe similar behavior in the observation states visited and rewards received by the model, which is evident from the distance metric plots using the MBPO-C methodology. In the evaluation returns plots, we note that the overall returns in both sparse reward environments are low. The performance is notably lower than the established baseline standard. Moreover, in contrast to the MBPO methodology, there is not a substantial performance improvement observed, even though we conducted model-consistent rollouts based on specific models from the ensemble.

Figure 6.4 illustrates the experiment conducted on a sparse pendulum using an MBPO agent, visualizing the reward distribution throughout the training. The hyperparameters employed are outlined in Table 6.1. The plotted data illustrates the comparison between the model, the environment, and the true reward distribution during training along with their corresponding probability density. We passed model observations and actions stored within the model buffer through the reward function of the sparse pendulum environment to generate the plot depicting the true reward distribution. This plot provides a ground truth for comparing if the model is imagining seeing the rewards as and when the rewards start appearing in the real environment.

During episode 1, the agent visits a wide variety of states since the learned model is inaccurate, as a result, predictions are less accurate and hence the wider distribution of rewards. As the agent gathers more experience and updates its knowledge of the model, it can make more accurate predictions about the future rewards associated with different actions. As the training progresses, rewards start to appear in the real environment and since the agent has gained some experience during the period,

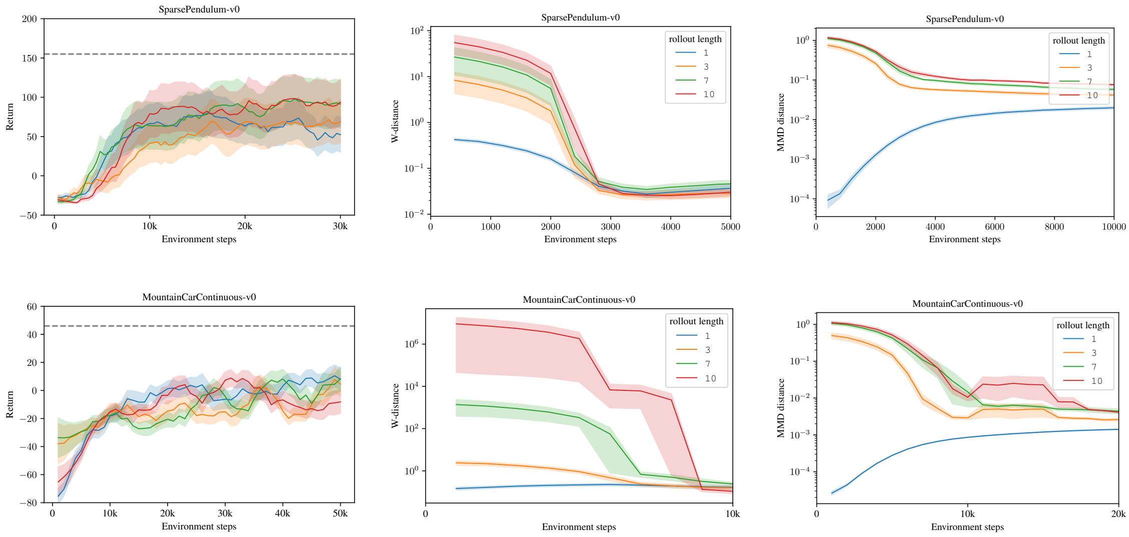


Figure 6.3: Ablation study over rollout length hyperparameter conducted using MBPO-C agent in $SparsePendulumv0$ (Top row) and $MountainCarContinuousv0$ (Bottom row) environment. (Left column) We report the learning curves, (Center column) corresponds to the Wasserstein distance metric, and (Right column) corresponds to the MMD distance metric of different rollout lengths. Results presented show average returns over 20 random seeds which are smoothened by a moving average filter and report the mean (Solid lines) and standard error (Shaded regions).

it starts to make less accurate predictions around the reward region. The probability distribution is wider since there is not enough data around the region of sparse reward. Starting from Episode 30, the agent accumulates experience around the reward region, causing the probability distribution to contract and become concentrated in the area associated with rewards. By comparing the model's perception of rewards during training with that of the environment and the true reward distribution, we can enhance our comprehension of how the model interprets rewards. In the rest of the study, we will investigate how different combinations of hyperparameters impact the model perceived states and reward distributions which is helpful in solving the sparse reward.

6.2.2 Rollouts per Step Ablation

In this subsection, we present the ablation study conducted on the number of rollouts per step hyperparameter. We conduct the study using MBPO and MBPO-C style methods in both sparse pendulum and continuous mountain car environments. The ablation study was carried out with M = $\{10, 50, 400, 1000\}$, and the rest of the hyperparameters were fixed to the base values presented in Table 6.1.

The number of rollouts per step hyperparameter controls the amount of data we generate using the model. In the experiments carried out in both environments, each epoch lasted for 400 environment steps. The hyperparameter signifies that M number of initial states are drawn from the environment buffer and k-step rollouts are performed from each of those M initial states.

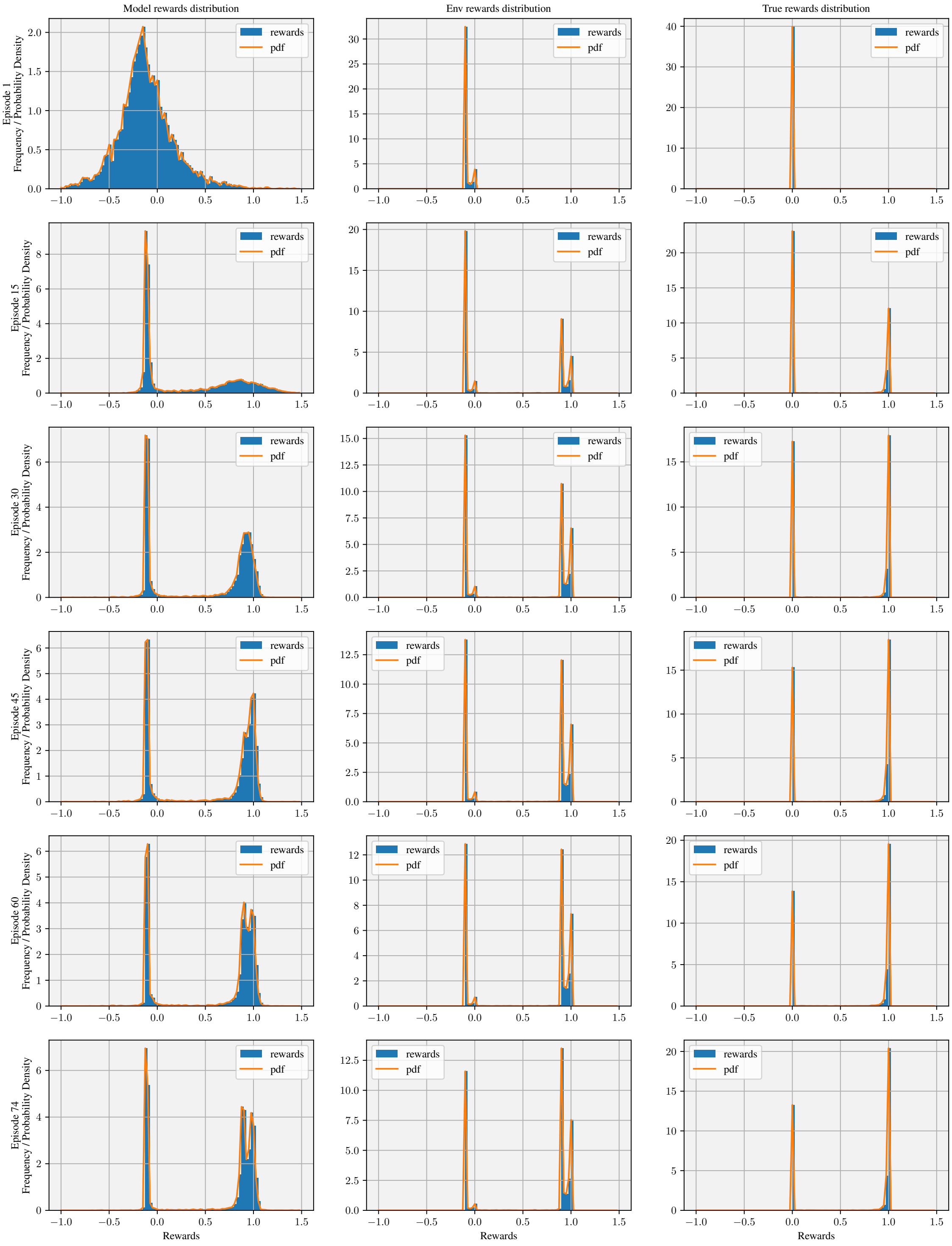


Figure 6.4: Visualization of reward distribution and *pdf* during training in sparse pendulum swing-up task using MBPO agent. (Left column) corresponds to the reward distribution predicted by the model. (Center column) corresponds to reward distribution as original rewards start appearing in the environment. (Right column) corresponds to true reward distribution where model observations and actions are passed to the reward function which serves as ground truth for comparison with environment rewards.

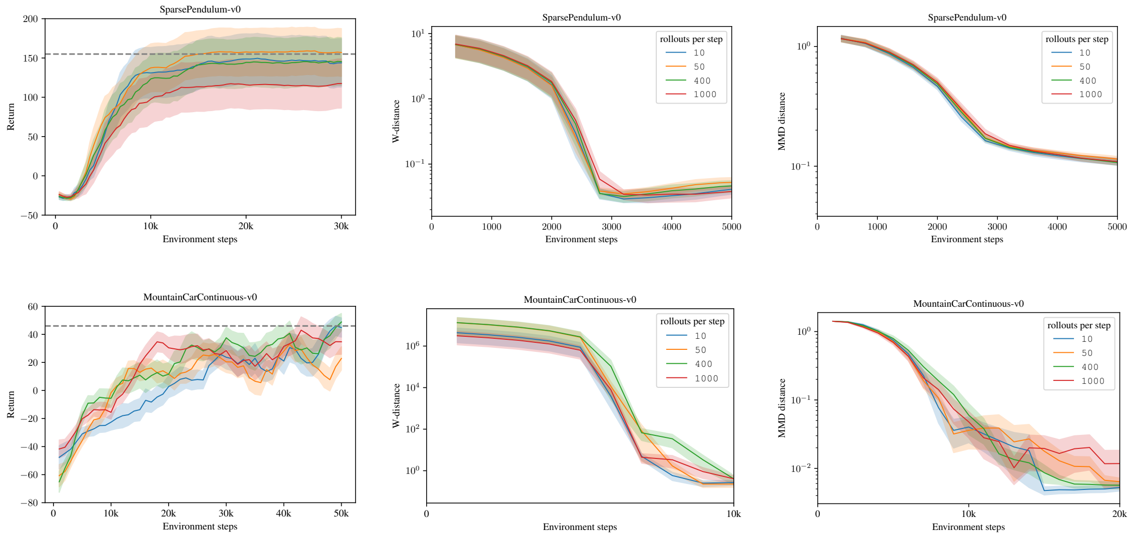


Figure 6.5: Ablation study over the number of rollouts per step hyperparameter conducted using MBPO agent in $SparsePendulumv0$ (Top row) and $MountainCarContinuousv0$ (Bottom row) environment. (Left column) We report the learning curves, (Center column) corresponds to the Wasserstein distance metric, and (Right column) corresponds to the MMD distance metric of different rollout lengths. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

Figure 6.5 illustrates the ablation study conducted in the sparse pendulum environment (Top row) and the mountain car environment (bottom row). In the sparse pendulum environment, we observe from the learning curves that having $M = 1000$ resulted in lesser returns in comparison with the rest of the values of the ablation studies. This behavior can be attributed to the fact that generating a large number of rollouts per step has the potential to introduce more noise and variance in the learning process or if the model becomes too complex, the agent may start to memorize specific trajectories rather than generalizing well to unseen data. In the mountain car environment, we can observe that with $M = 1000$, the model required fewer iterations to reach the maximum return. However, as training advanced, we identified a pattern similar to what was observed in the sparse pendulum environment.

From the Wasserstein distance metric plot (center column) and the MMD distance metric plot (right column), We can observe that regardless of the varying amounts of initial states sampled from the environment buffer, the difference between model-observed states and reward distributions resulted in almost identical distance metric curves as the training progressed. This indicates that the model is visiting the same sets of states and receiving the same set of rewards, regardless of the quantity of initial states that were sampled from the environment buffer.

Figure 6.6 presents the results of the extended ablation study conducted using MBPO-C methodology. From the results, we can infer that the overall returns in both sparse reward environments are lower compared to the baseline set. There is no significant improvement in performance observed. As for the distance metric plots, we notice a similar behavior to that observed using the MBPO methodology in the sparse pendulum environment. However, in the mountain car environment, a

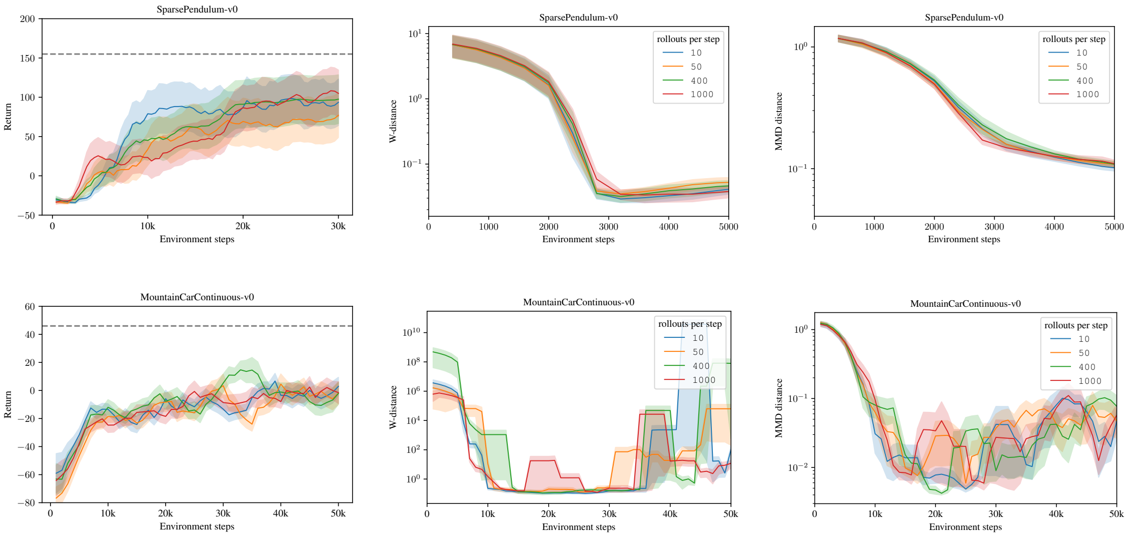


Figure 6.6: Ablation study over the number of rollouts per step hyperparameter conducted using MBPO-C agent in $SparsePendulumv0$ (Top row) and $MountainCarContinuousv0$ (Bottom row) environment. (Left column) We report the learning curves, (Center column) corresponds to the Wasserstein distance metric, and (Right column) corresponds to the MMD distance metric of different rollout lengths. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

distinct behavior is observed, where the distance metric plots fluctuate as training progresses. In the evaluation returns plot, it can be observed that employing a model-consistent rollout did not aid the model in achieving higher returns.

6.2.3 Updates to Retain Buffer Ablation

In this subsection, we present the results of the ablation study conducted on updates to the retain buffer hyperparameters. We performed the experiments with $R = \{1, 3, 7, 10\}$ while the remaining hyperparameters were kept fixed at the base values outlined in Table 6.1. The number of updates to retain the buffer determines the number of past experiences that can be restored inside the model buffer without overwriting it. Typically, a higher value of this hyperparameter empowers the model to learn from a wider range of off-policy data and has the potential to enhance performance. A value of $R = 1$ signifies that the buffer is frequently overwritten, and the model learns primarily from the most recent experiences. This limitation restricts the agent's capacity to learn from past experiences. Experiences within the buffer are overwritten in accordance with a first-in, first-out (FIFO) policy.

In the evaluation return plots depicted in Figure 6.7, It's evident that when setting the number of updates to retain buffer value at 1 in the sparse pendulum environment, the model operates in a manner closely resembling on-policy behavior. This led to the model achieving the highest performance compared to the other values in the ablation study. However, the presence of off-policy

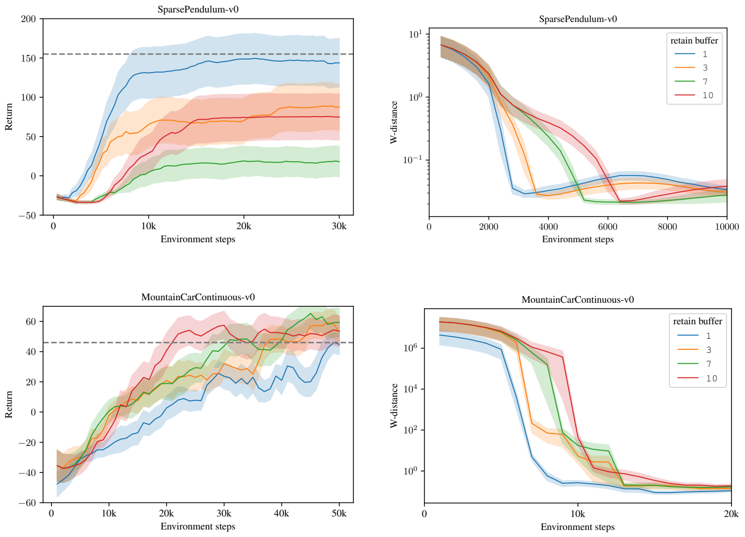


Figure 6.7: Ablation study over the number of updates to retain buffer hyperparameter conducted using MBPO agent in $SparsePendulumv0$ (Top row) and $MountainCarContinuousv0$ (Bottom row) environment. (Left column) We report the learning curves, (Center column) corresponds to the Wasserstein distance metric, and (Right column) corresponds to the MMD distance metric of different rollout lengths. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

data tends to impact the performance. Conversely, in the mountain car environment, greater availability of off-policy data proved advantageous. It indicates that retaining past information assisted the model in solving the complex mountain car environment.

This allows the agent to learn from a diverse set of experiences and can lead to more stable and faster learning which is clearly evident from the evaluation return plots. The continuous mountain car task might require exploring a large action space to find the optimal policy. By retaining diverse off-policy data, the agent can explore various actions and states that it might not have encountered recently. This enables the model to better explore the action space and discover effective policies more efficiently. Moreover, In the continuous mountain car environment, the model needs to learn how to apply the right amount of force to the car to overcome gravity and reach the goal. This requires fine-tuning actions based on subtle state changes. Retaining off-policy data helps the model to learn from past experiences where it may have successfully encountered similar situations, helping it learn effective strategies for exploration and exploitation.

In the Wasserstein distance metric plots (center column), We observe that in both sparse reward environments, at the onset of training, retaining any quantity of off-policy data maintained a relatively consistent discrepancy between the model-generated and environment reward distributions. As training advanced, a smaller number of updates to retain buffer resulted in a quicker reduction of this discrepancy. Likewise, we can observe a similar trend from the MMD distance metric plots (Right column), where initially, a notable discrepancy exists between the model-observed states and the environment states. As training advances, higher values of updates to retain buffer require a substantial number of environment steps to achieve convergence.

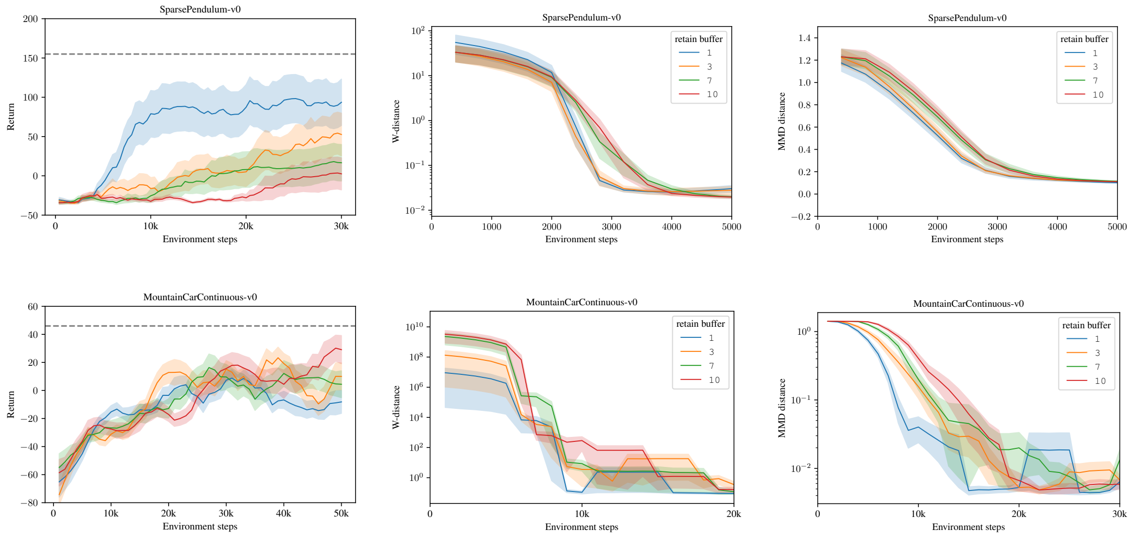


Figure 6.8: Ablation study over the number of updates to retain buffer hyperparameter conducted using MBPO-C agent in $SparsePendulumv0$ (Top row) and $MountainCarContinuousv0$ (Bottom row) environment. (Left column) We report the learning curves, (Center column) corresponds to the Wasserstein distance metric, and (Right column) corresponds to the MMD distance metric of different rollout lengths. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

We extended the ablation study of the number of updates to retain buffer using the MBPO-C methodology, as illustrated in Figure 6.8. We note a comparable trend in evaluation returns, reward distribution, and observation state distribution.

Taking into account the previous ablation studies using the MBPO-C methodology, it becomes apparent that no performance improvement was achieved in comparison with the MBPO methodology. The MBPO-C methodology did not yield any specific patterns, and both methodologies generated trajectories that exhibit comparable randomness. This is clearly evident from the distance metric plots, as they depict a very similar state distribution by the end. In the subsequent ablation study and further experiments, we exclusively focus on the MBPO methodology to continue our investigation.

6.2.4 Ensemble Size Ablation

In this subsection, we carried out an ablation study on the Ensemble Size hyperparameter. The ensemble size determines the number of neural network models present inside the ensemble. The experiments were conducted with N values of 1, 3, 7, 10, while the rest of the hyperparameters were maintained at the base values outlined in Table 6.1. The ablation study was performed using the MBPO methodology in both the sparse pendulum and mountain car environments.

Ensemble size refers to the number of models or dynamics functions used in the ensemble. Each model present in the ensemble is a probabilistic neural network whose outputs parameterize a Gaussian distribution. The idea behind ensemble is that by utilizing the predictions of multiple

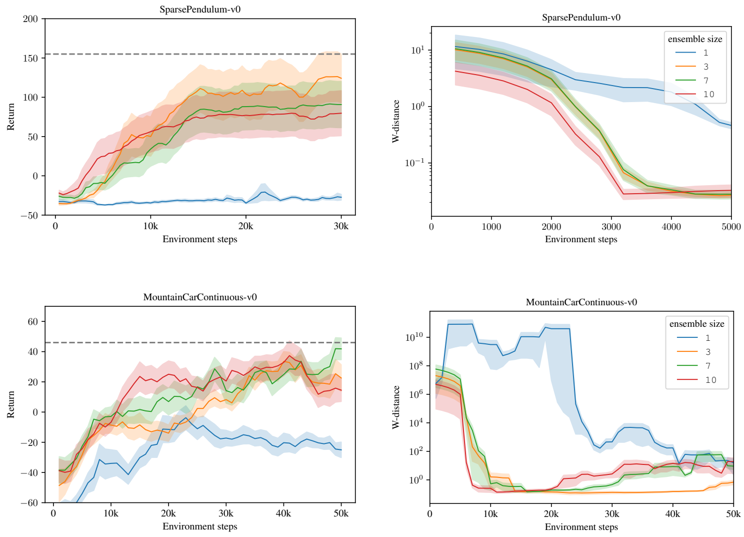


Figure 6.9: Ablation study over ensemble size hyperparameter conducted using MBPO agent in $SparsePendulumv0$ (Top row) and $MountainCarContinuousv0$ (Bottom row) environment. (Left column) We report the learning curves, (Center column) corresponds to the Wasserstein distance metric, and (Right column) corresponds to the MMD distance metric of different rollout lengths. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

models, you can often achieve better performance and more robust results compared to using a single model. In this study, a model is uniformly sampled from the ensemble of models for each rollout, and a k-step rollout is performed starting from the initial states sampled from the environment buffer. The experience tuple collected is later added to the model buffer.

In Figure 6.9, we can observe that an ensemble size of 1 failed to accurately capture the dynamics in both the sparse reward environments. If the ensemble size is 1, we essentially possess only a single neural network model, and there is no ensemble present. In this case, we rely solely on the capabilities of that single neural network to formulate predictions and learn from the data. The concepts and benefits of the ensemble, such as diversity, variance reduction, and improved generalization, would not be applicable as only one model is involved.

In the Wasserstein distance metric plots (Center column), we can infer from the results that the discrepancy between reward distributions remains relatively consistent for ensemble sizes greater than 1 in both the sparse reward environments. Interestingly, with an ensemble size of 1, the discrepancy between the model-generated and environment reward distribution never converges throughout the entire training.

The MMD distance metric plot (right column) and the Wasserstein distance metric both show a consistent trend: the model-observed states do not converge to the states observed in the environment throughout the entire training when we set the ensemble size to 1. This suggests that an ensemble size of 1 is incapable of capturing the genuine dynamics of the environment. An ensemble size greater than 1 captures the environment's dynamics and reduces discrepancies using fewer environment steps compared to smaller ensemble sizes.

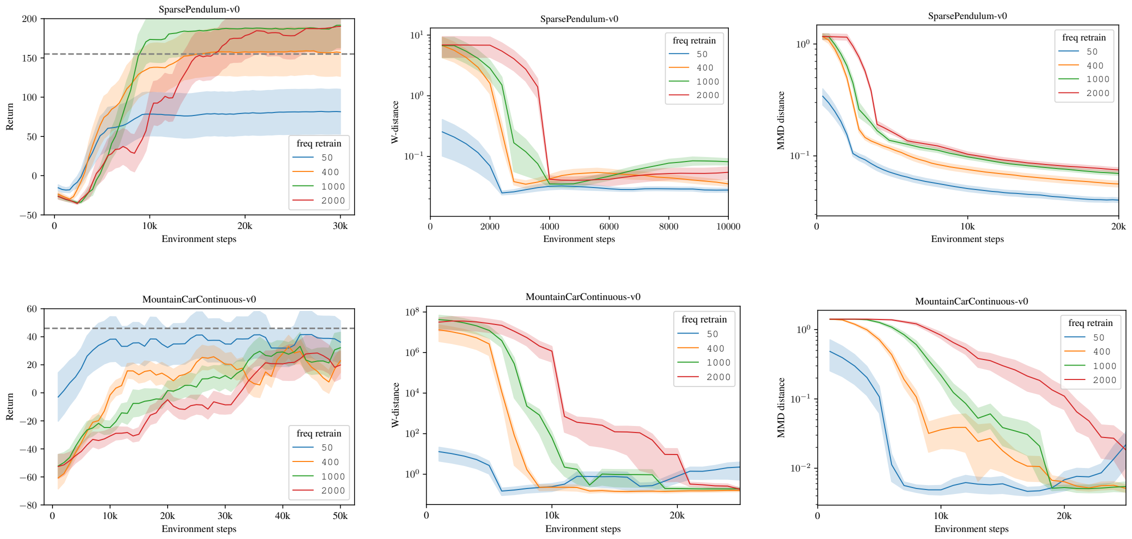


Figure 6.10: Ablation study over frequency retrain hyperparameter conducted using MBPO agent in $SparsePendulumv0$ (Top row) and $MountainCarContinuousv0$ (Bottom row) environment. (Left column) We report the learning curves, (Center column) corresponds to the Wasserstein distance metric, and (Right column) corresponds to the MMD distance metric of different rollout lengths. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

6.2.5 Frequency Retrain Ablation

In this subsection, an ablation study was carried out on the Frequency retrain hyperparameter. This hyperparameter specifies the number of batches to process before the model parameters are updated. For instance, if we set the frequency of retraining to 10, the model parameters will be updated after processing 10 batches of data. The experiments were conducted with F values of 50, 400, 1000, 2000, while the rest of the hyperparameters were maintained at the base values outlined in Table 6.1. The ablation study was performed using the MBPO methodology in both the sparse pendulum and mountain car environments.

Figure 6.10 illustrates that in the evaluation return plot (Left column), a higher frequency retrain value led to increased evaluation returns in the sparse pendulum environment. In contrast, for the continuous mountain car environment, a frequency retrain value of 50 yielded significantly higher evaluation returns compared to the other values examined in the ablation study. This indicates that reducing the frequency of updating the model parameters in the sparse pendulum environment aided the model in attaining higher returns. Conversely, in the intricate mountain car environment, achieving substantial returns necessitated more frequent updates to the model parameters.

In the Wasserstein distance metric plots (Center column), we notice that increasing the frequency of updating the model parameters aided in diminishing the disparity between model-generated rewards and environment rewards, requiring fewer environment steps. This trend was observed in both sparse reward environments. Similarly, we observed the same trend in the MMD distance metric plot (right column), suggesting that frequent updates of model parameters expedite the model's learning process.

6.2.6 Trainer Patience Ablation

In this subsection, we present the results of the ablation study that we conducted on the Trainer Patience hyperparameter. During training, the agent collects data from the environment iteratively, updates its model based on the collected data, and later employs the model to make decisions. The Patience hyperparameter becomes significant during the model update step. If the agent's performance does not show improvement for a specific number of consecutive epochs, as specified by the patience hyperparameter, the model training terminates to prevent potential overfitting. We conducted the experiments with values of $p = \{1, 3, 7, 10\}$ while keeping all other hyperparameters fixed at the base values outlined in Table 6.1. We carried out the ablation study on both the sparse pendulum and mountain car environments using the MBPO methodology.

In Figure 6.11, we observe that a smaller patience value in the sparse pendulum environment caused the training to conclude earlier as soon as notable improvements ceased. Allowing the model to train for an extended period without substantial enhancements resulted in a model that did not enhance the agent's performance. In the more challenging mountain car environment, employing a higher patience value permitted the training process to persist despite the absence of immediate improvements. This allowed the agent more time to explore and acquire a more refined model representation. With increased training iterations, the model successfully captured finer nuances and intricacies of the environment, resulting in enhanced predictions and more informed decision-making.

The distance metric plots reveal that employing a higher patience value leads to more similar distributions of rewards and observation states between the model and the environment. Conversely, a smaller patience value yields more dissimilar distributions, and as the training advances, the plots eventually converge to a lower distance. In the mountain car environment, the behavior of IPMs differs from that of the sparse pendulum environment. At the start of the training, a lower patience value yields more dissimilar reward and state space distributions. As the training progresses, the MMD distance diminishes for a brief period, and patience values greater than 1 cause an increase in dissimilarity between the model's state space distribution and the environment's.

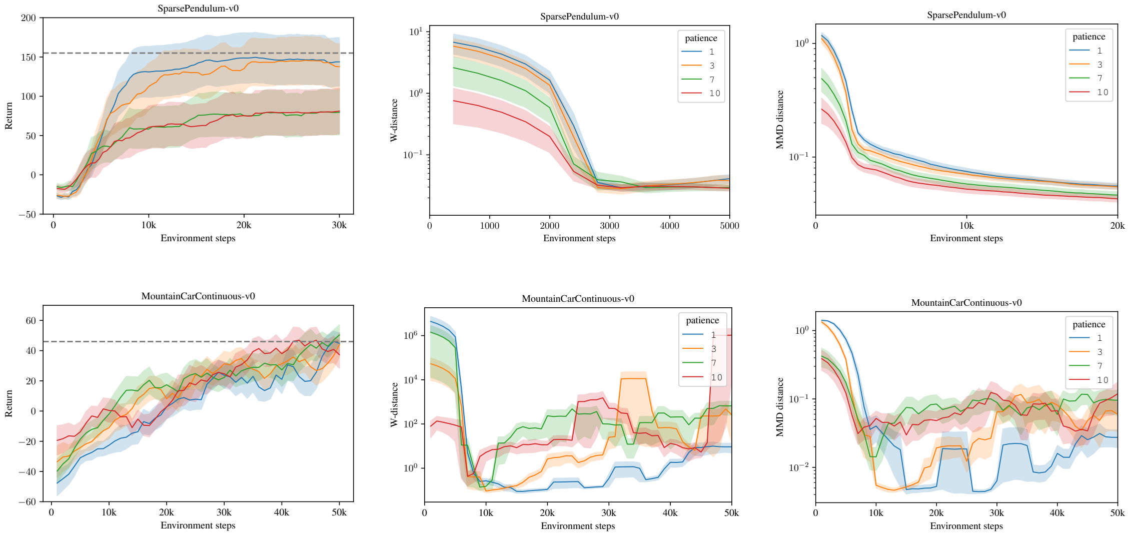


Figure 6.11: Ablation study over trainer patience hyperparameter conducted using MBPO agent in *SparsePendulumv0* (Top row) and *MountainCarContinuousv0* (Bottom row) environment. (Left column) We report the learning curves, (Center column) corresponds to the Wasserstein distance metric, and (Right column) corresponds to the MMD distance metric of different rollout lengths. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

6.3 Dense Reward Cheetah Run Environment

In the previous sections, we detailed the outcomes of the hyperparameter ablation study performed on sparse reward environments. In this section, we introduce the hyperparameter ablation study conducted on the dense reward environment *CheetahRunv0* using the MBPO methodology. The experiments were conducted on rollout length, number of rollouts per step, and updates to retain buffer, the results presented show average returns over 20 random seeds which are smoothened by a moving average filter and report the mean (solid lines) and standard error (shaded regions). The results of the ablation study along with the metrics are discussed in further subsections. We utilize the hyperparameter settings from Table 6.1, which were used to conduct the experiment. This study aimed primarily to investigate the behavior of the key components of model-based reinforcement learning algorithms when faced with scenarios involving dense rewards.

6.3.1 Rollout Length Ablation

In this subsection, an ablation study was conducted on the rollout length hyperparameter. The study was conducted using $k = \{1, 3, 5, 7, 10\}$ and keeping all other hyperparameters fixed as presented in Table 6.1. In Figure 6.12, It can be seen that in this complex and dense reward environment, having a longer rollout length aided the agent in discovering new states in the beginning. Rollout length 1, did not contribute to any performance gain throughout the entire training period which suggests that having a lesser value of rollout length in a complex environment like cheetah run did not contribute to the performance.

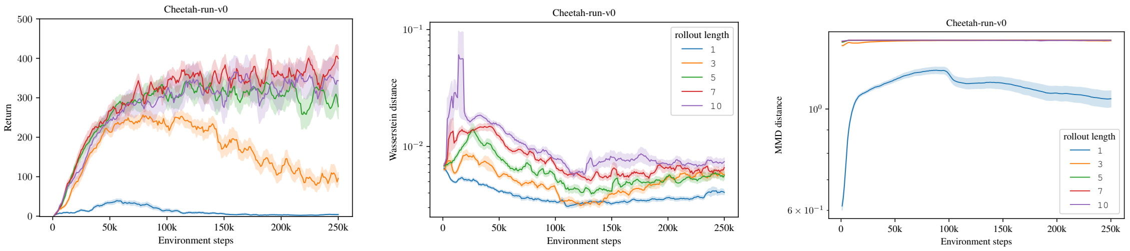


Figure 6.12: Ablation study over rollout length hyperparameter conducted using MBPO agent in $CheetahRunv0$ environment. (Left column) We report the learning curves, (Center column) corresponds to the Wasserstein distance metric, and (Right column) corresponds to the MMD distance metric of different rollout lengths. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

In the evaluation returns plot (Left column), we can deduce the influence of different rollout lengths from the ablation study conducted in the cheetah-run environment. When the rollout length value is set to 1, we observe that the evaluation returns remained at zero for the majority of the training duration, and higher rollout length values resulted in correspondingly higher returns. The trend observed in this ablation study aligns with the trends observed in both sparse reward environments, indicating that the rollout length should be of a reasonable length to enable the model to achieve higher evaluation returns since longer rollout lengths emphasize the increased exploration of unfamiliar states.

In the Wasserstein distance metric plot (Center column), we notice that the gap between model-generated rewards and environment reward distributions initially grows during the initial iterations, and the disparity begins to decrease as the training advances. Interestingly, a rollout length value of 1 exhibits a decreasing trend right from the beginning of the training. This phenomenon can be attributed to the fact that the environment is designed as a dense reward environment, where the model receives a reward for each action it takes, thereby promoting increased exploration with higher rollout lengths. On the other hand, despite the environment being configured for dense rewards, setting the rollout length value to 1 did not facilitate effective exploration for the model.

The MMD distance metric plots (Right column) show an initial upward trend at the commencement of training, and after 100k steps, the disparity between model-observed states and the environment's state distribution begins to diminish. This behavior suggests that, during the initial stages of training, the model is incentivized to engage in more exploration due to the presence of dense rewards. This corresponds to the observed upward trend in the MMD distance metric plot. As the model becomes more acquainted with the dynamics of the environment, the distance between the distributions begins to decrease for the remainder of the training.

6.3.2 Rollouts per Step Ablation

In this subsection, we delve into the outcomes of the ablation study performed on the hyperparameter for the number of rollouts per step. The study encompassed values of $M = \{1, 50, 400\ ,1000\}$, and employed the fixed base hyperparameters outlined in Table 6.1.

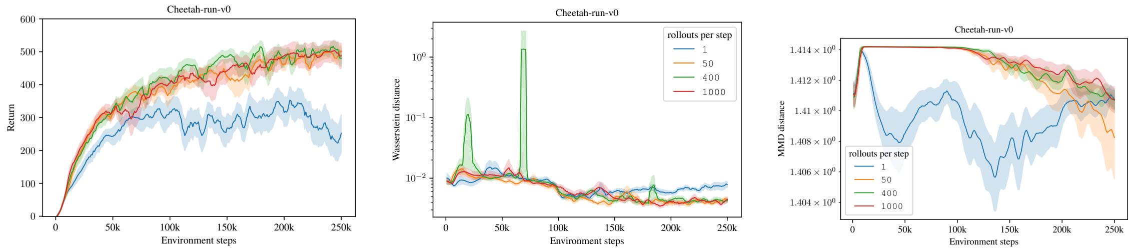


Figure 6.13: Ablation study over the number of rollouts per step hyperparameter conducted using MBPO agent in $CheetahRunv0$ environment. (Left column) We report the learning curves, (Center column) corresponds to the Wasserstein distance metric, and (Right column) corresponds to the MMD distance metric of different rollout lengths. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

In Figure 6.13, we observe in the evaluation return plot (Left column) that having a number of initial states sampled from the environment buffer greater than 1 resulted in higher returns. However, there is no significant distinction indicating a lesser impact of the number of states sampled. Setting the number of rollouts per step value to one, led to lower returns in comparison to the other values examined in the ablation study. The returns remained relatively constant for a brief duration and then began to decline towards the end of the training. Interestingly, setting the rollouts per step value to one yielded a substantial amount of returns. This outcome can be attributed to other concurrent hyperparameters and the fact that the cheetah-run environment features dense rewards.

In the Wasserstein distance metric plot (Center column), it is evident that irrespective of the number of rollouts per step sampled from the environment buffer and the execution of k-step rollouts from M initial states, the disparity between the model-generated rewards and the environment's reward distribution converges to a smaller value as the training progresses.

The MMD distance metric plot shows that regardless of the value of the number of rollouts per step hyperparameter, the disparity between the model-observed states and the environment states increases during the initial training phase, influenced by the dense reward nature of the cheetah-run environment. The model capitalizes on this concept and engages in increased exploration during the early stages. During a short duration of training, the disparity remains steady for higher values of rollouts per step, with the exception of value 1. At approximately 150k environment steps, the difference between the two sets of observed state distributions begins to diminish, signifying that the model has started the process of observing states resembling those seen in the environment. The fluctuating behavior of the rollouts per step value of 1 indicates that it is an insufficient amount of a satisfactory number of initial states sampled from the environment buffer for effective exploration in the complex dense-reward cheetah-run environment.

6.3.3 Updates to Retain Buffer Ablation

In this subsection, the ablation study is extended to the number of updates to retain buffer hyperparameter. The ablation was carried out using values $R = \{1, 3, 7, 10\}$, and the remaining hyperparameters were fixed to the base values presented in Table 6.1.

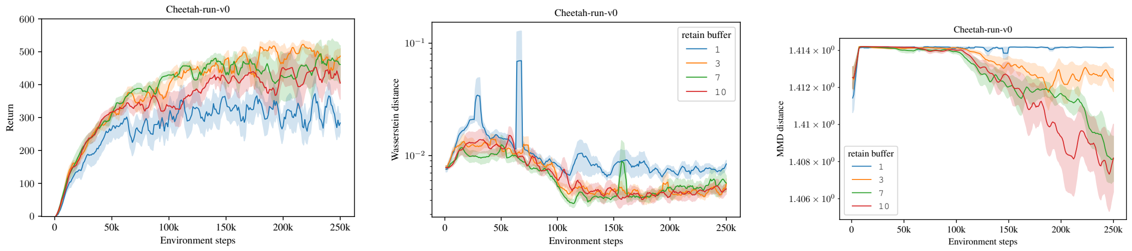


Figure 6.14: Ablation study over the number of updates to retain buffer hyperparameter conducted using MBPO agent in $CheetahRunv0$ environment. (Left column) We report the learning curves, (Center column) corresponds to the Wasserstein distance metric, and (Right column) corresponds to the MMD distance metric of different rollout lengths. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

The evaluation return plot displayed in Figure 6.14 clearly demonstrates that, within a more intricate environment like cheetah-run, enhancing the retention of off-policy experiences significantly improved the model's ability to effectively explore the complex environment, despite the environment's dense reward structure. This observed behavior of the model also parallels the behavior exhibited by the model in the complex mountain car environment, characterized by sparse rewards. This suggests that increasing the number of updates to retain the buffer in complex environments, irrespective of the reward density, aids the model in achieving higher returns.

In the Wasserstein distance metric plot (Center column), we can observe the discrepancy between the reward distributions is higher for the initial parts of the training, and as the model gets starts to capture the true environment dynamics the distance between the distributions starts to reduce to a lower value. Interestingly, the distance remains higher for $R = 1$ in comparison to the other values. The frequent overwriting of the buffer encourages the model to explore increasingly unfamiliar states, given the environment's dense reward nature.

Similarly, in the MMD distance metric plots, we notice that the disparity between model-observed states and the environment state distributions initially increases and remains constant for the first 100k environment steps. Upon further interactions with the environment, we observe that the value $R = 1$ remains constant until the end of the training, while the distance between the distributions begins to decrease for the remaining values of updates to retain the buffer. This indicates that having the model buffer frequently overwritten in a dense reward environment encourages the model to explore more unfamiliar states and eventually the non-convergence behavior of the MMD distance metric.

6.4 Reduced MBPO Performance

In the preceding subsections, we performed comprehensive hyperparameter ablation studies on both sparse and dense reward environments to comprehend the influence of each hyperparameter on the learning performance of the model. Utilizing the IPMs, we elucidated the influence of each hyperparameter on the extent of variation between the reward distribution and the observed state distribution of the model in comparison to the environment. From the results obtained in the hyperparameter ablation study, we cannot deduce conclusions regarding which individual hyperparameter or combinations of hyperparameters of the MBRL algorithm contributed to the returns in the sparse reward scenarios.

To gain a deeper understanding of the primary contributing hyperparameter or a combination of hyperparameters in the MBRL algorithm, we make adjustments to the performance of the MBPO algorithm to align it with the performance of model-free counterpart SAC in sparse reward scenarios. Due to the SAC agent's inadequate performance, the agent did not fare well in sparse reward scenarios involving action cost. By utilizing the existing model-based hyperparameters, it is achievable to approximate the performance of the MBPO agent to that of the model-free SAC. Table 6.2 outlines the base values of the hyperparameters employed and an ensemble size of value $N = 1$ to reduce the performance of the MBPO agent close model-free SAC performance in sparse reward scenarios with action cost. This experiment was carried out with 20 random seeds with an action cost of 0.1 in the case of $SparsePendulumv0$ environment and an inbuilt action cost in the reward function of the mountain car. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds. As part of our further investigation in this study, we design experiments in which each hyperparameter is individually scaled up and analyzed in isolation to ascertain the influence of the hyperparameter on the performance of the MBPO agent in sparse reward scenarios. As illustrated in Figure 6.15, we can observe the influence of adjusting the model-based hyperparameters in both sparse reward environments. The main reason for achieving a performance level for MBPO comparable to that of a model-free SAC agent is that MBPO internally employs model-free SAC as a policy optimization component. Thus, MBPO can be conceptualized as an extended iteration of model-free SAC within a model-based context.

Figure 6.16 illustrates the comparison of the learning curves between the model-based MBPO and MBPO operating in proximity to a model-free SAC agent, accompanied by the Wasserstein and MMD distance metrics. The variation in performance between MBPO and MBPO operating akin to a model-free SAC agent (MBPO SAC) is distinctly evident in both sparse reward environments, as observed from the evaluation returns plots. It is also evident that the evaluation returns are not completely zero in both sparse reward environments. This can be attributed to the hyperparameter setting where the ensemble size is configured to a value of five, enabling the model-based agent to discover certain rewards.

In the Wasserstein distance metric plot (Center column), we observe that, during the initial stages of training, the MBPO agent exhibits a larger disparity between the model-generated rewards and the environment reward distribution compared to the MBPO SAC agent. However, as the training advances, the MBPO SAC agent manages converges to a smaller distance in comparison to the MBPO agent, because the model-based agent is operating closely in alignment with model-free SAC agents, it demonstrates an on-policy behavior. Likewise, we also witness this same behavior in the continuous mountain car environment.

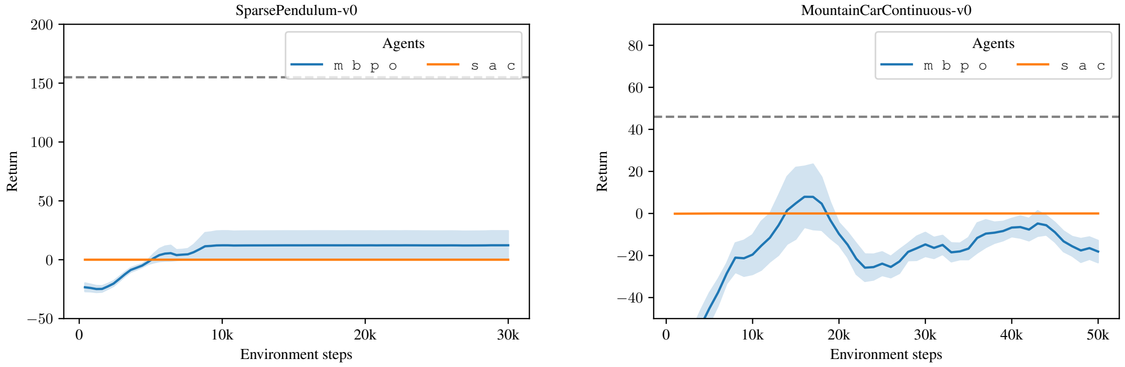


Figure 6.15: Learning curves representing the reduced performance of an MBPO agent operating close to model-free SAC agent in both $SparsePendulumv0$ environment and $MountainCarContinuousV0$ environment using hyperparameter settings from Table 6.2. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

In the MMD distance metric plot (Right column), we observe that the disparity between model-observed states and the environment state distribution is higher for the MBPO agent and comparatively lower for the MBPO SAC agent in both sparse reward environments. The results derived from the MMD distance metric plot indicate that the MBPO SAC agent is exploring states that closely resemble those visited in the environment. This can be attributed to the on-policy behavior exhibited by the agent.

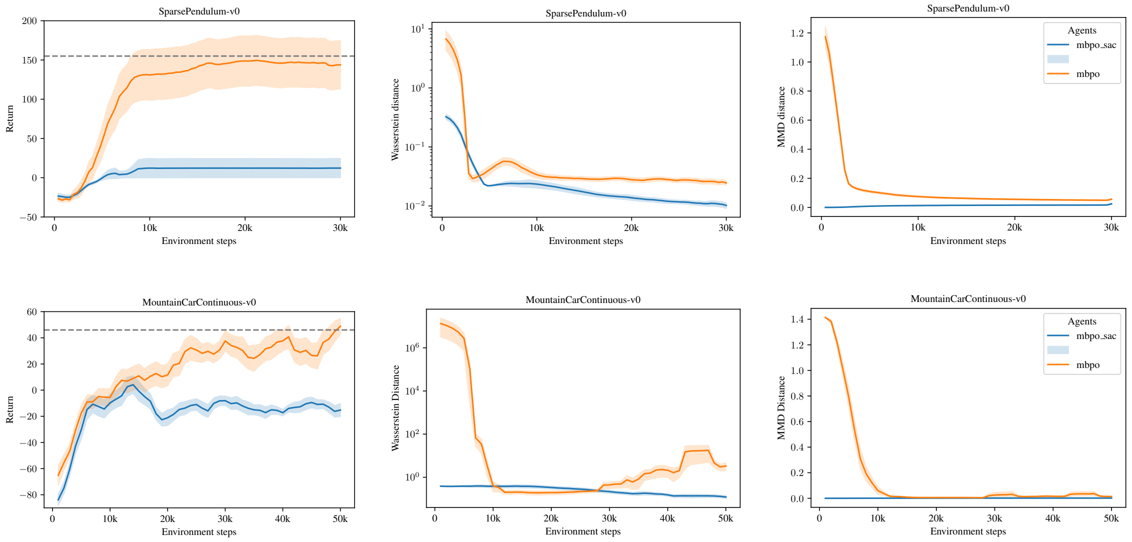


Figure 6.16: (Left column) We report the learning curves, (Center column) corresponds to the Wasserstein distance metric, and (Right column) corresponds to the MMD distance metric. A comparative metric study was conducted between the MBPO agent and MBPO agent operating close to model-free SAC agent using hyperparameters from 6.2 in both $SparsePendulumv0$ and $MountainCarContinuous - V0$ environment. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

6.4.1 Ensemble Size - Rollout Length Ablation

In this subsection, we conduct an ablation study to comprehend the influence of ensemble size and rollout length hyperparameters on the learning performance of the model under sparse reward scenarios, subsequent to diminishing the performance of the MBPO agent to approximate the MBPO SAC setting. The experiments were executed with values of $N = \{1, 3, 7, 10\}$ and $k = \{1, 3, 7, 10\}$, and the remaining hyperparameter settings were maintained as outlined in Table 6.2. The experiments are conducted over 20 random seeds and the results are the average (solid lines) and standard error (shaded regions).

Figure 6.17 depicts the evaluation returns (Top row), Wasserstein distance metric (Middle row), and MMD distance metric (Bottom row). The evaluation returns plot indicates that setting the rollout length of value $k = 1$ resulted in lower performance of the model in the sparse pendulum environment, regardless of varying ensemble sizes. This confirms the hypotheses we had in section 6.2.1 where rollout lengths of value greater than 1 contribute to better evaluation returns. Interestingly, a rollout length of $k = 10$ also did not contribute to better returns when compared to rollout lengths of value $k = 3$ and $k = 7$. A similar kind of trend is also observed in the mountain car environment which is depicted in Figure 6.18. This phenomenon can be attributed to the reliance of model-based methods on learned or approximated environment models. With longer rollouts, these inaccuracies become more prominent, increasing the likelihood of divergence between the model's predictions and the actual behavior of the environment. Consequently, this can result in sub-optimal policies and inadequate convergence. Additionally, errors in initial states or observations can compound over extended rollout lengths, leading to erroneous predictions

Table 6.2: MBPO Hyperparameter settings reduced close to SAC Performance

Hyperparameter	Sparse Pendulum	Mountain Car
# environment steps	30e3	50e3
T - # episodes	75	50
E - # steps per episode	400	1000
G - policy updates per step	20	
M - # model rollouts per step	150	250
F - frequency of model retrain (#steps)	200	
R - # updates to retain buffer	1	
N - # ensemble size	5	
k - rollout length	1	
trainer patience	1	
learning rate	3e-4	
Policy network	2 layers, 64 units, Tanh activation	
Q network	2 layers, 256 units, Tanh activation	
Model network	4 layers, 200 units, SiLU activation	

and potentially causing the agent to deviate significantly from the optimal policy. In the complex mountain car environment, we can also observe that merely increasing the rollout lengths was adequate to achieve returns surpassing the baseline standard established for the mountain car environment, even with an ensemble size set to one. This also validates the hypothesis discussed in section 6.2.1 that a larger value of rollout lengths did indeed assist the model in achieving higher returns in more complex environments like continuous mountain car.

In the Wasserstein distance metric plot (Middle row) in Figure 6.17, we observe that the curve associated with a rollout length of one, with the ensemble size value set to one, exhibited a non-convergent behavior throughout the entire training duration, conversely, in the remaining plots where the ensemble size is greater than one, even a rollout length of one displayed a convergent behavior. This suggests that when the dynamics of the environment are more accurately represented using ensemble sizes greater than one, even a rollout length of one is adequate in the case of the simple sparse pendulum. However, this pattern is not observed in the complex mountain car environment (Middle row) in Figure 6.18, indicating that regardless of the ensemble size, a rollout length of one is insufficient for effective exploration of the model in a complex environment.

In the MMD distance metric plots (Bottom row) in Figure 6.17, the difference between the model-observed states and the environment states increased consistently throughout the entire training period. This indicates that an ensemble size of one is inadequate to capture the environment dynamics effectively, and the model, regardless of the rollout length, continues to explore without

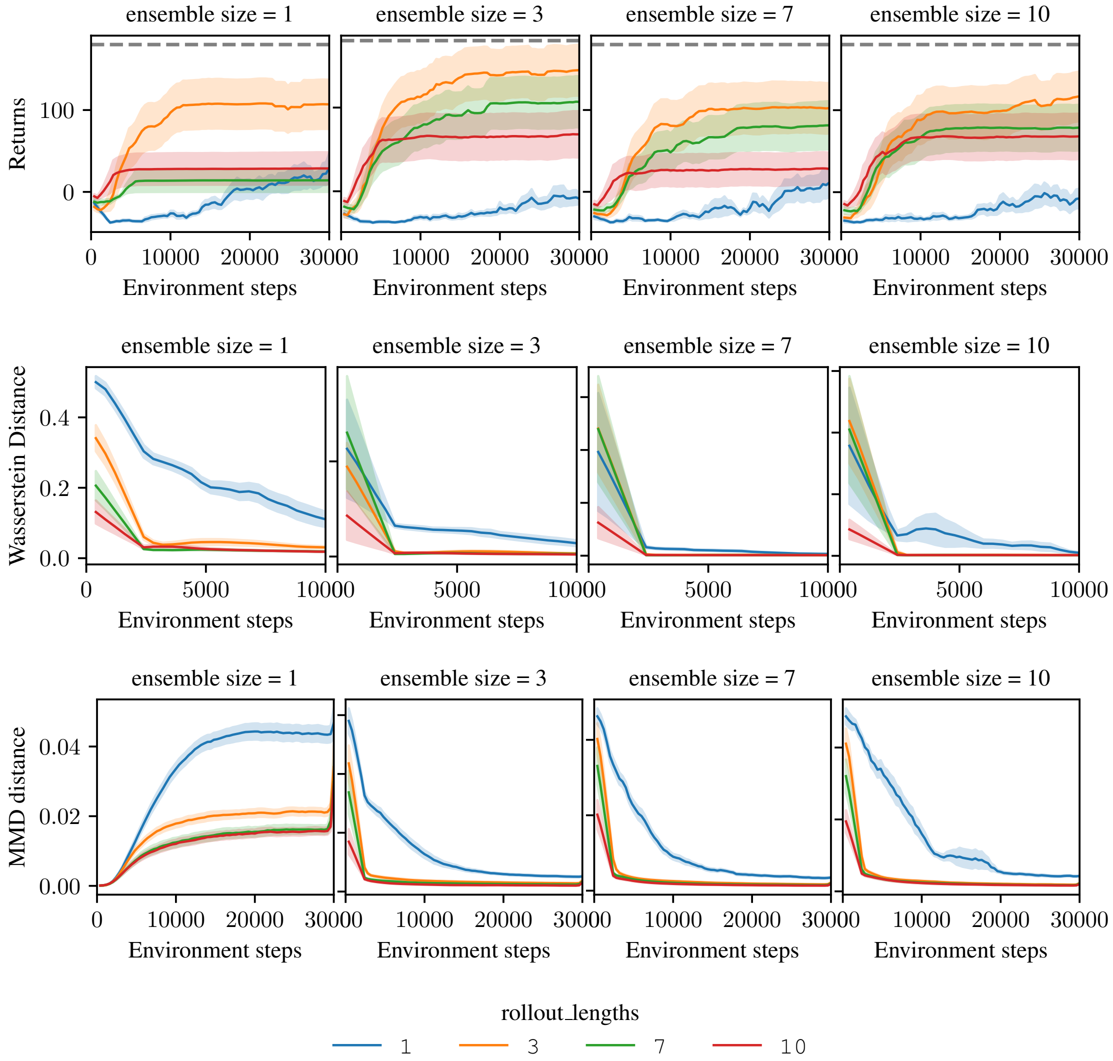


Figure 6.17: (Top row) We report the learning curves, (Middle row) corresponds to the Wasserstein distance metric, and (Bottom row) corresponds to the MMD distance metric. An ablation study over ensemble size and rollout length hyperparameters was conducted on an MBPO agent operating close to a model-free SAC agent using hyperparameters from Table 6.2 in *SparsePendulumv*0 environment. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

demonstrating a convergent behavior. However, this behavior is not observed in the plots where the ensemble size is greater than one. A similar trend is also observed in the continuous mountain car environment (Bottom row) in Figure 6.18.

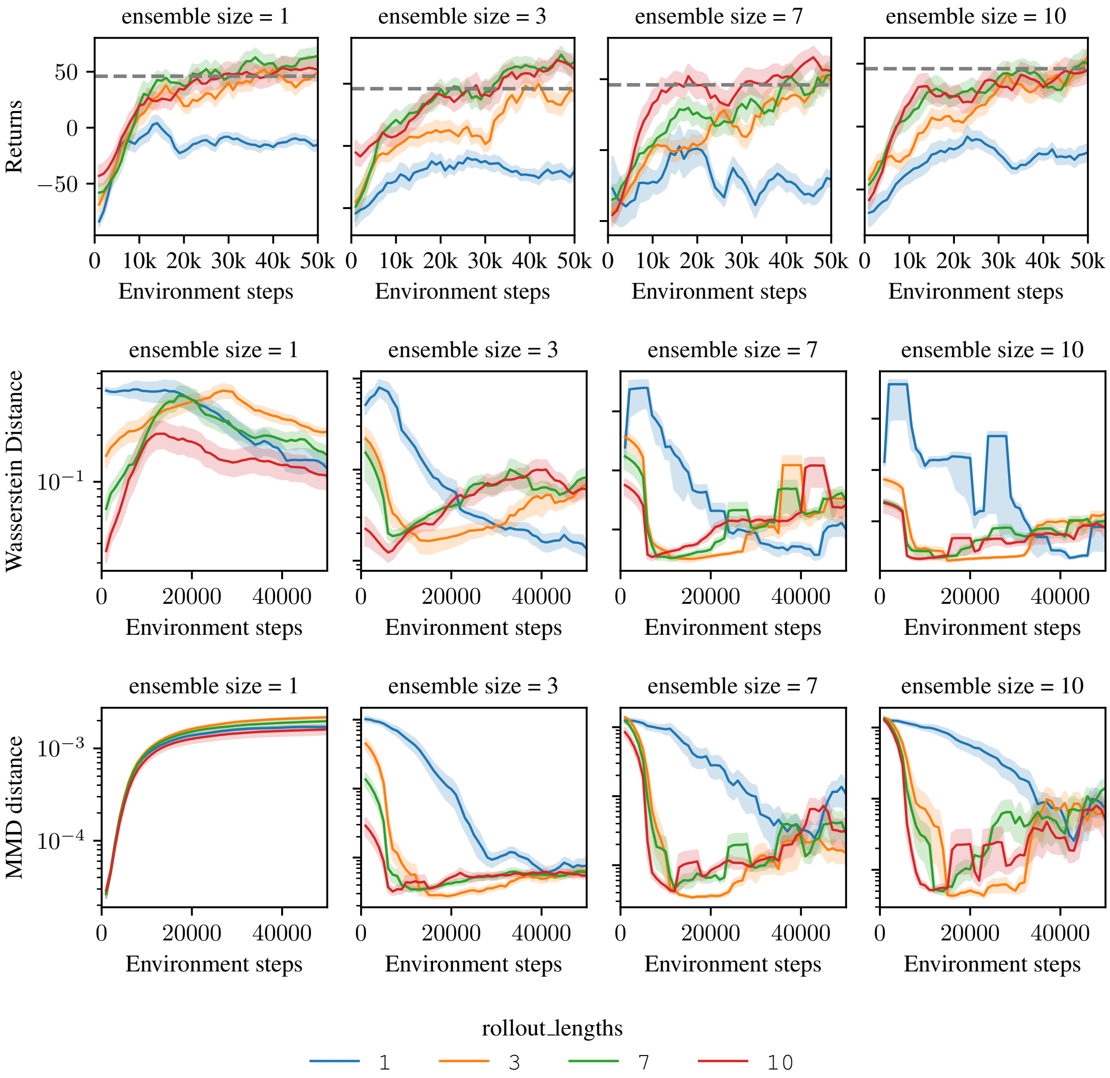


Figure 6.18: (Top row) We report the learning curves, (Middle row) corresponds to the Wasserstein distance metric, and (Bottom row) corresponds to the MMD distance metric. An ablation study over ensemble size and rollout length hyperparameters was conducted on an MBPO agent operating close to a model-free SAC agent using hyperparameters from Table 6.2 in $MountainCarContinuousv0$ environment. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

6.4.2 Rollouts per Step - Frequency Retrain

In this subsection, we present the outcomes of an ablation study conducted on the number of rollouts per step and model retrain frequency hyperparameters. We investigate the influence of these hyperparameters on the learning performance while keeping the remaining hyperparameter settings set with those detailed in Table 6.2. The experiments were executed with values of $M = \{50, 400, 1000, 2000\}$ and $F = \{50, 400, 1000, 2000\}$ and conducted over 20 random seeds and the results are the average (solid lines) and standard error (shaded regions).

Figure 6.19 illustrates the evaluation returns (Top row), Wasserstein distance metric (Middle row), and MMD distance metric (Bottom row) in the sparse pendulum environment. The evaluation return plot (Top row) in Figure 6.19, reveals that the number of rollouts per step value of $M = 50$ led to subpar performance, regardless of the increased number of model retrain frequencies indicating that the number of initial states sampled from the environment buffer is inadequate for the model to enhance its learning performance, regardless of the frequency of model parameter updates. Conversely, setting the number of rollouts per step to $M = 2000$ yielded improved performance compared to having fewer initial states. However, the learning performance of the model with $M = 2000$ still remains lower than that of values $M = 400$ and $M = 1000$ in the sparse pendulum environment. The more amount of data we sample from the environment buffer the noise associated with the data is also increased which in turn hurts the performance in the sparse pendulum environment. A comparable trend is likewise noticed in the mountain car environment (Top row) in Figure 6.20, where setting the number of rollouts per step to $M = 1000$ and $M = 2000$ did not contribute to an improvement in the model's learning performance. We can deduce that $M = 50$ and $M = 400$ were adequately effective in assisting the model to achieve better learning performance with greater sample efficiency.

In the Wasserstein distance metric plot (Middle row) in Figure 6.17, we observe a larger discrepancy between the model-generated and environment reward distributions for the number of rollouts per step values exceeding $M = 50$ during the initial phase of training. However, as the training progressed, these discrepancies displayed a convergent behavior. Despite the subpar performance in evaluation returns with $M = 50$, the model still succeeded in discovering rewards, although the disparity between distributions was lower compared to rollout per step values higher than 50 at the beginning of the training. Surprisingly, this convergent trend is also observed in the mountain car environment (Middle row) in Figure 6.18. However, each metric curve converged to a distinct point at the end of the training, indicating that different frequencies of updating the model parameters resulted in varied convergence points. This consistency with the results obtained in subsection 6.2.5 of the mountain car environment further supports the findings.

In the MMD distance metric plots (Bottom row) in Figure 6.19 and Figure 6.20, we notice a consistent trend in both sets of plots. The discrepancy between model-observed states and the environment state distribution exhibited a slight increase over the course of training. This trend remained consistent regardless of the varying number of rollouts per step and the diverse frequencies of model parameter updates.

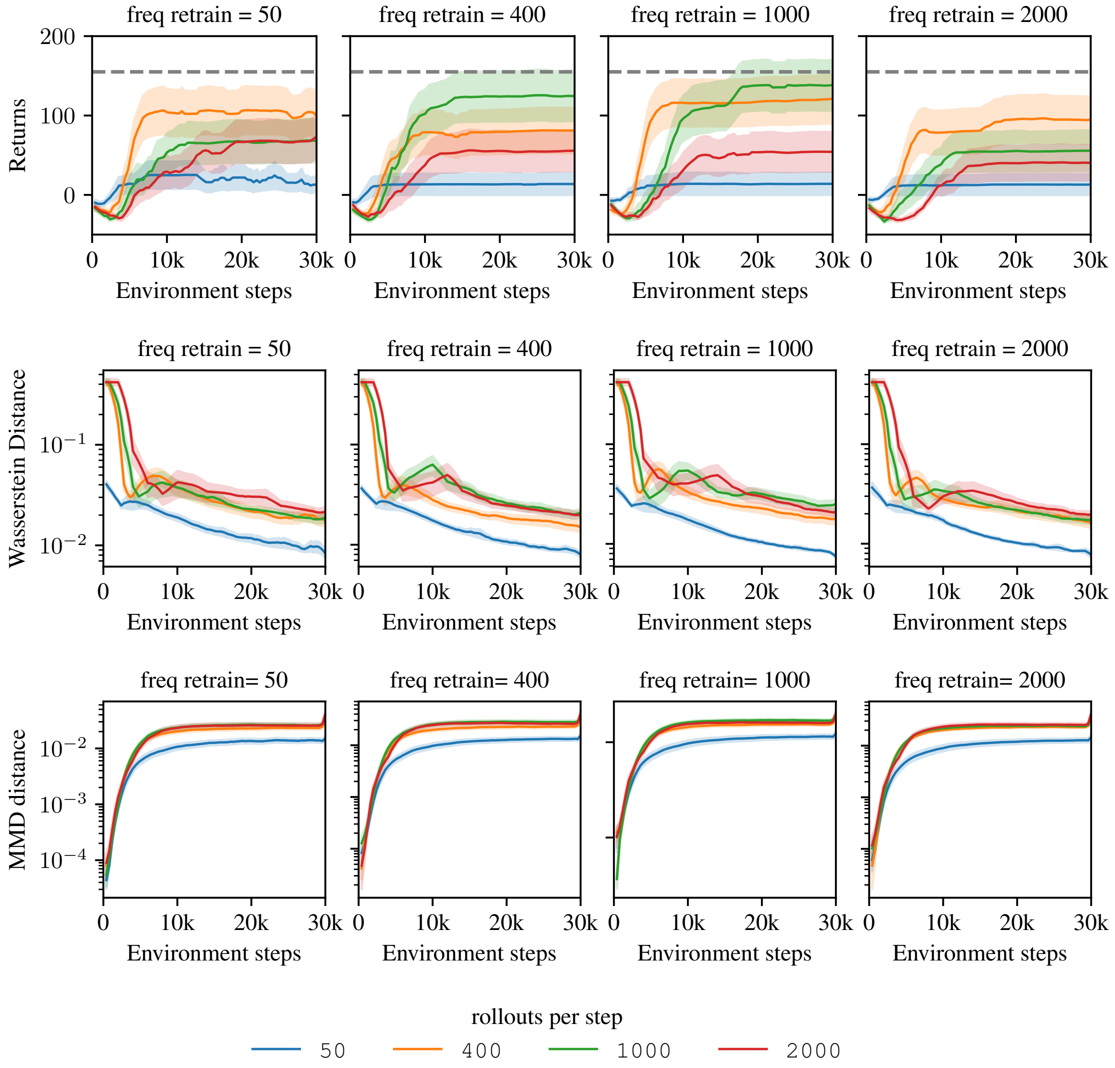


Figure 6.19: (Top row) We report the learning curves, (Middle row) corresponds to the Wasserstein distance metric, and (Bottom row) corresponds to the MMD distance metric. An ablation study over the number of rollouts per step and frequency retrain hyperparameters was conducted on an MBPO agent operating close to a model-free SAC agent using hyperparameters from Table 6.2 in $SparsePendulumv0$ environment. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

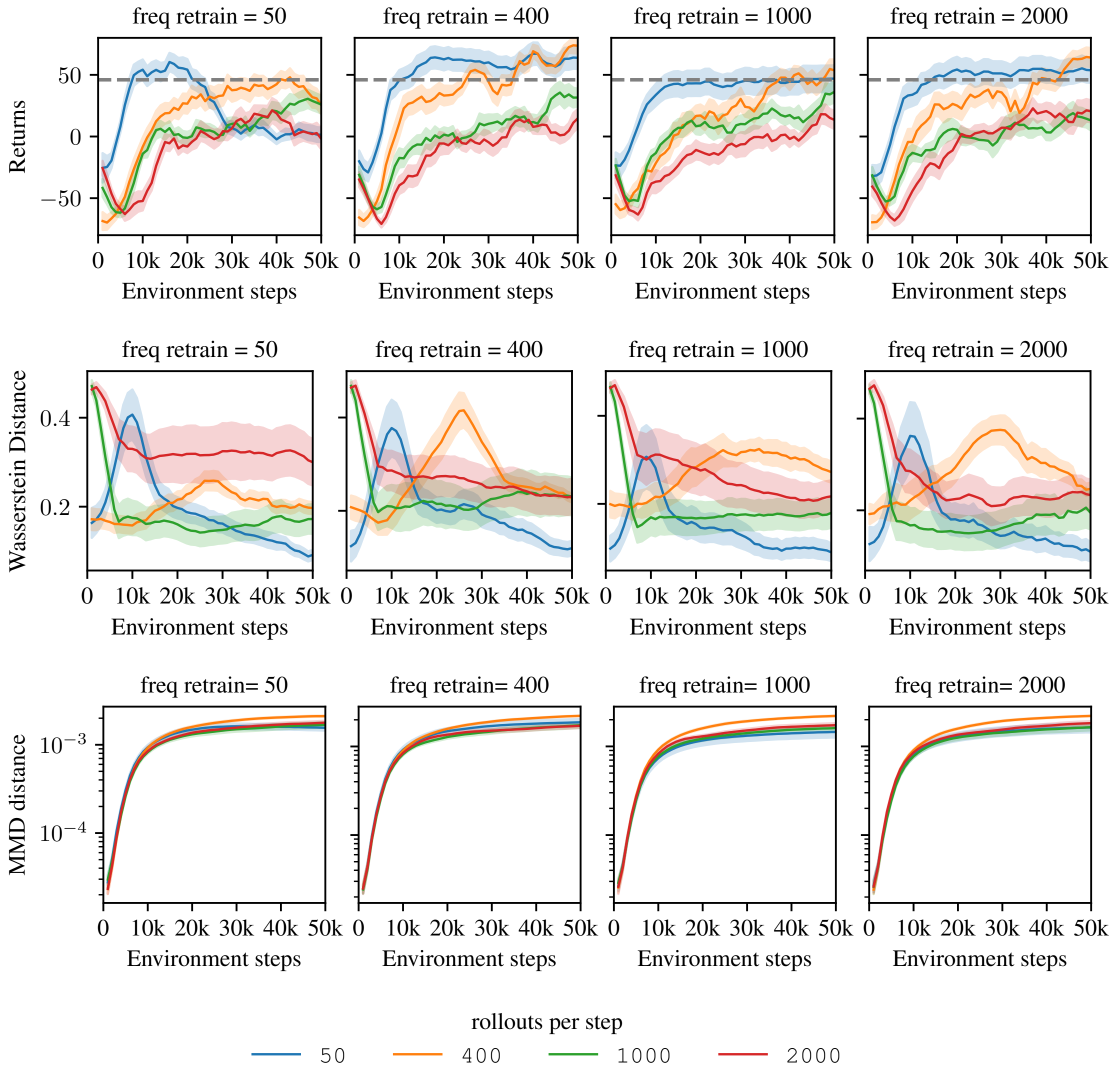


Figure 6.20: (Top row) We report the learning curves, (Middle row) corresponds to the Wasserstein distance metric, and (Bottom row) corresponds to the MMD distance metric. An ablation study over the number of rollouts per step and frequency retrain hyperparameters was conducted on an MBPO agent operating close to a model-free SAC agent using hyperparameters from Table 6.2 in $MountainCarContinuousv0$ environment. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

6.4.3 Rollout Length - Rollouts per Step

In this subsection, we discuss the results obtained from an ablation study conducted on rollout length and the number of rollouts per step hyperparameters. We conduct this ablation study to infer the impact of these hyperparameters on the learning performance of the model while keeping the remaining hyperparameter settings set with those detailed in Table 6.2. The experiments were carried out with hyperparameter values of $k = \{10, 20, 30\}$ and $M = \{50, 400, 1000, 2000\}$ and conducted over 20 random seeds, and the results are the average (solid lines) and standard error (shaded regions).

Figure 6.21 presents the evaluation returns (Top row), Wasserstein distance metric (Middle row), and MMD distance metric (Bottom row) in the sparse pendulum environment. In the evaluation return plot (Top row) of Figure 6.21, it is interesting to note that regardless of longer rollout lengths and the number of rollouts per step, the model's performance remained relatively consistent throughout the training. Across all tested cases, the combination of hyperparameters was unable to reach the established baseline for the sparse pendulum environment. This behavior is also in line with the continuous mountain car environment (Top row) shown in Figure 6.22, where despite longer rollout lengths and an increased number of initial states sampled from the environment buffer, the model displayed a consistent performance across all the tested combinations. Interestingly, all the hyperparameter combinations that were tested managed to achieve the baseline standard set for the mountain car environment. The outcomes of the ablation study indicate that even with an increased number of initial states, elevating the rollout length to higher values did not yield a significant impact on learning performance. This finding supports the assertion that longer rollouts tend to enhance sample efficiency in the short term but ultimately lead to convergence towards suboptimal policies [HTB19].

In the Wasserstein distance metric plot (Middle row) in Figure 6.21 and Figure 6.22, we observe that the difference between the model-generated rewards and the environment reward distributions remains relatively constant across all the tested combinations during the initial stages of training. Furthermore, as the training advances, this consistent behavior is maintained in the convergence pattern. The results indicate that despite sampling M number of initial states from the environment buffer and altering the length of rollouts, the model consistently achieved similar convergence behavior in all combinations tested.

In the MMD distance metric plots (Bottom row) in Figure 6.21 and Figure 6.22, a consistent pattern is evident in both sets of plots. The disparity between the model-observed states and the environment state distribution follows a comparable trend as observed in the Wasserstein distance metric plot, indicating that there is no significant alteration in the model-observed state distribution. Surprisingly, the convergence toward a state distribution that closely aligns with the environment distribution is less sample efficient with longer rollouts, particularly in the complex mountain car environment.

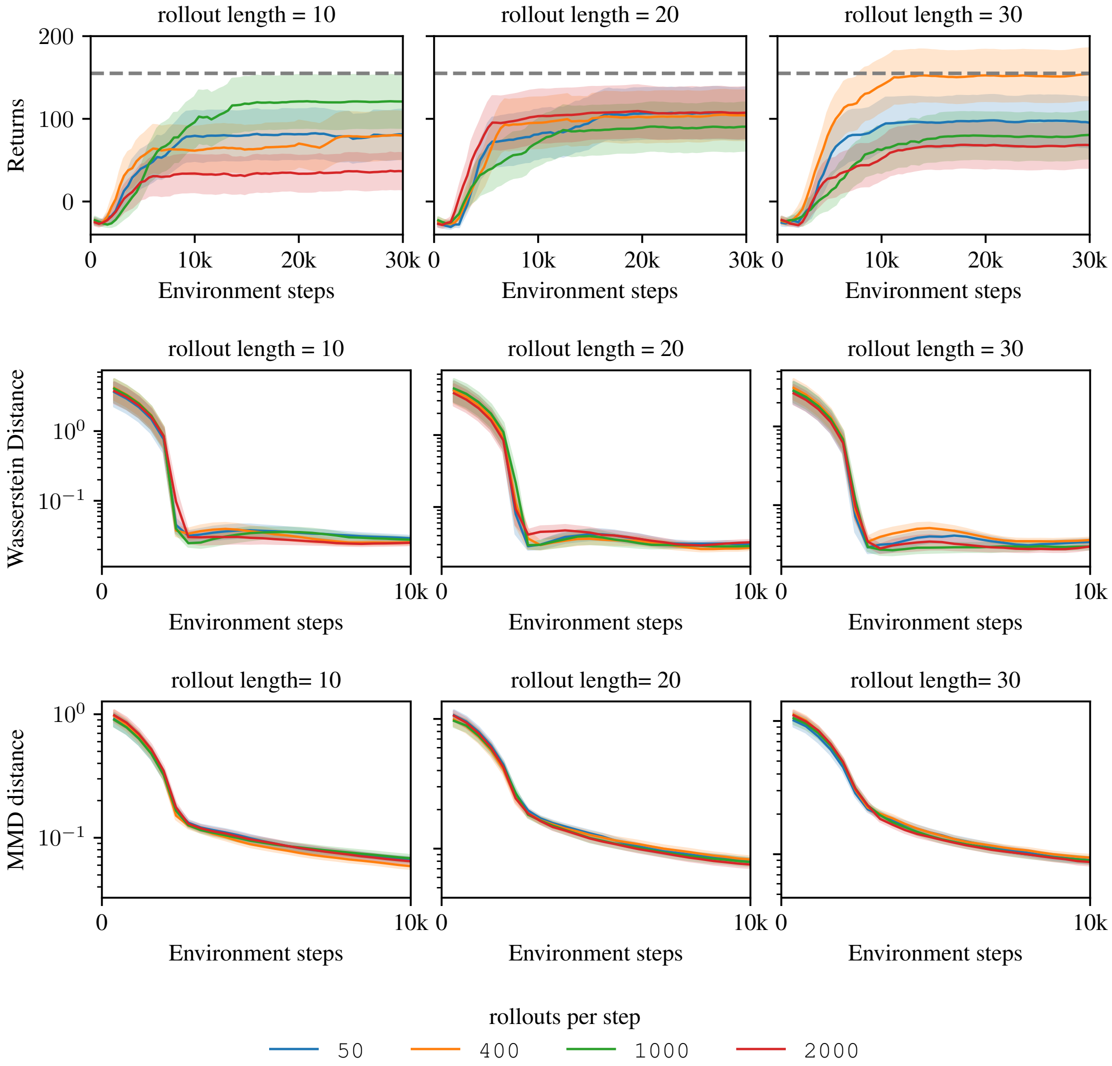


Figure 6.21: (Top row) We report the learning curves, (Middle row) corresponds to the Wasserstein distance metric, and (Bottom row) corresponds to the MMD distance metric. An ablation study over rollout length and the number of rollouts per step hyperparameters was conducted on an MBPO agent operating close to a model-free SAC agent using hyperparameters from Table 6.2 in $SparsePendulumv0$ environment. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

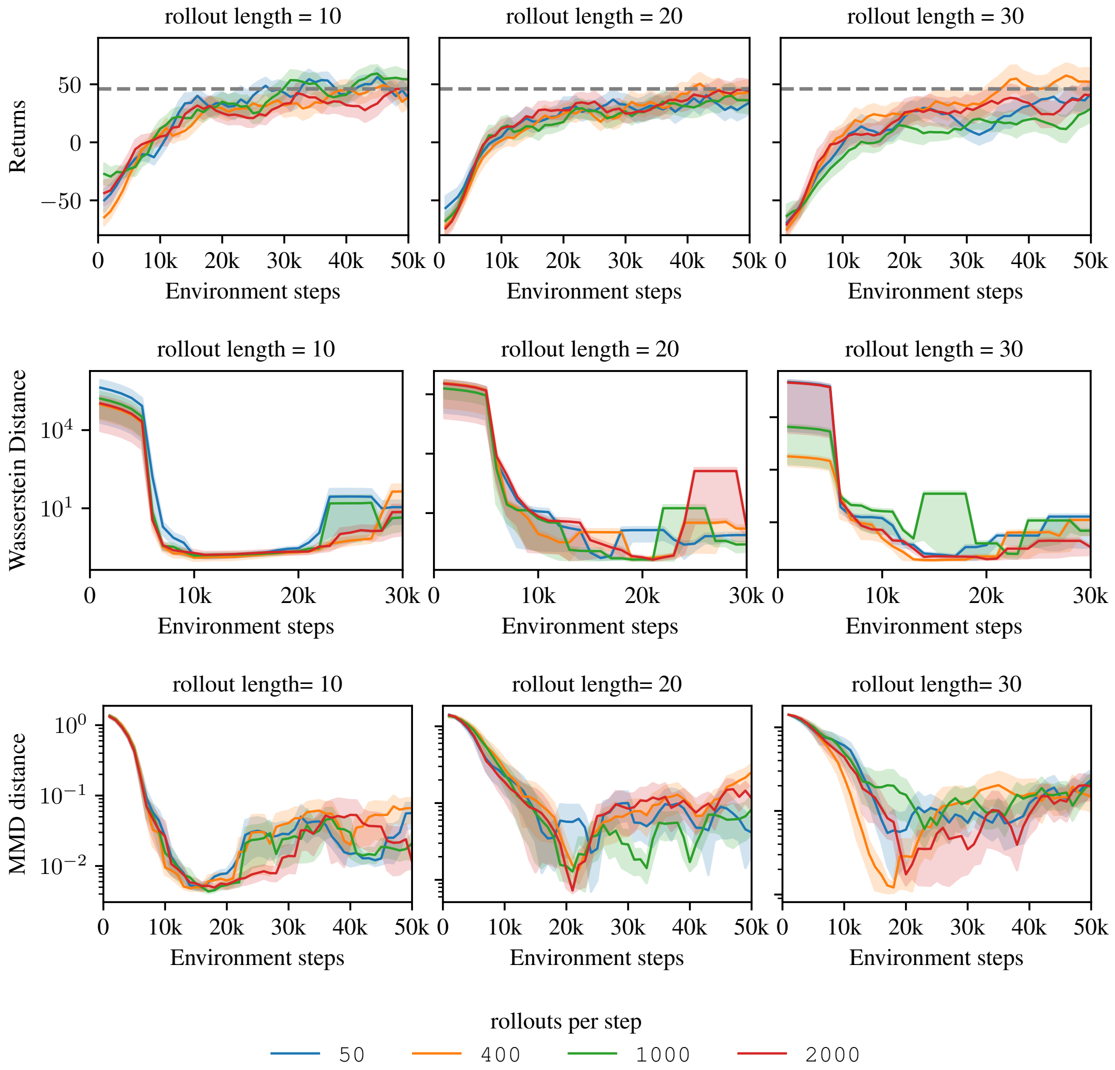


Figure 6.22: (Top row) We report the learning curves, (Middle row) corresponds to the Wasserstein distance metric, and (Bottom row) corresponds to the MMD distance metric. An ablation study over rollout length and the number of rollouts per step hyperparameters was conducted on an MBPO agent operating close to a model-free SAC agent using hyperparameters from Table 6.2 in $MountainCarContinuousv0$ environment. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

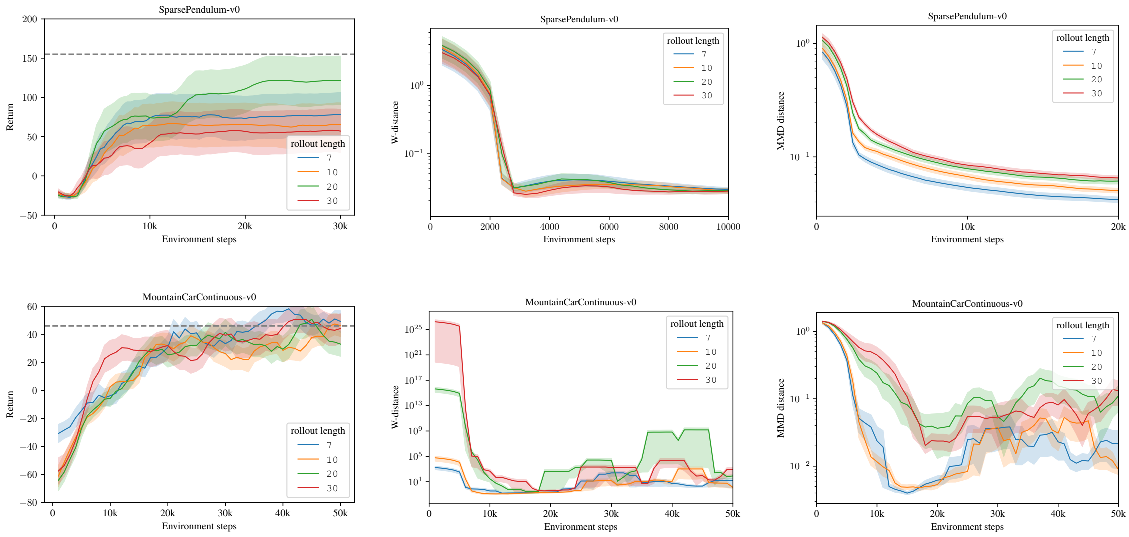


Figure 6.23: Ablation study over longer rollout lengths using MBPO agent operating close to a model-free SAC agent in both $SparsePendulumv0$ and $MountainCarContinuousv0$ environment using hyperparameter settings from Table 6.2. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

6.4.4 Final Ablation study

In the previous subsections, we extensively conducted an ablation study on various hyperparameters to comprehend their impact on learning performance in both sparse reward environments. Through the experiments conducted thus far, after reducing the MBPO agent to a model-free setting, we have identified a set of key hyperparameters that contribute to learning performance in sparse reward scenarios. In this subsection, we undertake a final ablation study to confirm the influence of the selected model-based hyperparameters in sparse reward scenarios in both environments.

Figure 6.23 depicts the influence of longer rollout lengths in both sparse reward environments. The experiments were carried out with hyperparameter values of $k = \{7, 10, 20, 30\}$ and conducted over 20 random seeds, and the results are the average (solid lines) and standard error (shaded regions). The remaining hyperparameter values were set accordingly to the values mentioned in Table 6.1. The sparse pendulum environment (Top row) demonstrates that a rollout length of $k = 20$ exhibited superior learning efficiency in comparison to the other values tested in the ablation study. Likewise, in the complex mountain car environment (Bottom row), the impact of rollout lengths of varying values resulted in relatively similar learning performance, all of which managed to surpass the baseline performance set for the mountain car environment. These results validate the impact of rollout length, as deduced from the previous experiments conducted in the earlier subsections. The results from the Wasserstein distance and MMD distance metric plots also align with the findings from the previous subsections.

Figure 6.24 presents the ablation study conducted on the number of updates to retain the buffer, while maintaining the rest of the hyperparameter settings as detailed in Table 6.2 on both sets of sparse reward environments. The evaluation return plot (Left column) clearly shows that retaining

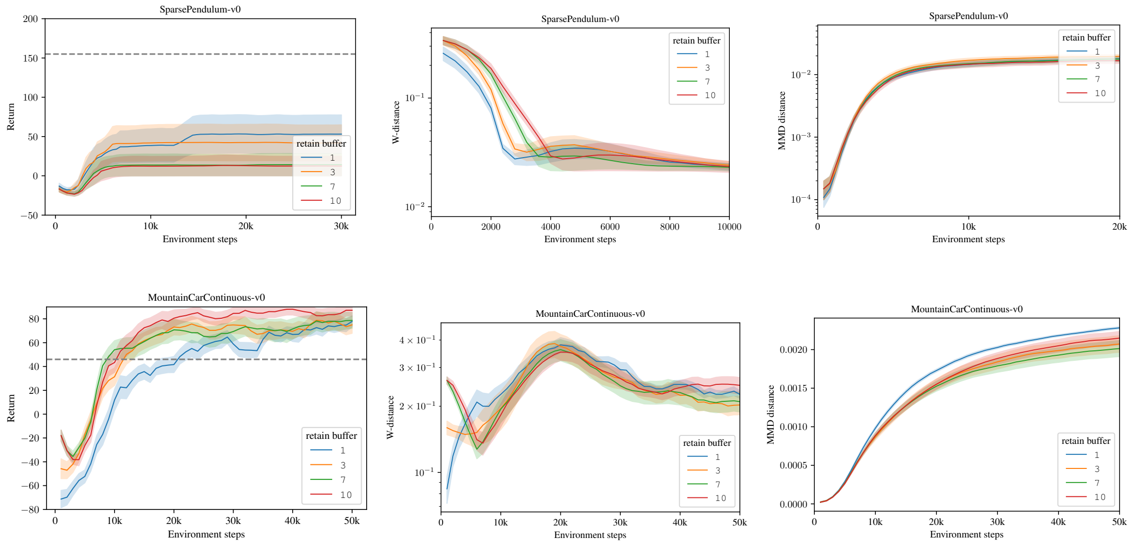


Figure 6.24: Ablation study over updates to retain buffer using MBPO agent operating close to a model-free SAC agent in $SparsePendulumv0$ - and $MountainCarContinuousv0$ environment using hyperparameter settings from Table 6.2. The results are the average (solid lines) and standard error (shaded regions) over 20 random seeds.

a larger amount of off-policy data within the model buffer had a more significant impact on the learning performance in the mountain car environment. In the sparse pendulum environment, the amount of off-policy data retained inside the model buffer did not contribute to the improvement of the learning performance, regardless of its quantity. This observation suggests that the model tends to perform better with on-policy updates in the sparse pendulum environment. This preference is evident in the evaluation return plot, where a number of updates to retain buffer with a value of $R = 1$ yielded better returns compared to other values. The results for the distance metric plot also align with the results from previous subsections. This confirmation supports the hypothesis that retaining off-policy data significantly enhances performance during the model learning phase in a complex mountain car environment and vice-versa effect on the sparse pendulum environment.

7 Discussion and Conclusion

In the following, we discuss our previously presented hyperparameter ablation results and give possible explanations. We gather more insights into model learning using the IPMs which were used in this work. Lastly, we state the insights gained throughout this thesis.

7.1 Results

In this work, we conducted an extensive study on the key components of the model-based reinforcement learning algorithm in scenarios with sparse rewards. We performed the study using two continuous control environments with sparse rewards: "$SparsePendulumV0$änd "$MountainCarContinuousV0$". To conduct further testing, we selected the model-based algorithm MBPO (Model-Based Policy Optimization) as our preferred approach. Using this algorithm, we executed an ablation study on various hyperparameters, including rollout length, number of rollouts per step, number of updates to retain in the buffer, frequency of retraining, and trainer patience. The goal of these investigations is to comprehend the impact of each hyperparameter on the learning performance of the model.

The initial set of ablation studies indicated that rollout length plays a crucial role in performance within both sparse and dense reward environments. However, the ablation study also uncovered that utilizing a rollout length value of $k = 1$ did not contribute to enhanced learning performance. Moreover, the observed states and reward distribution of the model never resembled the state and reward distribution of the environment using a rollout length value of $k = 1$.

Upon conducting additional testing, the ablation studies revealed that the number of updates to retain buffer also played a pivotal role in enhancing learning performance. This hyperparameter exhibited environment-specific behavior, showing that retaining a greater amount of off-policy data significantly improved learning performance within the intricate mountain car environment and cheetah-run environment. However, it had an adverse effect on learning performance within a straightforward sparse pendulum environment, where increased retention of off-policy data correlated with decreased performance.

Furthermore, we conducted an ablation study on the ensemble size hyperparameter, revealing that an ensemble size of $N = 1$ did not yield any positive returns in either of the sparse reward environments. When the ensemble size value exceeded one, the probabilistic models effectively captured the environment dynamics, thereby enhancing learning performance. This trend remains consistent across both sets of sparse reward environments.

Finally, we conducted tests to evaluate the impact of the number of rollouts per step and the frequency retrain hyperparameters on learning performance. The results we obtained from the ablation study indicated that both sets of hyperparameters had only a limited influence on the learning performance for higher values. However, sampling a greater number of initial states from the environment buffer resulted in subpar performance, as the additional states introduced accompanying noise. Increasing the frequency of model parameter updates proved superior to less frequent updates. This trend in the behavior of these hyperparameters was evident in both sparse reward environments.

Additionally, we carried out model-consistent rollouts (MBPO-C), where rollouts were executed under a specific model without mixing them. The objective was to achieve a value distribution with a mean that matches the true value distribution. We conducted the hyperparameter ablation study in parallel with this modified approach, and the results from the ablation study indicated that performing rollouts in a model-consistent manner did not significantly influence the learning performance of the model. Instead, it resulted in subpar performance compared to the standard MBPO method indicating that randomizing the probabilistic models for every rollout step had a significant impact on the learning performance.

7.2 Outlook

Our empirical analysis can be further expanded by the following suggestions. Currently, we fix the rollout lengths to specific values and conduct experiments accordingly. An alternative approach could involve designing an experiment where this hyperparameter dynamically changes during training, initiating with a lower value and adapting as training progresses. This approach would provide insights into the effects of adaptive rollout strategies in sparse reward scenarios. Another avenue for gaining deeper insight into the impact of each hyperparameter is to perform the ablation study across various benchmark environments encompassing multiple sparse reward scenarios. This would enable us to determine whether similar trends persist across diverse environments or if hyperparameter effects are specific to individual environments. Additionally, another potential area for exploration involves testing the impact of off-modelness within sparse reward scenarios. This can be achieved by uniformly sampling a model from the ensemble for each rollout step and then investigating how retaining the previous model parameters influences the data collected by the model. Such an exploration would shed light on the role of off-modelness in sparse reward learning.

7.3 Conclusion

In this study, we examined the influence of different model-based hyperparameters on model learning and observed which combination of these hyperparameters had the most substantial effect on the policy optimization process. We utilized two continuous control sparse reward environments for the investigation and assessed the outcomes using the state-of-the-art MBPO algorithm. By employing the defined IPM metrics, we acquired insight into how each hyperparameter impacted the model's observed state and reward distribution in relation to the environment distribution.

We discovered that longer rollout lengths made a substantial contribution to the evaluation returns in both sparse reward environments. We also observed that an ensemble size of one failed to effectively capture the environment dynamics in both sets of sparse reward environments. It was essential to opt for an ensemble size greater than one to more accurately capture the environment dynamics. Maintaining a moderate number of rollouts per step, combined with more frequent updates to the model parameters, proved sufficient to notably enhance evaluation returns, as this pattern was evident in both sparse reward environments. Regarding the retention of off-policy data, a higher level of retained off-policy data in the mountain car environment led to a significant increase in returns, whereas retaining off-policy data in the sparse pendulum environment was not necessary. Further evaluations across various sparse reward environments remain a topic for future research.

Bibliography

[AWR+17] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, W. Zaremba. “Hindsight Experience Replay”. In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/hash/453fadbd8a1a3af50a9df4df899537b5-Abstract.html (cit. on p. 17).

[BCP+16a] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba. *OpenAI Gym*. 2016. eprint: arXiv:1606.01540 (cit. on pp. 14, 39).

[BCP+16b] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba. *OpenAI Gym*. June 5, 2016. DOI: 10.48550/arXiv.1606.01540. arXiv: 1606.01540[cs]. URL: http://arxiv.org/abs/1606.01540 (visited on 06/29/2023) (cit. on p. 41).

[BESK18] Y. Burda, H. Edwards, A. Storkey, O. Klimov. *Exploration by Random Network Distillation*. arXiv:1810.12894 [cs, stat]. Oct. 2018. DOI: 10.48550/arXiv.1810.12894. URL: http://arxiv.org/abs/1810.12894 (cit. on p. 17).

[BHT+19] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, H. Lee. *Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion*. arXiv:1807.01675 [cs, stat]. June 2019. URL: http://arxiv.org/abs/1807.01675 (cit. on pp. 13, 27).

[BTZ22] A. Bhatia, P. S. Thomas, S. Zilberstein. *Adaptive Rollout Length for Model-Based RL Using Model-Free Deep RL*. June 7, 2022. DOI: 10.48550/arXiv.2206.02380. arXiv: 2206.02380[cs]. URL: http://arxiv.org/abs/2206.02380 (cit. on p. 46).

[CBK20] S. Curi, F. Berkenkamp, A. Krause. *Efficient Model-Based Reinforcement Learning through Optimistic Policy Search and Planning*. arXiv:2006.08684 [cs, eess, stat]. Dec. 2020. DOI: 10.48550/arXiv.2006.08684. URL: http://arxiv.org/abs/2006.08684 (cit. on pp. 14, 18, 40).

[CCML18] K. Chua, R. Calandra, R. McAllister, S. Levine. *Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models*. arXiv:1805.12114 [cs, stat]. Nov. 2018. URL: http://arxiv.org/abs/1805.12114 (cit. on pp. 13, 27, 31).

[CER20] A. Charpentier, R. Elie, C. Remlinger. *Reinforcement Learning in Economics and Finance*. Mar. 22, 2020. DOI: 10.48550/arXiv.2003.10014. arXiv: 2003.10014[cs, econ, q-fin]. URL: http://arxiv.org/abs/2003.10014 (cit. on p. 20).

[CM20] H. Charlesworth, G. Montana. *PlanGAN: Model-based Planning With Sparse Rewards and Multiple Goals*. arXiv:2006.00900 [cs, stat]. June 2020. DOI: 10.48550/arXiv.2006.00900. URL: http://arxiv.org/abs/2006.00900 (cit. on p. 15).

[CVLH19] K. Ciosek, Q. Vuong, R. Loftin, K. Hofmann. "Better Exploration with Optimistic Actor Critic". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: https://papers.nips.cc/paper_files/paper/2019/hash/a34bacf839b923770b2c360eefa26748-Abstract.html (cit. on p. 18).

[DDHB22] M. Dawood, N. Dengler, J. de Heuvel, M. Bennewitz. *Handling Sparse Rewards in Reinforcement Learning Using Model Predictive Control*. arXiv:2210.01525 [cs]. Oct. 2022. DOI: 10.48550/arXiv.2210.01525. URL: http://arxiv.org/abs/2210.01525 (cit. on p. 16).

[DMH19] G. Dulac-Arnold, D. Mankowitz, T. Hester. *Challenges of Real-World Reinforcement Learning*. arXiv:1904.12901 [cs, stat]. Apr. 2019. URL: http://arxiv.org/abs/1904.12901 (cit. on p. 12).

[DR11] M. P. Deisenroth, C. E. Rasmussen. "PILCO: a model-based and data-efficient approach to policy search". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Madison, WI, USA: Omnipress, June 28, 2011, pp. 465–472. ISBN: 978-1-4503-0619-5 (cit. on p. 13).

[EHPM22] O. Eberhard, J. Hollenstein, C. Pinneri, G. Martius. "Pink Noise Is All You Need: Colored Noise Exploration in Deep Reinforcement Learning". In: The Eleventh International Conference on Learning Representations. Sept. 29, 2022. URL: https://openreview.net/forum?id=hQ9V5QN27eS (cit. on p. 44).

[EUD17] S. Elfwing, E. Uchibe, K. Doya. "Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning. ArXiv e-prints (2017)". In: *arXiv preprint arXiv:1702.03118* 1702 (2017) (cit. on p. 28).

[FHM18] S. Fujimoto, H. van Hoof, D. Meger. *Addressing Function Approximation Error in Actor-Critic Methods*. Oct. 22, 2018. DOI: 10.48550/arXiv.1802.09477. arXiv: 1802.09477[cs, stat]. URL: http://arxiv.org/abs/1802.09477 (visited on 06/02/2023) (cit. on p. 27).

[FLZB22] L. P. Fröhlich, M. Lefarov, M. N. Zeilinger, F. Berkenkamp. *On-Policy Model Errors in Reinforcement Learning*. Mar. 3, 2022. DOI: 10.48550/arXiv.2110.07985. arXiv: 2110.07985[cs,eess]. URL: http://arxiv.org/abs/2110.07985 (cit. on p. 18).

[FM21] Y. Fan, Y. Ming. "Model-based Reinforcement Learning for Continuous Control with Posterior Sampling". In: *Proceedings of the 38th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, July 1, 2021, pp. 3078–3087. URL: https://proceedings.mlr.press/v139/fan21b.html (cit. on p. 18).

[FVM+22] A. Filos, E. Vértes, Z. Marinho, G. Farquhar, D. Borsa, A. Friesen, F. Behbahani, T. Schaul, A. Barreto, S. Osindero. *Model-Value Inconsistency as a Signal for Epistemic Uncertainty*. arXiv:2112.04153 [cs]. June 2022. URL: http://arxiv.org/abs/2112.04153 (cit. on p. 13).

[FWS+18] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, S. Levine. *Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning*. Feb. 28, 2018. DOI: 10.48550/arXiv.1803.00101. arXiv: 1803.00101[cs, stat]. URL: http://arxiv.org/abs/1803.00101 (visited on 06/01/2023) (cit. on p. 27).

[GL19] S. Z. Gou, Y. Liu. *DQN with model-based exploration: efficient learning on environments with sparse rewards*. arXiv:1903.09295 [cs, stat]. Mar. 2019. DOI: 10.48550/arXiv.1903.09295. URL: http://arxiv.org/abs/1903.09295 (cit. on p. 15).

[Gre13] A. Gretton. "Introduction to rkhs, and some simple kernel algorithms". In: *Adv. Top. Mach. Learn. Lecture Conducted from University College London* 16 (2013), pp. 5–3 (cit. on p. 35).

[HC20] Y. Huang, Y. Chen. *Autonomous Driving with Deep Learning: A Survey of State-of-Art Technologies*. July 4, 2020. DOI: 10.48550/arXiv.2006.06091. arXiv: 2006.06091[cs]. URL: http://arxiv.org/abs/2006.06091 (cit. on p. 21).

[HCD+16] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. D. Turck, P. Abbeel. "Curiosity-driven Exploration in Deep Reinforcement Learning via Bayesian Neural Networks". en. In: (Aug. 2016). URL: https://openreview.net/forum?id=S1YfFqlq (cit. on p. 16).

[HTB19] G. Z. Holland, E. J. Talvitie, M. Bowling. *The Effect of Planning Shape on Dyna-style Planning in High-dimensional State Spaces*. Mar. 28, 2019. DOI: 10.48550/arXiv.1806.01825. arXiv: 1806.01825[cs]. URL: http://arxiv.org/abs/1806.01825 (cit. on p. 71).

[HZAL18] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine. *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. Aug. 8, 2018. DOI: 10.48550/arXiv.1801.01290. arXiv: 1801.01290[cs, stat]. URL: http://arxiv.org/abs/1801.01290 (cit. on pp. 26, 43).

[JFZL21] M. Janner, J. Fu, M. Zhang, S. Levine. *When to Trust Your Model: Model-Based Policy Optimization*. arXiv:1906.08253 [cs, stat]. Nov. 2021. URL: http://arxiv.org/abs/1906.08253 (cit. on pp. 13, 18, 28, 31, 43).

[KBP13] J. Kober, J. A. Bagnell, J. Peters. "Reinforcement learning in robotics: A survey". In: *The International Journal of Robotics Research* 32.11 (Sept. 1, 2013). Publisher: SAGE Publications Ltd STM, pp. 1238–1274. ISSN: 0278-3649. DOI: 10.1177/0278364913495721. URL: https://doi.org/10.1177/0278364913495721 (cit. on p. 20).

[KCM20] R. Kaushik, K. Chatzilygeroudis, J.-B. Mouret. *Multi-objective Model-based Policy Search for Data-efficient Learning with Sparse Rewards*. arXiv:1806.09351 [cs, stat]. Mar. 2020. DOI: 10.48550/arXiv.1806.09351. URL: http://arxiv.org/abs/1806.09351 (cit. on p. 15).

[LK13] S. Levine, V. Koltun. "Guided Policy Search". In: *Proceedings of the 30th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 1938-7228. PMLR, May 26, 2013, pp. 1–9. URL: https://proceedings.mlr.press/v28/levine13.html (visited on 04/09/2023) (cit. on p. 18).

[LLL+20] B. Li, T. Lu, J. Li, N. Lu, Y. Cai, S. Wang. "ACDER: Augmented Curiosity-Driven Experience Replay". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 2577-087X. May 2020, pp. 4218–4224. DOI: 10.1109/ICRA40945.2020.9197421 (cit. on p. 16).

[LTA20] X. Lu, S. Tiomkin, P. Abbeel. *Predictive Coding for Boosting Deep Reinforcement Learning with Sparse Rewards*. arXiv:1912.13414 [cs, stat]. Aug. 2020. DOI: 10.48550/arXiv.1912.13414. URL: http://arxiv.org/abs/1912.13414 (cit. on p. 15).

[LWD+22] Y. Luo, Y. Wang, K. Dong, Q. Zhang, E. Cheng, Z. Sun, B. Song. *Relay Hindsight Experience Replay: Self-Guided Continual Reinforcement Learning for Sequential Object Manipulation Tasks with Sparse Rewards*. arXiv:2208.00843 [cs]. Nov. 2022. DOI: 10.48550/arXiv.2208.00843. URL: http://arxiv.org/abs/2208.00843 (cit. on p. 17).

[LWRG22] H. Lei, P. Weng, J. Rojas, Y. Guan. “Planning with Q-Values in Sparse Reward Reinforcement Learning”. en. In: *Intelligent Robotics and Applications*. Ed. by H. Liu, Z. Yin, L. Liu, L. Jiang, G. Gu, X. Wu, W. Ren. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2022, pp. 603–614. ISBN: 978-3-031-13844-7. DOI: 10.1007/978-3-031-13844-7_56 (cit. on p. 16).

[LWZZ21] S. Li, X. Wang, W. Zhang, X. Zhang. “A Model-Based Approach to Solve the Sparse Reward Problem”. In: *2021 4th International Conference on Pattern Recognition and Artificial Intelligence (PRAI)*. Aug. 2021, pp. 476–480. DOI: 10.1109/PRAI53619.2021.9551093 (cit. on p. 15).

[LXL+21] Y. Luo, H. Xu, Y. Li, Y. Tian, T. Darrell, T. Ma. *Algorithmic Framework for Model-based Deep Reinforcement Learning with Theoretical Guarantees*. arXiv:1807.03858 [cs, stat]. Feb. 2021. DOI: 10.48550/arXiv.1807.03858. URL: http://arxiv.org/abs/1807.03858 (cit. on p. 28).

[MKS+13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller. *Playing Atari with Deep Reinforcement Learning*. Dec. 19, 2013. DOI: 10.48550/arXiv.1312.5602. arXiv: 1312.5602[cs]. URL: http://arxiv.org/abs/1312.5602 (cit. on p. 21).

[MR21] R. McCarthy, S. J. Redmond. *Imaginary Hindsight Experience Replay: Curious Model-based Learning for Sparse Reward Tasks*. arXiv:2110.02414 [cs]. Oct. 2021. DOI: 10.48550/arXiv.2110.02414. URL: http://arxiv.org/abs/2110.02414 (cit. on p. 17).

[NHM21] A. D. Noel, C. van Hoof, B. Millidge. *Online reinforcement learning with sparse rewards through an active inference capsule*. arXiv:2106.02390 [cs, stat]. June 2021. DOI: 10.48550/arXiv.2106.02390. URL: http://arxiv.org/abs/2106.02390 (cit. on p. 15).

[NKTK23] A. Nikulin, V. Kurenkov, D. Tarasov, S. Kolesnikov. *Anti-Exploration by Random Network Distillation*. arXiv:2301.13616 [cs]. Jan. 2023. DOI: 10.48550/arXiv.2301.13616. URL: http://arxiv.org/abs/2301.13616 (cit. on p. 17).

[Ope23] OpenAI. *GPT-4 Technical Report*. Mar. 27, 2023. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774[cs]. URL: http://arxiv.org/abs/2303.08774 (cit. on p. 20).

[PAED17] D. Pathak, P. Agrawal, A. A. Efros, T. Darrell. *Curiosity-driven Exploration by Self-supervised Prediction*. arXiv:1705.05363 [cs, stat]. May 2017. DOI: 10.48550/arXiv.1705.05363. URL: http://arxiv.org/abs/1705.05363 (cit. on p. 16).

[PAZ+21] L. Pineda, B. Amos, A. Zhang, N. O. Lambert, R. Calandra. *MBRL-Lib: A Modular Library for Model-based Reinforcement Learning*. arXiv:2104.10159 [cs, eess]. Apr. 2021. DOI: 10.48550/arXiv.2104.10159. URL: http://arxiv.org/abs/2104.10159 (cit. on pp. 28, 43).

[QPC20] J. Queeney, I. C. Paschalidis, C. G. Cassandras. *Uncertainty-Aware Policy Optimization: A Robust, Adaptive Trust Region Approach*. arXiv:2012.10791 [cs, stat]. Dec. 2020. DOI: 10.48550/arXiv.2012.10791. URL: http://arxiv.org/abs/2012.10791 (cit. on p. 18).

[RKS21] A. Raffin, J. Kober, F. Stulp. *Smooth Exploration for Robotic Reinforcement Learning*. June 20, 2021. arXiv: 2005.05719[cs,stat]. URL: http://arxiv.org/abs/2005.05719 (cit. on p. 44).

[RYH+21] S. Rammohan, S. Yu, B. He, E. Hsiung, E. Rosen, S. Tellex, G. Konidaris. *Value-Based Reinforcement Learning for Continuous Control Robotic Manipulation in Multi-Task Sparse Reward Settings*. arXiv:2107.13356 [cs]. July 2021. DOI: 10.48550/arXiv.2107.13356. URL: http://arxiv.org/abs/2107.13356 (cit. on p. 16).

[SB18] R. S. Sutton, A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018 (cit. on p. 20).

[SB20] R. S. Sutton, A. G. Barto. "Reinforcement Learning: An Introduction". en. In: (2020) (cit. on p. 12).

[SFG+09] B. K. Sriperumbudur, K. Fukumizu, A. Gretton, B. Schölkopf, G. R. G. Lanckriet. *On integral probability metrics, \phi-divergences and binary classification*. Oct. 12, 2009. DOI: 10.48550/arXiv.0901.2698. arXiv: 0901.2698[cs,math]. URL: http://arxiv.org/abs/0901.2698 (cit. on p. 33).

[SGK22] J. She, J. K. Gupta, M. J. Kochenderfer. *Agent-Time Attention for Sparse Rewards Multi-Agent Reinforcement Learning*. arXiv:2210.17540 [cs]. Oct. 2022. DOI: 10.48550/arXiv.2210.17540. URL: http://arxiv.org/abs/2210.17540 (cit. on p. 16).

[SHM+16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (Jan. 2016). Number: 7587 Publisher: Nature Publishing Group, pp. 484–489. ISSN: 1476-4687. DOI: 10.1038/nature16961. URL: https://www.nature.com/articles/nature16961 (visited on 06/01/2023) (cit. on p. 27).

[Sil15] D. Silver. *Lectures on Reinforcement Learning*. URL: https://www.davidsilver.uk/teaching/. 2015 (cit. on p. 25).

[SLM+17] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, P. Abbeel. *Trust Region Policy Optimization*. Apr. 20, 2017. DOI: 10.48550/arXiv.1502.05477. arXiv: 1502.05477[cs]. URL: http://arxiv.org/abs/1502.05477 (visited on 05/20/2023) (cit. on p. 25).

[SML+18] J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. Oct. 20, 2018. DOI: 10.48550/arXiv.1506.02438. arXiv: 1506.02438[cs]. URL: http://arxiv.org/abs/1506.02438 (visited on 05/20/2023) (cit. on p. 25).

[SPP18] R. K. Sinha, R. Pandey, R. Pattnaik. *Deep Learning For Computer Vision Tasks: A review*. Apr. 11, 2018. DOI: 10.48550/arXiv.1804.03928. arXiv: 1804.03928[cs]. URL: http://arxiv.org/abs/1804.03928 (cit. on p. 21).

[SRD+20] R. Sekar, O. Rybkin, K. Daniilidis, P. Abbeel, D. Hafner, D. Pathak. *Planning to Explore via Self-Supervised World Models*. June 30, 2020. DOI: 10.48550/arXiv.2005.05960. arXiv: 2005.05960[cs,stat]. URL: http://arxiv.org/abs/2005.05960 (cit. on p. 16).

[SSS+17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis. “Mastering the game of Go without human knowledge”. In: *Nature* 550.7676 (Oct. 2017). Number: 7676 Publisher: Nature Publishing Group, pp. 354–359. ISSN: 1476-4687. URL: https://www.nature.com/articles/nature24270 (cit. on pp. 12, 21).

[SWD+17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov. *Proximal Policy Optimization Algorithms*. arXiv:1707.06347 [cs]. Aug. 2017. DOI: 10.48550/arXiv.1707.06347. URL: http://arxiv.org/abs/1707.06347 (cit. on p. 25).

[TMD+20] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, Y. Tassa. “dm$_control : Softwareandtasksforcontinuouscontrol$”. In: *Software Impacts* 6 (2020), p. 100022. ISSN: 2665-9638. DOI: https://doi.org/10.1016/j.simpa.2020.100022. URL: https://www.sciencedirect.com/science/article/pii/S2665963820300099 (cit. on pp. 14, 39).

[TSK+21] A. Torfi, R. A. Shirvani, Y. Keneshloo, N. Tavaf, E. A. Fox. *Natural Language Processing Advancements By Deep Learning: A Survey*. Feb. 27, 2021. DOI: 10.48550/arXiv.2003.01200. arXiv: 2003.01200[cs]. URL: http://arxiv.org/abs/2003.01200 (cit. on p. 21).

[VEB+17] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, J. Quan, S. Gaffney, S. Petersen, K. Simonyan, T. Schaul, H. van Hasselt, D. Silver, T. Lillicrap, K. Calderone, P. Keet, A. Brunasso, D. Lawrence, A. Ekermo, J. Repp, R. Tsing. *StarCraft II: A New Challenge for Reinforcement Learning*. Aug. 16, 2017. arXiv: 1708.04782[cs]. URL: http://arxiv.org/abs/1708.04782 (cit. on p. 12).

[VGO+20] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2 (cit. on p. 34).

[VHS+18] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, M. Riedmiller. *Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards*. arXiv:1707.08817 [cs]. Oct. 2018. DOI: 10.48550/arXiv.1707.08817. URL: http://arxiv.org/abs/1707.08817 (cit. on p. 15).

[WBD+22] A. Wilcox, A. Balakrishna, J. Dedieu, W. Benslimane, D. S. Brown, K. Goldberg. *Monte Carlo Augmented Actor-Critic for Sparse Reward Deep Reinforcement Learning from Suboptimal Demonstrations*. arXiv:2210.07432 [cs]. Oct. 2022. DOI: 10.48550/arXiv.2210.07432. URL: http://arxiv.org/abs/2210.07432 (cit. on p. 16).

[WEH+22] P. Wu, A. Escontrela, D. Hafner, K. Goldberg, P. Abbeel. *DayDreamer: World Models for Physical Robot Learning*. June 28, 2022. DOI: 10.48550/arXiv.2206.14176. arXiv: 2206.14176[cs]. URL: http://arxiv.org/abs/2206.14176 (visited on 05/07/2023) (cit. on p. 13).

[WWW21] C. Wulur, C. Weber, S. Wermter. “Planning-integrated Policy for Efficient Reinforcement Learning in Sparse-reward Environments”. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. ISSN: 2161-4407. July 2021, pp. 1–8. DOI: 10.1109/IJCNN52387.2021.9533509 (cit. on p. 15).

[WWWZ20] C. Wang, J. Wang, J. Wang, X. Zhang. “Deep-Reinforcement-Learning-Based Autonomous UAV Navigation With Sparse Rewards”. In: *IEEE Internet of Things Journal* 7.7 (July 2020). Conference Name: IEEE Internet of Things Journal, pp. 6180–6190. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.2973193 (cit. on p. 15).

[YFH+21] R. Yang, M. Fang, L. Han, Y. Du, F. Luo, X. Li. *MHER: Model-based Hindsight Experience Replay*. arXiv:2107.00306 [cs]. Nov. 2021. DOI: 10.48550/arXiv.2107.00306. URL: http://arxiv.org/abs/2107.00306 (cit. on p. 18).

[YTY+20] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, T. Ma. “MOPO: Model-based Offline Policy Optimization”. In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 14129–14142. URL: https://proceedings.neurips.cc/paper/2020/hash/a322852ce0df73e204b7e67cbbef0d0a-Abstract.html (cit. on p. 19).

[ZLW20] Q. Zhou, H. Li, J. Wang. “Deep Model-Based Reinforcement Learning via Estimated Uncertainty and Conservative Policy Optimization”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.4 (Apr. 3, 2020). Number: 04, pp. 6941–6948. ISSN: 2374-3468. DOI: 10.1609/aaai.v34i04.6177. URL: https://ojs.aaai.org/index.php/AAAI/article/view/6177 (cit. on p. 18).

[ZZLL20] G. Zuo, Q. Zhao, J. Lu, J. Li. “Efficient hindsight reinforcement learning using demonstrations for robotic tasks with sparse rewards”. en. In: *International Journal of Advanced Robotic Systems* 17.1 (Jan. 2020). Publisher: SAGE Publications, p. 1729881419898342. ISSN: 1729-8806. DOI: 10.1177/1729881419898342. URL: https://doi.org/10.1177/1729881419898342 (cit. on p. 17).

[ZZW+19] B. Zhou, H. Zeng, F. Wang, Y. Li, H. Tian. *Efficient and Robust Reinforcement Learning with Uncertainty-based Value Expansion*. arXiv:1912.05328 [cs]. Dec. 2019. URL: http://arxiv.org/abs/1912.05328 (cit. on p. 13).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, 14th August 2023 Ravi Akash

place, date, signature