

# RAM

● ROBOTICS  
AND  
MECHATRONICS

## EXPLOITATION OF SYMMETRY IN REINFORCEMENT LEARNING AND TRANSFORMER-BASED REINFORCEMENT LEARNING FOR CONTROL TASKS

A. (Ammar) Alhannafi

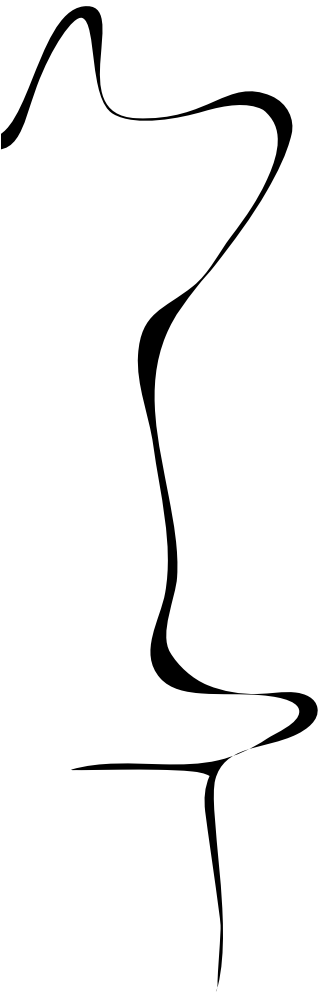
MSC ASSIGNMENT

**Committee:**

prof. dr. ir. S. Stramigioli  
dr. ir. F. Califano  
dr. R. Zanella  
dr. E. Mocanu

December, 2024

077RaM2024  
Robotics and Mechatronics  
EEMCS  
University of Twente  
P.O. Box 217  
7500 AE Enschede  
The Netherlands



## Summary

Robots operating in dynamic and unstructured environments require highly adaptable control mechanisms to handle uncertainty and variability. **R**einforcement **L**earning (RL) carries the potential to realize such mechanisms and has already demonstrated notable successes in robotics. However, challenges such as data inefficiency, poor generalization, and limited scalability still hinder its broader adoption.

This work aims to mitigate these challenges by proposing a unified framework that integrates symmetry exploitation with transformer-based decision models. Symmetry exploitation is formalized through the Markov Decision Process homomorphism framework, which abstracts redundancies arising from symmetries in the state-action space. This abstraction reduces the solution space and enables RL agents to converge faster, making them more sample-efficient and generalizable. Meanwhile, transformer architectures, renowned for their scalability in other domains, can be adapted to mitigate the scalability issues in RL.

The proposed framework was validated through implementation and evaluation in simulated environments, including discrete and continuous control tasks such as *CartPole* and *Inverted Pendulum*. Results demonstrate that symmetry-aware models not only converge faster but also generalize higher across equivalent states compared to conventional RL and standard transformer-based methods.

These findings confirm the hypothesis that exploiting symmetries in RL improves sample efficiency and model generalization across symmetric states and actions, aiding in putting a step forward in scalable robotic control.

## Acknowledgments

I would like to express my gratitude to everyone who supported me throughout the journey of completing my master's degree and this thesis. Without their guidance and encouragement, this work would not have been possible.

First, I am deeply thankful to my supervisors, Dr. Ir. F. Califano and Dr. R. Zanella, for their invaluable guidance and patience. They taught me how to become a better researcher and how to express my ideas. I must also say that my presentation skills have been improved thanks to their constructive feedback. I am equally grateful to the members of my committee, Prof. Dr. Ir. S. Stramigioli, for heading the committee, and Dr. E. Mocanu, for accepting to be part of the examination committee and helping to evaluate my work in the field of reinforcement learning.

To my wife, Esraa, I owe my deepest gratitude for her support, love, and patience throughout this challenging journey. Your belief in me gave me the strength to achieve this milestone.

To my family and friends, thank you for your unconditional support and encouragement. Your presence and belief in me have been invaluable.

I dedicate this humble work to the memory of my brother and my father.

*Ammar Alhannafi*  
*Enschede, November 2024*

---

## List of Acronyms

<b>RL</b>	<b>R</b> einforcement <b>L</b> earning
<b>DRL</b>	<b>D</b> eep <b>R</b> einforcement <b>L</b> earning
<b>PPO</b>	<b>P</b> roximal <b>P</b> olicy <b>O</b> ptimization
<b>TD3</b>	<b>T</b> win <b>D</b> elayed <b>D</b> eep <b>D</b> eterministic <b>P</b> olicy <b>G</b> radient
<b>MDP</b>	<b>M</b> arkov <b>D</b> ecision <b>P</b> rocess
<b>DP</b>	<b>D</b> ynamic <b>P</b> rogramming
<b>MC</b>	<b>M</b> onte <b>C</b> arlo
<b>TD</b>	<b>T</b> emporal <b>D</b> ifference
<b>NNs</b>	<b>N</b> eural <b>N</b> etworks
<b>DQN</b>	<b>D</b> eep <b>Q</b> - <b>N</b> etwork
<b>RNN</b>	<b>R</b> ecurrent <b>N</b> eural <b>N</b> etwork
<b>LSTM</b>	<b>L</b> ong <b>S</b> hort- <b>T</b> erm <b>M</b> emory
<b>CNN</b>	<b>C</b> onvolutional <b>N</b> eural <b>N</b> etwork
<b>NLP</b>	<b>N</b> atural <b>L</b> anguage <b>P</b> rocessing
<b>GPT</b>	<b>G</b> enerative <b>P</b> re-trained <b>T</b> ransformer
<b>BERT</b>	<b>B</b> idirectional <b>E</b> ncoder <b>R</b> epresentations from <b>T</b> ransformers
<b>RTG</b>	<b>R</b> eturn- <b>T</b> o- <b>G</b> o
<b>MLP</b>	<b>M</b> ulti- <b>L</b> ayer <b>P</b> erceptrons
<b>ReLU</b>	<b>R</b> ectified <b>L</b> inear <b>U</b> nit

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Motivation . . . . .	2
1.3	Research Statement . . . . .	3
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	Reinforcement Learning (RL) . . . . .	5
2.2	Transformer Architectures . . . . .	12
2.3	Symmetry in Reinforcement Learning for control . . . . .	17
<b>3</b>	<b>Methodology</b>	<b>25</b>
3.1	Finite MDP Homomorphic Networks . . . . .	25
3.2	Continuous MDP Homomorphic Networks . . . . .	29
3.3	MDP Homomorphic Decision Transformer . . . . .	33
<b>4</b>	<b>Experiments</b>	<b>37</b>
4.1	Environments . . . . .	37
4.2	Metrics . . . . .	39
4.3	Implementation and Fine-tuning Procedure . . . . .	41
4.4	Results . . . . .	42
<b>5</b>	<b>Conclusion and Recommendations</b>	<b>47</b>
5.1	Conclusion . . . . .	47
5.2	Recommendations . . . . .	47
<b>A</b>	<b>Continuous Optimal Value Equivalence</b>	<b>55</b>
<b>B</b>	<b>Universal Approximations of Equivariant Maps by Neural Networks</b>	<b>57</b>
<b>C</b>	<b>TD3 Algorithm</b>	<b>59</b>
<b>D</b>	<b>Reacher Environment</b>	<b>61</b>

## List of Figures

1.1	A reflection symmetry example: the state on the right mirrors the state on the left over the vertical axis. Consequently, the same control strategy that stabilizes the pole on one left can be applied to the right. . . . .	3
1.2	The proposed unified framework that combines symmetry, transformer architectures, and RL . . . . .	4
2.1	The agent–environment interaction in reinforcement learning. . . . .	6
2.2	Comparison between traditional Q-Learning (tabular) and Deep Q-Learning (utilizing neural networks). Here, a simple greedy policy is used just for demonstration purposes. . . . .	10
2.3	Illustration of Offline Reinforcement Learning during training and deployment phase. . . . .	11
2.4	Decision Transformer architecture, where states, action, and returns are embedded separately using normal linear layers with a learned positional encoding layer added. The input propagates through these layers, a GPT architecture, and ends with a linear decoder which decodes the optimal action	13
2.5	Architecture of the Trajectory Transformer. States, actions, rewards, and RTG are discretized and embedded. The GPT-based architecture processes this, and a linear decoder predicts the next output in the sequence. These sequences are then evaluated by the Beam Search based on RTG values, which outputs the most promising trajectories that have yielded the highest cumulative rewards. . . . .	14
2.6	(a) Causal self-attention mechanism for an input sequence of size 4 with an embedding dimension of 5. The $(\cdot)$ denotes a matrix multiplication operation (b) Single block structure of the GPT model. . . . .	16
2.7	A grid world example of MDP model minimization for symmetrical state-action pairs. On the left is the original MDP, while on the right is the reduced version over the counter diagonal axis. . . . .	17
2.8	Illustration of bijective and surjective properties . . . . .	18
2.9	Illustration a homomorphism in the grid world example, the homomorphism is a surjection from the original $\mathcal{M}$ to its abstract version $\bar{\mathcal{M}}$ . Equivalent state-action pairs in $\mathcal{M}$ are grouped into a block, which is then mapped by $h$ to a single state-action pair in $\bar{\mathcal{M}}$ . This process preserves the transition structure and rewards between $\bar{\mathcal{M}}$ and $\mathcal{M}$ . . . . .	19
2.10	Illustration of the MDP homomorphism where we display the relation between the original MDP and its reduced abstract version. . . . .	21
3.1	Illustration of the equivariance and invariance properties of the policy and value functions under the automorphisms $h^e$ and $h^1$ in the pole balancing problem. . . . .	27
3.2	Illustration of continuous rotational symmetry in the Gym Reacher environment, where equivalent action sequences can result from rotating the target and the first link by the same angle. . . . .	30

3.3	Illustration of the equivariance and invariance properties of the policy and Q-value function under automorphisms in the pole-balancing problem, where the state and action spaces are continuous. . . . .	32
3.4	Illustrative example of trajectory mapping under homomorphism in a grid-world problem, where the dotted cells are barriers. . . . .	33
3.5	Illustration of the mechanistic approach, showing the incremental addition of components to the residual stream. The equations on the right represent the transformations applied in each step starting from input sequence $X$ until output $Y$ . . . . .	36
4.1	Training curves for episodic returns on CartPole environment. The orange curve represents the standard PPO, while the blue curve represents the MDP homomorphic PPO. The MDP homomorphic PPO converges faster to the maximum episodic return of 500, which indicates greater sample efficiency. . . . .	42
4.2	Training curves for cumulative average returns CartPole environment. The orange curve represents the standard PPO, while the blue curve represents the MDP homomorphic PPO. The MDP homomorphic PPO accumulates higher returns more quickly, which also indicates greater sample efficiency. . . . .	43
4.3	Training curves for episodic returns on the Inverted Pendulum environment. The orange curve represents the standard TD3, while the blue curve represents the MDP homomorphic TD3. The MDP homomorphic TD3 converges faster, which indicates greater sample efficiency also in continuous settings. . . . .	44
4.4	Training curves for cumulative average returns on the Inverted Pendulum environment. The orange curve represents the standard TD3, while the blue curve represents the MDP homomorphic TD3. The MDP homomorphic TD3 accumulates higher returns more quickly, which indicates greater sample efficiency in continuous settings. . . . .	44
4.5	Comparison between the equivariance errors for the policy network of MDP homomorphic TD3 and their standard versions. The equivariance error for the policy of MDP homomorphic TD3 remains near zero throughout training, which indicates higher generalization across symmetric states. . . . .	45
4.6	Comparison between invariance errors for the Q-value network of MDP homomorphic TD3 and their standard versions. The invariance error in the Q-Network of MDP homomorphic TD3 starts high but decreases steadily during training, which reflects improved generalization over time as symmetry constraints are acting here as regularizers. . . . .	45
4.7	Learning curves of episodic returns for standard Decision Transformer and MDP Homomorphic variant. . . . .	46
4.8	Equivariance error during training for the MDP Homomorphic Decision Transformer and MDP Homomorphic variant. . . . .	46
D.1	90-degree rotation is applied to the first joint and the target, where it can be intuitively interpreted as two equivalent state-action pair . . . . .	61

## List of Tables

4.1	Comparison of CartPole and Inverted Pendulum Environments . . . . .	38
4.2	Automorphisms and their matrix representation for CartPole and Inverted Pendulum . . . . .	38





# 1 Introduction

## 1.1 Context

In recent years, there has been a growing demand for robots capable of performing complex tasks in unstructured environments and beyond industrial settings. The tasks include autonomous navigation through crowded urban spaces [1], object manipulation in cluttered and dynamic environments [2], or collaborative tasks alongside humans such as home assistance task[3]. Such tasks pose challenges for traditional model-based control in accurately modeling systems characterized by high uncertainty, variable dynamics, and interactions with ever-changing and unpredictable surroundings [4].

The rise of RL has introduced a new paradigm for approaching such complex tasks. In RL, an agent learns to make an optimal decision by continuously interacting with the environment and refining its decision-making process through a feedback signal [5]. The success of RL is mainly rooted in integrating deep learning as powerful function approximators, which then become known as **Deep Reinforcement Learning (DRL)**. Notable achievements include surpassing human performance on some Atari games [6] and mastering the game of AlphaGo [7] that combined deep learning and RL techniques to defeat professional Go players in a game known for its complexity and large search space. In robotics, DRL enables robots to learn to map sensory input directly into low-level actions (control output) with minimal engineering effort and without the need for explicit dynamical models [5]. Furthermore, DRL has enabled robots to learn locomotion skills like walking [8] and dexterous manipulation tasks such as doors opening with robotic multi-fingered arm [9].

Despite these successes, applying RL in robotics has exposed several challenges. These include **data inefficiency**, where large quantities of interactions (data) are required for training, often on the order of millions of interactions [10]. Take for example the popular spinning-up benchmark by OpenAI for simulated robotic control tasks, where 3 million interactions are performed to evaluate different state-of-the-art RL algorithms on reward performance. Another challenge is **poor generalization** or overfitting, where the RL algorithm might fail in unseen scenarios or even struggle to produce similar actions (control output) to similar states [11]. Additionally, **lack of interpretability**, it is not easy due to the black-box nature of DRL to understand how and why certain decisions are made by the trained policy (controller) [12]. Lastly, in DRL, **scalability** refers to the ability of an algorithm to handle increasingly complex tasks with large state space. As the tasks become more complicated, larger neural networks are often utilized for their greater expressiveness and capacity, which enable them to model and solve these tasks. This approach, which originated from successes in fields like computer vision and natural language processing, seems intuitive but does not directly translate to DRL. That is to say, simply increasing network size does not consistently lead to better performance in DRL. Instead, it often introduces **stability issues**, such as divergence, oscillations, or even complete training collapse [13].

To mitigate challenges such as **scalability** and **interpretability**, the recent usage of transformer architectures (i.e. the backbone of OpenAI ChatGPT) in RL might offer potential solutions [14], [15]. Transformers have proven successful in natural language processing (NLP) and computer vision (CV) due to their ability to handle long-range dependencies within data, allowing them to make sense of sequences, whether they are sentences, series of images, or even decision-making sequences [16], [17], [18]. Applying transformer architectures to RL can mitigate issues of **scalability** by managing complex tasks and alleviating **RL-stability** issues when using larger model sizes or larger datasets

[16]. The self-attention mechanism inherent in transformers enables models to focus on relevant states and actions, and thus by visualising the attention weights as heatmaps, it becomes easy to **interpret** which parts of the environment affect the decision-making [15]. In Robotics, transformer architectures are proving to be the architecture of choice for large-capacity models not only for single robotic tasks but also for multi-task and multi-skill robotic systems. These models can be applied to both high-level task planning and low-level, end-to-end control [19]. Examples of successful transformer-based models include **RT-1** [20], **RT-2** [21], **RT-X** [22], **PACT** [23], and **SMART** [24].

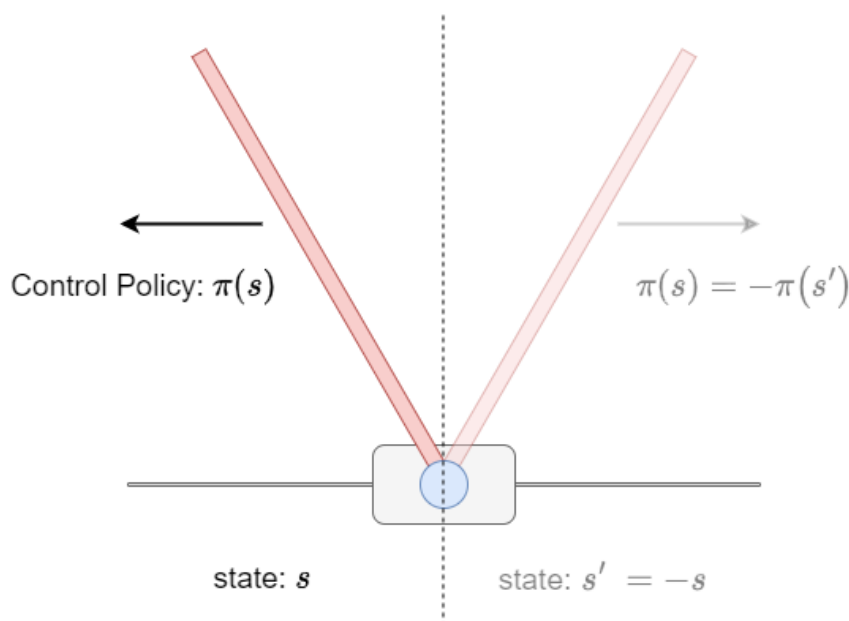
Furthermore, a promising approach to mitigate challenges such as **data inefficiency** and **poor generalization** involves incorporating prior knowledge into the learning process, rather than relying solely on data. This prior knowledge can take different forms, including physical laws (e.g., conservation laws), symmetrical properties of systems (e.g., rotational or translational invariance), or intuitive human understanding of physical interactions (e.g., object permanence) [25], [26]. Leveraging such prior knowledge can improve both sample efficiency and generalization by narrowing the solution search space and allowing learned policies to be reused across different states, which leads to faster learning and improved generalization [25], [26].

By integrating prior knowledge with transformer architectures, we can mitigate key challenges in applying RL to robotics. On the one hand, Transformer architectures offer an inherent advantage in scalability, which renders it possible to handle increasingly complex tasks with large state spaces while also providing interpretable decision-making through their self-attention mechanism. On the other hand, prior knowledge guides the learning process toward more sample-efficient solutions, which also serve to generalize to new situations or similar states.

## 1.2 Motivation

Many control tasks exhibit structural symmetries which can be defined as repeated structures detected by transformations such as reflection, or rotation that when applied to the system do not alter its control behavior. Take for example, the pole-balancing problem [5], also known as the cart-pole or inverted pendulum problem. Here, the controller (or policy) should keep a pole in an upright position by exerting a force on a movable cart, either towards the right or left, to prevent it from falling over. In this problem, we can intuitively identify a reflection symmetry over the vertical axis, as described by [27] and illustrated in Figure 1.1: "Balancing a pole that falls to the right requires an equivalent, but mirrored, strategy to one that falls to the left". Therefore, when learning control strategy in RL, a reflection symmetry such as the one in the pole-balancing problem if exploited, allows the agent to generalize its experience across equivalent states and actions, which in turn reduces the complexity of the problem, enables sample efficiency, and speeds up the learning process [27].

In RL, there have been numerous successful attempts to leverage symmetry, starting with the pioneering work of Balaraman Ravindran and G. Barto in 2001 [28]. Their model minimization framework used the notion of homomorphism to derive smaller, equivalent models of problems that exhibit symmetries. Fast-forward to recent years, researchers have extended this framework by incorporating symmetry into neural networks [29], [27]. The novelty of these works lies in explicitly constraining neural networks to reflect the symmetries inherent in the environment, which offers a more robust approach compared to other implicit methods. That is to say, implicit methods such as data augmentation [30], state space manipulation [31], or using auxiliary loss function [32] encourage a neural network to respect certain symmetries, but they do not guarantee it.



Furthermore, despite the successes of integrating transformer architecture into the RL framework, these models still suffer from the **sample inefficiency** problems, they require namely vast amounts of data to generalize to unseen scenarios and perform well.

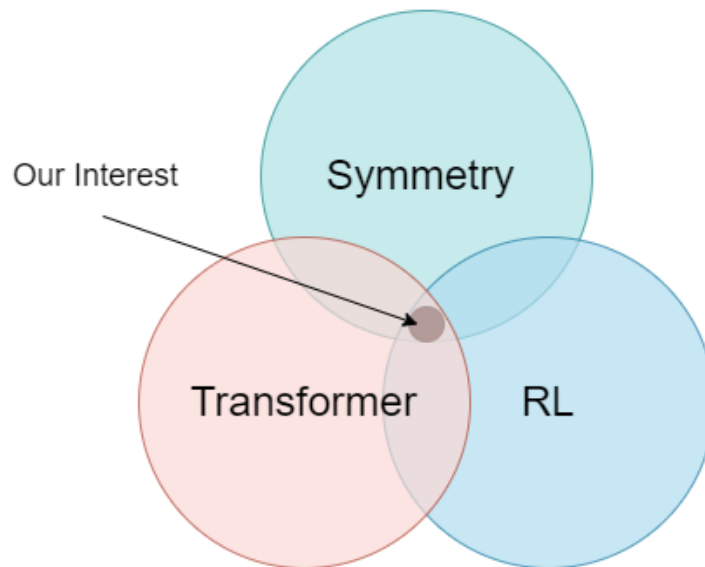
By combining the symmetry exploitation for more efficient learning and generalization, with the scalability and interpretability of transformer architectures, this work attempts to create a new synergy in RL to solve robotic tasks. While this work does not attempt to tackle all of the challenges previously mentioned, it specifically focuses on improving sample efficiency and generalization in simple, single-task scenarios within simulated environments that involve both continuous and discrete action spaces. These tasks will utilize direct state information as input, such as sensor measurements (e.g., positions, velocities, and angles), commonly found in robotics.

To incorporate symmetry into the transformer architecture, this work will build on the model minimization framework i.e., **Markov Decision Process (MDP)** homomorphism proposed by [28]. For the implementation of symmetry constraints, we will draw from the methodology outlined by [27], which serves as the foundation for incorporating symmetry into the transformer neural layers.

### 1.3 Research Statement

The main objective of this thesis is to explore and develop a unified framework that integrates symmetry, transformer architectures, and RL to mitigate the challenges of sample efficiency and generalization in control tasks in robotics. The specific components of this research include:

- **Symmetry into RL Algorithms:** This component will explore how symmetry constraints can improve the efficiency and generalization of RL algorithms. The first step will replicate the work of [27] to verify that incorporating symmetry into the **Proximal Policy Optimization (PPO)** [33] algorithm can speed up learning on tasks with discrete action spaces. Success will be evaluated based on the convergence speed



**Figure 1.2:** The proposed unified framework that combines symmetry, transformer architectures, and RL

compared to conventional PPO implementations. In the second part, we will extend the theoretical framework of MDP homomorphism by [28] from discrete to continuous settings and enable symmetry exploitation in **Twin Delayed Deep Deterministic Policy Gradient (TD3)** [34] which is a powerful algorithm proved to work effectively on control tasks with continuous state and action spaces. We hypothesize that continuous and discrete symmetry exploitation in RL algorithms in general, so not only for PPO, will converge faster and generalize better in both discrete and continuous control tasks.

- **Symmetry into Transformer-Based RL:** This component will focus on establishing the theoretical foundation to allow transformer-based RL models to become symmetry-aware, and we will investigate how symmetry constraints can be implemented in Transformer architecture. This component will build upon and extend the decision transformer framework used in RL [14].
- **Evaluation on Benchmark:** The framework will be tested on tasks from the Gym benchmark [35], such as the Cartpole, Inverted Pendulum, and Reacher, which involve continuous and discrete control tasks. These tasks will use direct sensor measurements (e.g., positions, velocities, and angles) as inputs. The performance of the proposed symmetry-aware transformer model will be compared against its conventional version in terms of sample efficiency, and generalization to new scenarios or similar states presented in the environment.

## 2 Background

This chapter presents the concepts needed in order to fully comprehend the subjects discussed in this thesis. We start by reviewing the RL framework and its mathematical formalization as MDP. Next, we explain transformer architectures as the core element of the Decision Transformers and Trajectory Transformers. Finally, we discuss how symmetries can enhance RL by improving sample efficiency and generalization. The framework where we exploit symmetries as a formal tool is also introduced, followed by the methods of how to enforce symmetries during model training.

### 2.1 Reinforcement Learning (RL)

*Note: The majority of the content in this section is based on the standard textbook of Sutton and Barto 2018 [5].*

RL is a machine learning paradigm that teaches an agent how to make a decision in a dynamic environment. The teaching process involves learning through environmental interactions. That is to say, an agent interacts with the environment at each timestep and receives feedback as a reward. Based on this reward, the agent should alter its behavior and thus improve its policy in order to maximize the reward summed over all future timesteps (i.e. cumulative reward). Therefore, the agent is not guided and never told which actions are correct, but it should discover which sequence of actions yields the best cumulative rewards.

The inspiration for such a learning approach can be rooted back to animal learning theories, these theories state that actions followed by rewarding outcomes are more likely to be repeated as a result of receiving a stimulus (reinforcer). This approach also bears a strong resemblance to human learning, where the cortico-basal ganglia loops adjust the behavior by processing both immediate and future rewards, as well as past experiences. These neural loops, much like RL models, use predictions and feedback to guide decision-making over time [36].

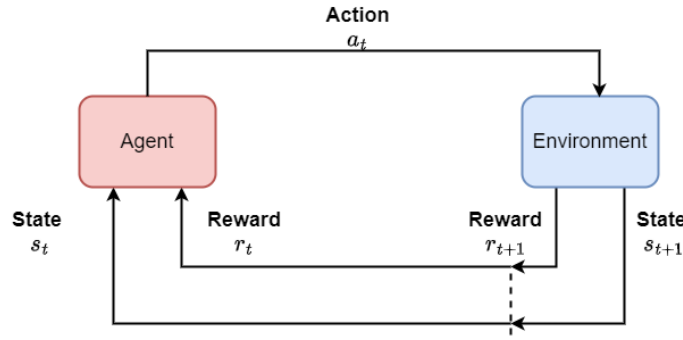
Terms such as **Agent**, **Reward**, and **Environments** mentioned previously, along with other related elements, form the key elements of the RL framework. These components are explained through the lens of the MDP framework in the following section. Additionally, the main approaches to solving RL problems are also discussed. Finally, the concepts of Deep Reinforcement Learning (DRL) and Offline Reinforcement, which are extensions of traditional RL that we build upon in this work, are also explained in sections 2.1.2, and 2.1.3 respectively.

#### 2.1.1 Markov Decision Processes (MDP)

The MDP provides a mathematical formalism to sequential decision-making problems, and it is deemed an ideal modeling approach for problems involving agent-environment interactions, as the case with RL, see Figure 2.1. Additionally, the MDP enables us to formally define the objective of our problem and provides a structured way to achieve it. Furthermore, the reason behind the name is that the MDP model adheres to the Markov property, which can be stated as:

**Definition 2.1: Markov Property [5]**

The future state depends only on the current state and action, not on the sequence of events that preceded it.



**Figure 2.1:** The agent–environment interaction in reinforcement learning. Diagram is adapted from [5]

Mathematically speaking, a MDP is defined by the 5-tuple  $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$ , where  $\mathcal{S}$  is the set of all possible states  $s \in \mathcal{S}$ , which can be continuous or discrete.  $\mathcal{A}$  is the set of actions  $a \in \mathcal{A}$ , also either continuous or discrete.  $\mathcal{T}(s_{t+1} | s_t, a_t)$  denotes the transition probability from state  $s_t$  to  $s_{t+1}$ , representing the underlying dynamics of the system.  $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  is the reward function, and  $\gamma \in (0, 1]$  is the discount factor.

Under the MDP framework, a trajectory  $\tau$  is defined as the sequence of states and actions of length  $H$ :

$$\tau = (s_1, a_1, s_2, a_2, \dots, s_H, a_H), \quad (2.1)$$

Where  $H$  represents the horizon, which is the maximum number of time steps allowed in each episode. This can either be finite (the finite-horizon case) or infinite (the infinite-horizon case). The total return for a given trajectory  $\tau$  is defined as the sum of all discounted rewards:

$$R(\tau) := \sum_{t=0}^H \gamma^t r_t \quad (2.2)$$

Here,  $\gamma \in (0, 1]$  is the discount factor. When  $\gamma = 1$ , typically used in finite-horizon problems, future rewards are not discounted and are valued equally to immediate rewards. On the other hand, when  $0 < \gamma < 1$ , future rewards are discounted, placing more emphasis on immediate rewards.

A policy  $\pi$  specifies the action(s) to be taken for each state. A stochastic policy  $\pi$  provides a probability distribution over actions for each state, denoted as  $\pi(a|s)$ . A deterministic policy, on the other hand, directly assigns a single action to each state, denoted as  $a = \pi(s)$ <sup>1</sup>.

The expected return of a policy  $\pi$  is expressed as:

$$\eta(\pi) := \mathbb{E}^\pi[R(\tau)] \quad (2.3)$$

Here,  $\mathbb{E}^\pi$  indicates that all actions are drawn according to the policy  $\pi$ . The main objective of reinforcement learning is to determine a policy that maximizes the expected return:

$$\pi^* = \arg \max_{\pi \in \Pi} \eta(\pi) \quad (2.4)$$

Where  $\Pi$  represents a set of candidate policies.

<sup>1</sup>Deterministic policies can be viewed as a subset of stochastic policies where the probability distribution is a Dirac delta function focused on one action

In addition to the terms introduced previously, we now introduce some quantities of interest that aid in solving the RL objective in (2.4). These quantities, which are called **value functions**, indicate how good and rewarding a particular state or action is and they are used in almost every RL algorithm.

The state-value function  $V^\pi(s)$  represents the expected cumulative reward received by the agent starting from state  $s$  and following a policy  $\pi$  thereafter:

$$V^\pi(s) := \mathbb{E}^\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s \right] \quad (2.5)$$

Similarly, the action-value function  $Q^\pi(s, a)$  represents the expected cumulative reward received by the agent by taking action  $a$  in state  $s$  and continuing to follow the policy  $\pi$ :

$$Q^\pi(s, a) := \mathbb{E}^\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a \right] \quad (2.6)$$

We can further introduce the **advantage function**  $A^\pi(s, a)$ , which measures how much better taking an action  $a$  in state  $s$  is compared to the average action in that state under policy  $\pi$ . It is defined as:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \quad (2.7)$$

The functions defined in equations (2.6), (2.5) can be redefined in recursive form, known as the Bellman Expectation Equations, by decomposing the expected return into two parts, namely; the immediate reward at the current step and the discounted future value function from the next step.

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{s' \sim \mathcal{T}} \left[ r(s, a) + \gamma \mathbb{E}^\pi \left[ \sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \mid s_1 = s' \right] \right] \\ &= \mathbb{E}_{s' \sim \mathcal{T}} \left[ r(s, a) + \gamma \mathbb{E}^\pi \left[ \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \mid s_0 = s' \right] \right] \\ &= \mathbb{E}_{s' \sim \mathcal{T}} [r(s, a) + \gamma V^\pi(s')]. \end{aligned} \quad (2.8)$$

This is a recursive form because the state-value function at  $s$  depends on the state-value function of the next state  $s'$ . Furthermore, since the state-value function  $V^\pi(\mathbf{s})$  is defined as the expected value of  $Q^\pi(\mathbf{s}, \mathbf{a})$ :

$$V^\pi(\mathbf{s}) = \mathbb{E}_{\mathbf{a} \sim \pi} [Q^\pi(\mathbf{s}, \mathbf{a})] \quad (2.9)$$

We can combine these equations to obtain the following recursive form of  $Q^\pi(\mathbf{s}, \mathbf{a})$ :

$$Q^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \mathbb{E}_{s' \sim \mathcal{T}} [V^\pi(s')] \quad (2.10)$$

Finally, the **optimal state-value function**  $V^*(s)$  is the maximum value function over all policies. Similarly, the **optimal action-value function**  $Q^*(s, a)$  is the maximum value-action function over all policies:

$$\begin{aligned} V^*(s) &= \max_{\pi} V^\pi(s) \\ Q^*(s, a) &= \max_{\pi} Q^\pi(s, a) \end{aligned} \quad (2.11)$$



### Common Approaches to Solve RL Problems

With the formalization of MDP and the brief introduction of RL elements, several fundamental frameworks for learning can be introduced which are commonly used when approaching RL problems:

#### Dynamic Programming (DP)

DP assumes complete knowledge of the environment dynamics (i.e., the transition function  $\mathcal{T}(s_{t+1} | s_t, a_t)$  and reward function  $r(s, a)$ ) and finds an optimal solution by breaking down the problems into sub-problems and reusing the optimal solution to sub-problems in order to find the overall optimal one. This approach allows the optimal solution to be found iteratively by the subsequent evaluation and improvement of either value function or policy as noted by the two main DP algorithms: Policy and Value Iteration.

#### Monte Carlo Methods

Monte Carlo (MC) methods do not require knowledge of the transition dynamics, unlike the DP methods. Instead, they approximate value functions by averaging the returns from sampled trajectories (i.e., complete episodes). MC methods perform value function updates only after an entire trajectory (episode) is completed in the environment. Popular MC algorithms include Monte Carlo Policy Evaluation with its two versions; First Visit Monte Carlo and Every Visit Monte Carlo.

#### Temporal Difference (TD) Learning

Similar to MC methods, TD learning is a model-free approach, meaning it does not require knowledge of the transition dynamics of the environment. However, unlike MC methods, they perform value function updates incrementally after each time step in the trajectory. Two common TD algorithms are: **SARSA** is an on-policy method that updates the action-value function based on the current policy. **Q-learning** is another popular algorithm which is an off-policy method that directly learns the optimal policy, independent of the current policy.

### 2.1.2 Deep Reinforcement Learning (DRL)

Traditionally, all RL methods introduced in the previous section, like Q-Learning or Monte Carlo Policy Evaluation, utilize tabular methods in which the state and action spaces are sufficiently small for the value functions to be exactly represented as a table or array. However, when the problem involves a large state and action space or it is continuous in nature, such as the case with the robotic control problems, representing exact value function becomes intractable. This paves the way to the utilization of approximate methods, where our goal is not to find an exact solution (i.e. exact optimal policy or optimal value function) but an approximate one [37]. The approximate solution should be learned from a limited subset of the state and action spaces and then it should be generalized to unseen states and actions [37].

Among these approximate methods are those that utilize **Neural Networks** (NNs) as non-linear function approximators. NNs are computational models inspired by the human brain. They consist of layers of interconnected units called neurons (similar to biological neurons), where each neuron is connected to other neurons via connections called weights (similar to synapses). These weights, typically symbolized as  $\theta$ , determine the strength of the connection between different neurons. Additionally, similar to the action potential in human brain cells that introduces non-linearity [38], NNs rely on activation functions such as **Rectified Linear Unit** (ReLU) as non-linear transformations applied to the output of each neuron. Furthermore, NNs can learn complex relationships between inputs (e.g., the current state of an environment) and outputs (e.g., actions to take) by progressively updating the weights until the relationship is approximated as closely as possible to the desired one.

The integration of NNs and RL has a long history, highlighted by several key works. TD-Gammon [39] paper is one of the earliest works on using NNs for value function approximation in RL, where an agent is learned to play backgammon at an expert level. Followed by the significant contribution of [40], which initiates the use of gradient-based methods (i.e. a method to update the neural network weights) to immediately learn a representation of a policy without the use of value functions, an algorithm known as REINFORCE. Additionally, the seminal work on the **Deep Q-Network** (DQN) by [6] enabled NNs to learn the action-value function  $Q(s, a)$  directly from high-dimensional inputs such as images. DQN allowed an agent to achieve human-level performance in playing Atari games, where the state space (pixel values from the game screen) was far too large for traditional methods.

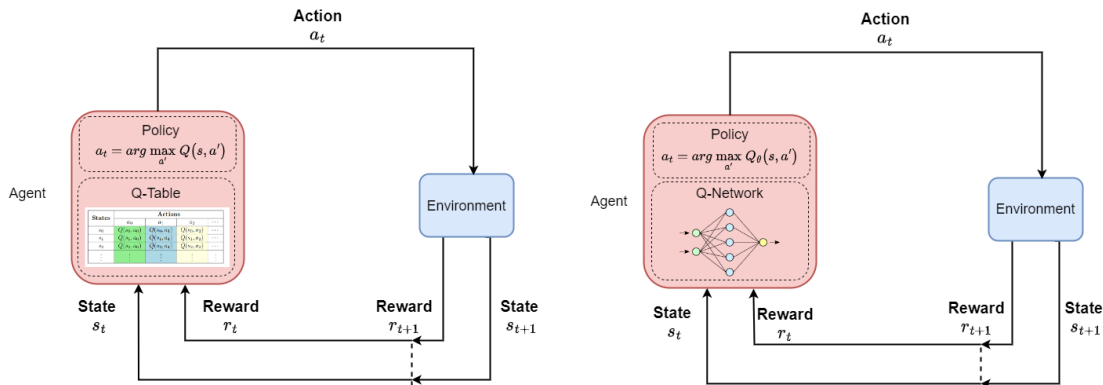
These seminal works, along with others, gave rise to the Deep Reinforcement Learning (DRL) sub-field. In DRL, the term "deep" refers to the use of many layers in the NNs with the intuition of composing many nonlinear layers can enable learning very complex functions [41]. Building on advancements in deep learning, DRL involves the use of different deep architectures such as **Convolutional Neural Network** (CNN), **Recurrent Neural Network** (RNN), and **Long Short-Term Memory** (LSTM), which are tailored to handle different types of data (e.g. images or raw sensory data).

This combination has introduced many state-of-the-art algorithms. To encapsulate them in one taxonomy, researchers have introduced a categorization of these methods into three main approaches [5], [37]:

- **Value-Based methods:** These methods approximate the state value function  $V_\theta(s)$  or the action-state value function  $Q_\theta(s, a)$  using deep neural networks, where  $\theta$  represents the parameters (weights) of the network. These methods then derive an implicit policy that maximizes the estimated value functions. Examples include DQN and its variants, such as double DQN [42] and Dueling DQN [43].

- **Policy-Based Methods:** Instead of learning value functions, policy-based methods learn directly the policy  $\pi_\theta$  which maps states to actions, where  $\theta$  represents the parameters (weights) of the network. Examples include Deterministic Policy Gradient (DPG) and REINFORCE; also known as Monte-Carlo policy gradient [5].
- **Actor-Critic Methods:** These methods combine the merits of the value-based and policy-based methods. A deep network acts as a critic and estimates the value function, while a separate deep network acts as an actor and learns the policy in the direction suggested by the critic. Examples of actor-critic methods include PPO [33], and TD3 [34].

Finally, we conclude this section by illustrating the key difference between traditional tabular Q-learning and its neural network-based approximation version in Figure 2.2. As indicated previously, traditional Q-learning relies on a Q-table to store the value of state-action pairs, while Deep Q-Network utilizes neural networks for approximating the Q-value function.



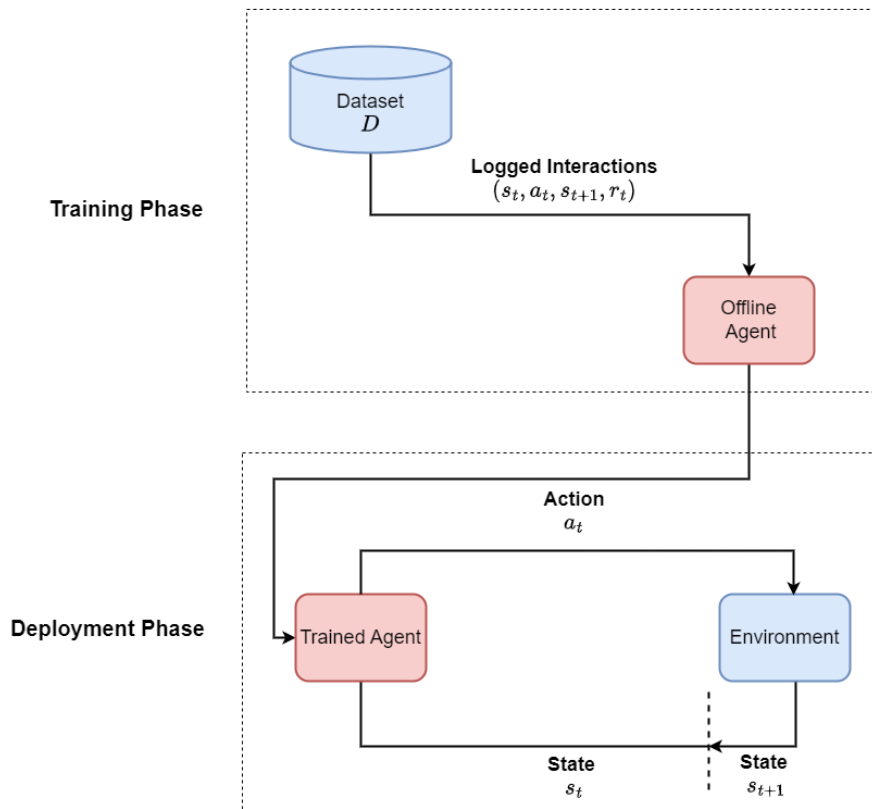
**Figure 2.2:** Comparison between traditional Q-Learning (tabular) and Deep Q-Learning (utilizing neural networks). Here, a simple greedy policy is used just for demonstration purposes.

### 2.1.3 Offline Reinforcement Learning

The learning paradigm in reinforcement learning mainly revolves around continuous interaction with the environment in an online fashion, that is to say, an agent initially explores random actions and improves its behavior upon the reward it receives. At the start of the learning process, the agent **explores** highly random action, but as the agent learns, it starts to **exploit** its knowledge and eventually reduces its random exploration as it converges towards an optimal policy, if at least one exists [5]. If applied to real-world problems, the randomness in the actions during exploration might lead to risky events and be prohibitively expensive. This also applies to the later learning stages, as the exploitation of the agent knowledge may still be sub-optimal [44]. Take for example, the utilization of RL in real-world robotic manipulators, during the exploration phase the robot might perform unpredictable and uncontrolled actions that could harm humans standing in the vicinity of them, and if not it might lead to wear and tear of the robot itself. Additionally, a robotic manipulator learning a variety of robotic manipulation skills needs a huge number of interactions to converge to an optimal policy, which also incurs expensive costs.

Therefore, to mitigate the previous challenges, researchers see the potential in utilizing offline RL, in which an agent learns an optimal policy on previously collected data, without any further interaction with the environment, as can be seen in Figure 2.3. Thus any previously collected data for a specific application can be reused to train an agent, similar

to supervised machine learning. In this work, we train the Transformer-based approaches in an offline fashion.



**Figure 2.3:** Illustration of Offline Reinforcement Learning during training and deployment phase.

## 2.2 Transformer Architectures

The Transformer entered the world of deep learning as a powerful architecture with the publication of "Attention Is All You Need" paper by [45]. The term "Attention" in the title refers to the use of a mechanism called self-attention, which enables the model to capture long-range relationships within data. This capability has enabled the transformer to outperform other architectures, such as RNN and LSTM, in sequence-to-sequence machine translation tasks. Furthermore, due to its highly expressive power and ability to model large-scale contextual data, this architecture has seen a lot of success in other fields such as **Natural Language Processing (NLP)** [46], [47], computer vision (vision transformers) [48], audio and video processing [49], [50], and reinforcement learning [16], [17], [18]. Moreover, its large modeling capacity gives birth to new architecture variations that are trained on internet-scale datasets, two known examples are:

- **Generative Pre-trained Transformer (GPT)**: A unidirectional Transformer variant used for automatic text generation, such as OpenAI's GPT-x models (e.g., GPT-2, GPT-3, and GPT-4) [46], [51],[52].
- **Bidirectional Encoder Representations from Transformers (BERT)**: A bidirectional Transformer variant used for various NLP tasks, including improving Google's search engine capabilities [47].

As demonstrated by models like ChatGPT, one can provide a seed sentence to generate an article on specific topic, and receive a well-structured, contextually relevant article. This raises a natural question whether we can apply a similar concept in RL. That is to say, if we provide trajectories of states, actions, and rewards to train GPT-like models, could the model then generate an optimal trajectory that maximizes cumulative rewards?

This question was the motivation for two pioneering works i.e. Decision Transformer [14] and Trajectory Transformer [15]. These works attempt to re-frame the RL as sequence modeling problem and addressing it conditionally using high-capacity sequence model architectures such as GPT. This re-framing differs from traditional RL approaches, which focuses on finding an optimal value functions or policy. Instead, it obtains an auto-regressive model which by conditioning on desired rewards, past states, and past actions, it predicts future actions that maximize rewards. Additionally, both models leverage the concept of Offline RL, where an agent is trained in a supervised fashion using previously collected data, adapting to the training method of transformer.

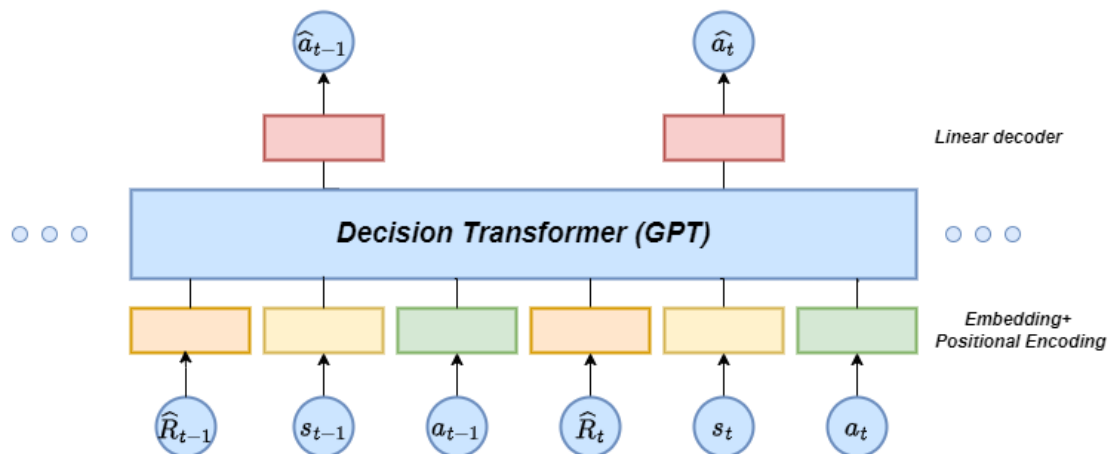
### 2.2.1 Decision and Trajectory Transformers

*Note: The majority of the content in this section is based on two references [14], [15] unless otherwise cited.*

The Decision and Trajectory Transformers, illustrated in Figures 2.4, 2.5, are quite similar in their working principles when applied in offline RL settings. Starting with the Decision Transformer, it learns directly a policy by conditioning the next action on the past trajectory and future **Return-To-Go (RTG)** values. Its focus lies solely on action prediction without any explicit modeling of state and reward transitions. Decision Transformer represent a trajectory as sequences of states  $s_t$ , actions  $a_t$ , and RTG values  $\hat{R}_t$  at different time steps:

$$\tau_{dt} = (\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, \dots, \hat{R}_T, s_T, a_T)$$

The RTG is the sum of rewards from the current time step until the end of the episode  $\hat{R}_t = \sum_{t'=t}^T r_{t'}$ , and it is used instead of immediate rewards to guide the model choosing the actions which maximize the cumulative reward of trajectory. During training, the Decision Transformer samples trajectories from offline datasets and minimizes the loss over the predicted action  $\hat{a}_t$ , conditioned on the sequence of past states, past actions, and



**Figure 2.4:** Decision Transformer architecture, where states, action, and returns are embedded separately using normal linear layers with a learned positional encoding layer added. The input propagates through these layers, a GPT architecture, and ends with a linear decoder which decodes the optimal action, the figure is adapted from [14].

RTG values in an auto-regressive manner:

$$\mathcal{L}_{dt} = \sum_{t=1}^T \mathbb{E}_{\tau} \left[ \mathcal{L} \left( a_t, \hat{a}_t \mid \hat{R}_{1:t}, s_{1:t}, a_{1:t-1} \right) \right]$$

Where:

- $\hat{R}_{1:t}$  is the sequence of RTG values from timestep 1 to  $t$ ,
- $s_{1:t}$  and  $a_{1:t-1}$  are the sequences of past states and actions up to time  $t - 1$ ,
- $\mathcal{L}$  is the loss function (e.g., cross-entropy<sup>2</sup> for discrete actions or mean squared error for continuous actions).

Once the model is trained, it starts with a target return that guides its actions. After each action, the target return is updated by subtracting it from the immediate reward received. This process repeats until the end of the episode.

On the other hand, The Trajectory Transformer models entire trajectories, including states  $s_t$ , actions  $a_t$ , rewards  $r_t$ , and RTG values  $\hat{R}_t$  as sequences. Its focus lies solely on learning the dynamics of the environment and thus can act as predictive a model that can be used for planning. The trajectory for the Trajectory Transformer is represented as:

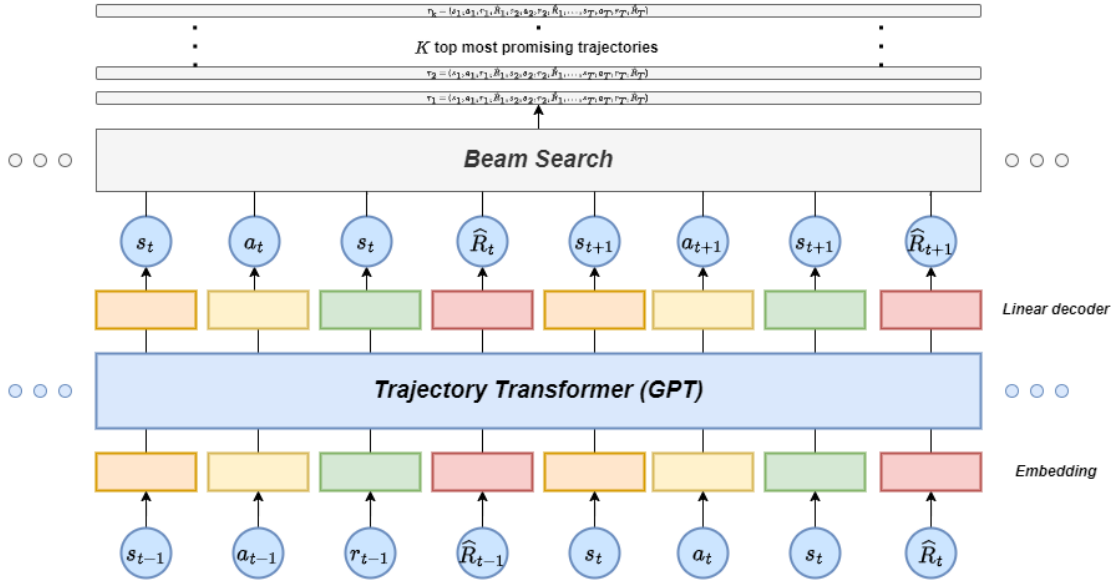
$$\tau_{tt} = (s_1, a_1, r_1, \hat{R}_1, s_2, a_2, r_2, \hat{R}_2, \dots, s_T, a_T, r_T, \hat{R}_T)$$

where each dimension in the trajectory either in the state, action, rewards, or RTG values are discretized independently. During training, the Trajectory Transformer tries to minimize the cross-entropy loss of the sequence of states, actions, rewards, and RTG values in an auto-regressive manner:

$$\mathcal{L}_{tt} = \sum_{t=1}^T \mathbb{E}_{\tau} \left[ \mathcal{L} \left( s_t, a_t, r_t, \hat{R}_t \mid s_{1:t-1}, a_{1:t-1}, r_{1:t-1}, \hat{R}_{1:t-1} \right) \right]$$

Where:

<sup>2</sup>Cross-entropy loss, is a measure used in machine learning to quantify the difference between two probability distributions i.e. the true distribution (the actual action) and the predicted distribution (the predicated action by the model)[53]



**Figure 2.5:** Architecture of the Trajectory Transformer. States, actions, rewards, and RTG are discretized and embedded. The GPT-based architecture processes this, and a linear decoder predicts the next output in the sequence. These sequences are then evaluated by the Beam Search based on RTG values, which outputs the most promising trajectories that have yielded the highest cumulative rewards. The Figure is adapted from [15].

- $s_t$ ,  $a_t$ ,  $r_t$ , and  $\hat{R}_t$  represent the state, action, reward, and RTG at time step  $t$ ,
- $s_{1:t-1}$ ,  $a_{1:t-1}$ ,  $r_{1:t-1}$ , and  $\hat{R}_{1:t-1}$  are the sequences of past states, actions, rewards, and RTG values,
- $\mathcal{L}$  is the combined cross-entropy loss function that includes appropriate sub-losses for each component <sup>3</sup>.

After training, the Trajectory Transformer is used to sample trajectory candidates and then it selects the reward-maximizing trajectory using the Beam Search algorithm. Generally, Beam Search is a heuristic search algorithm to keep track of the top  $k$  most likely candidate guided by a scoring function or likelihood. In our case, the beam search is guided by the RTG values which helps the search to keep track of the  $k$  best trajectories that are reward-maximizing.

### 2.2.2 GPT Architecture

As shown in Figures 2.4, 2.5, the GPT<sup>4</sup> serves as the core architecture for both Transformers, namely Decision Transformer and Trajectory Transformer. GPT is a unidirectional variant of the original transformer architecture introduced in [45]. It consists of multiple stacked blocks, each containing a causal self-attention layer, multi-layer perceptrons, and normalization layer [51].

Before the input data flows through these stacked blocks, it first passes through an embedding layer, which transforms the input sequence (e.g., states, actions, and RTG's) into higher dimensional vectors that enable capturing the semantic relationship between elements in the input sequence [17].

<sup>3</sup>For ease of exposition, we combine the individual losses into a single loss function  $\mathcal{L}$ . For instance, one single loss term for the state could be represented as  $\mathcal{L}_{\text{state}} = \mathcal{L}(s_t, \hat{s}_t | s_{1:t-1}, a_{1:t-1}, r_{1:t-1}, \hat{R}_{1:t-1})$ .

<sup>4</sup>Please note that the GPT architecture we mean here is GPT version 2 by [51]

It is worth noting that the GPT architecture typically uses a learned positional encoding layer next to the embedding layer to inject positional information about each element in the sequence. This approach is also used in Decision Transformers and Trajectory Transformers. However, since we chose not to use positional encoding in our work, it will not be discussed or included in our description. This decision aligns with works such as [54], [55] suggesting that the causal attention mechanism, which limits attention to one direction in the sequence, may provide the model with sufficient positional information.

After the input passes through the embedding layer, it propagates through the stacked blocks. In order to describe the layers composing one block, we consider an input matrix  $X$  (i.e. embedding output) propagating through these layers as shown in Figure 2.6. A column in matrix  $X$  represents the embedding of RTG value or single dimension<sup>5</sup> in the state or action spaces. Furthermore, the following explains the information added by each layer of the stacked blocked on this matrix until the output is reached:

1. Causal Self-Attention Layer (Single Head<sup>6</sup>): This layer enables each standardized column in  $X$  to attend to all previous columns by computing attention (inter-correlation) scores between them. It does so by first projecting the input  $X$  into three different matrices: Query  $Q = X \cdot W_q$ , Key  $K = X \cdot W_k$ , and Value  $V = X \cdot W_v$ . Second, it computes the attention scores by performing a dot product between the  $Q$  and  $K^T$  (i.e. every column of  $Q$  gets multiplied by every column of  $K$ ). The attention scores are then normalized row-wise to be a probability distribution between 0 and 1 by a softmax operation. The normalized attentions are finally multiplied with the  $V = X \cdot W_v$  to produce the final output of the self-attention layer. The operations of this layer are illustrated in Figure 2.6. Mathematically, the causal self-attention can be represented as follows:

$$\text{Causal Attention}(Q, K, V) = \text{softmax} \left( \frac{QK^T}{\sqrt{d_k}} + M \right) V \quad (2.12)$$

Where:

- $M$  is a mask matrix, where the upper triangular entries are set to  $-\infty$  to prevent attending to future positions (i.e. columns should look only to previous columns and not future ones), the  $-\infty$  added will ensure that the softmax gives a zero probability, as can be seen in Figure 2.6.
  - $d_k$  is the dimensionality of the embedding vector that is the same size for the query  $Q$  and Key  $K$ , which is used as a scaling factor to stabilize the gradient during training.
2. Normalization Layer 1: This layer normalizes the input matrix formed by the sum of the skip connection and the output of the attention layer (i.e., the weighted sum of values  $V$  after applying attention scores). This normalization step ensures to have a mean of 0 and a standard deviation which aids in training stability.
  3. Position-wise **Multi-Layer Perceptrons** (MLP)s: This layer aids in further feature extraction and introduces nonlinear transformation using activation functions like ReLU on each column (i.e. hence the name position-wise) in the output of the Normalization Layer 1.
  4. Normalization Layer 2: A second normalization step is applied after the MLP layer to ensure that the sum of the Normalization Layer 1's output and the MLP's output

<sup>5</sup>A single dimension refers to one specific element among the multiple elements that make up the state space (e.g., position, velocity, angle, or angular velocity).

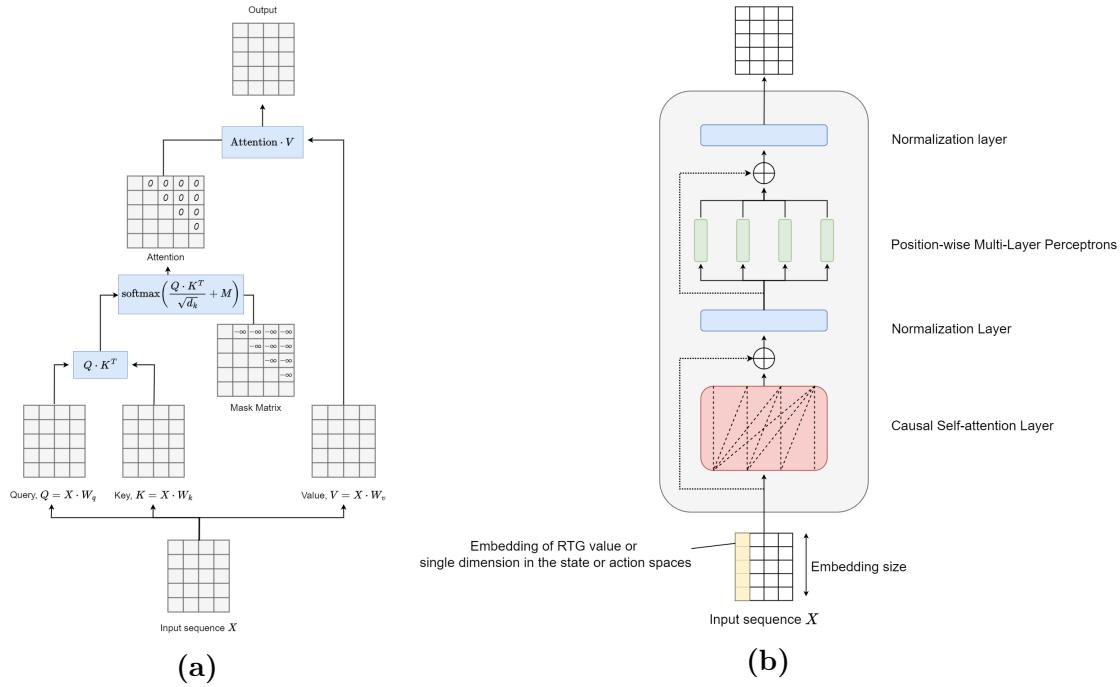
<sup>6</sup>We focus on single-head attention in this description for the sake of exposition. If multi-head attention were used, multiple sets of  $Q$ ,  $K$ , and  $V$  matrices would be created, each computing attention scores independently. The outputs would be concatenated and processed through a linear layer. The multi-head attention enables the model to capture diverse relationships by attending to different subspaces simultaneously.



maintains a mean of 0 and a standard deviation of 1. This further stabilizes the training process similar to Normalization Layer 1.

5. Residual Connections: These skip connections are added around the self-attention and Position-wise MLP as can be noted by the description of the normalization layers. These connections allow the original input to be combined with the output of each block, in order to prevent information loss and stabilize gradient flow during back-propagation.

Finally, the output of the stacked blocks is fed into a linear decoder, which generates the predicted next element in the input sequence, such as the next action in the case of the Decision Transformer.

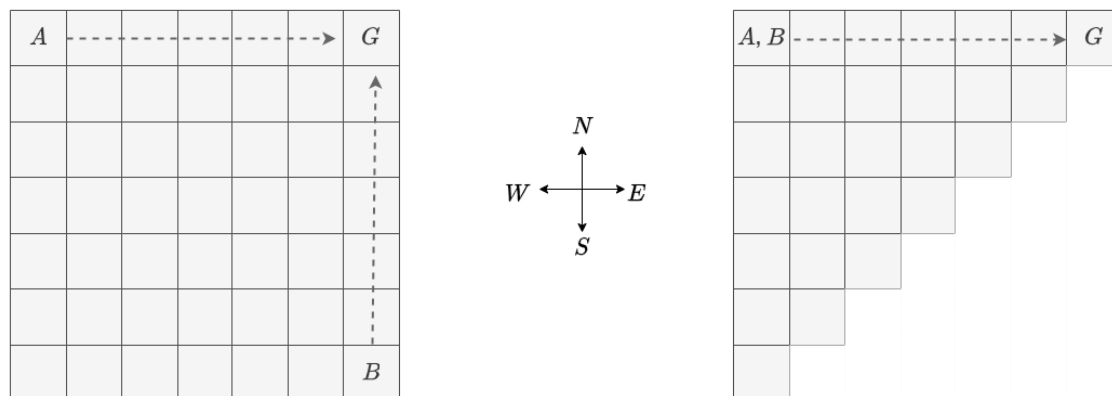


**Figure 2.6:** (a) Causal self-attention mechanism for an input sequence of size 4 with an embedding dimension of 5. The  $(\cdot)$  denotes a matrix multiplication operation (b) Single block structure of the GPT model. These figures are adapted from [17].

### 2.3 Symmetry in Reinforcement Learning for control

Symmetry in a system-theoretic sense refers to transformations such as rotations, reflections, or translations that when applied to a system, leave its dynamics unchanged. Such transformations result in an equivalent system response that requires an equivalent control strategy as before the transformation. Formally, the symmetry such as the one described in the example given in Figure 1.1, if exploited, will aid in deriving an equivalent smaller version of the original MDP by clustering equivalent state-action pairs and abstracting away any redundancy originating by symmetries. In the literature, this approach is tackled and formalized by the MDP Minimization Frameworks. One of the formalization is *bi-simulation* by [56] which exploits equivalent states in the MDP model, while the work in [28] extends the *bi-simulation* by exploiting the equivalence of joint state-action in MDP and based their minimization framework upon the notion of MDP homomorphisms. The MDP homomorphisms can be defined as a structure-preserving map between an MDP and its reduced abstracted version such that no important information is lost.

In order to describe the MDP homomorphisms framework and how the minimization process is undertaken, we use a simpler grid-world example from [28], shown in Figure 2.7. The grid world on the left side is the original MDP, where each cell corresponds to a state. An agent has four deterministic actions: North (N), South (S), East (E), and West (W), and it starts from  $A$  or  $B$  trying to reach the goal state  $G$ . One can see that there is a symmetry over the counter diagonal (i.e. bottom-left to top-right axis) in the joint state-action space. That is to say, taking action  $E$  in state  $A$  is equivalent to taking action  $N$  in state  $B$ , because both actions lead to states that are one step closer to the goal  $G$ . Additionally, if the agent started from  $A$  and reached goal  $G$ , then its policy can be reused by swap of actions and reflection of states over the counter diagonal. Thus the original MDP model can be reduced as can be seen on the right side of Figure 2.7.



**Figure 2.7:** A grid world example of MDP model minimization for symmetrical state-action pairs. On the left is the original MDP, while on the right is the reduced version over the counter diagonal axis. The diagram is adapted from [28]

In our work, we utilize the notion of MDP homomorphism as our formal tool to formalize and leverage symmetries in state-action space of finite and continuous MDPs, which we explain in detail in the following section.

### 2.3.1 MDP Homomorphism Framework

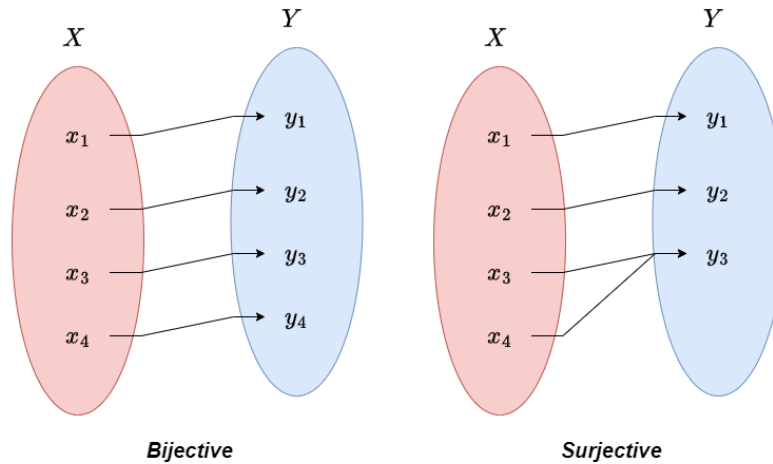
*Note: The content in this section is based on [28], [57], [58] unless otherwise cited*

#### Preliminary Notions

Before introducing the concept of finite MDP homomorphisms, we first outline preliminary notions that are needed for understanding MDP homomorphisms.

- **Group:** A *group* is an algebraic structure that consists of a set  $\mathcal{G}$  with a binary operator  $*$  :  $\mathcal{G} \times \mathcal{G} \rightarrow \mathcal{G}$  satisfying:
  - (i) **Closure:**  $h * g \in \mathcal{G}$  for all  $h, g \in \mathcal{G}$ .
  - (ii) **Identity:**  $\exists e \in \mathcal{G}$  such that  $g * e = e * g = g$  for all  $g \in \mathcal{G}$ .
  - (iii) **Inverse:**  $\forall g \in \mathcal{G}, \exists g^{-1} \in \mathcal{G}$  such that  $g * g^{-1} = g^{-1} * g = e$ .
  - (iv) **Associativity:**  $(g * h) * i = g * (h * i)$  for all  $g, h, i \in \mathcal{G}$ .
- **Group Action:** A *group action* of a group  $\mathcal{G}$  on a set  $X$  is a function  $\cdot$  :  $\mathcal{G} \times X \rightarrow X$  such that for all  $g, h \in \mathcal{G}$  and  $x \in X$ , the identity element  $e \in \mathcal{G}$  satisfies  $e \cdot x = x$ , and the action is compatible with the group operation, i.e.,  $g \cdot (h \cdot x) = (g \cdot h) \cdot x$ .
- **Equivalence Relation Induced by a Group:** Given a group  $\mathcal{G}$  acting on a set  $X$ , we define an equivalence relation  $\equiv_{\mathcal{G}}$  on  $X$  as follows: for  $x, x' \in X$ ,  $x \equiv_{\mathcal{G}} x'$  if there exists a  $h \in \mathcal{G}$  such that  $h \cdot x = x'$ .
- **Surjection (Onto Function):** A function  $f : X \rightarrow Y$  is called a *surjection* if for every  $y \in Y$ , there exists at least one  $x \in X$  such that  $f(x) = y$ . This means that  $f$  maps  $X$  onto  $Y$ , covering all elements of  $Y$ .
- **Bijection (One-to-One Correspondence):** A surjective function  $f : X \rightarrow Y$  is called a *bijection* if for every  $y \in Y$ , there exists exactly one  $x \in X$  such that  $f(x) = y$ , and each  $x \in X$  maps to a unique  $y \in Y$ . A bijection establishes a one-to-one correspondence between the sets  $X$  and  $Y$ .

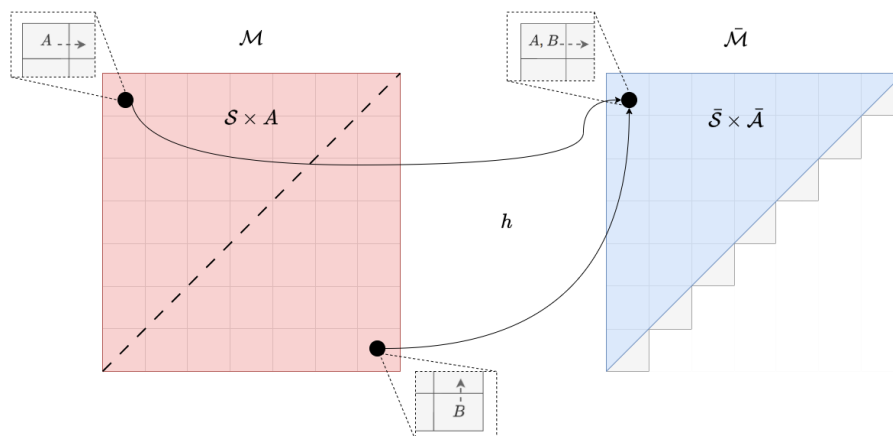
The surjection and bijection properties are illustrated in Figure 2.8



**Figure 2.8:** Illustration of bijective and surjective properties

### Finite MDP Homomorphism

In the example given in Figure 2.7, we consider state  $A$  and  $B$  equivalent in the sense that taking action  $E$  in  $A$  and  $N$  in  $B$  makes them one step closer to the goal  $G$ . This equivalence between state-action pairs allows us to define homomorphism from the original MDP  $\mathcal{M}$  to its reduced abstract version  $\bar{\mathcal{M}}$  by grouping these equivalent state-action pairs in  $\mathcal{M}$  and mapping them into single state-action pair in  $\bar{\mathcal{M}}$ , as can be seen in Figure 2.9. This grouping in  $\bar{\mathcal{M}}$  preserves both the transition structure and rewards since it exhibits similar behavior to the original  $\mathcal{M}$ .



**Figure 2.9:** Illustration a homomorphism in the grid world example, the homomorphism is a surjection from the original  $\mathcal{M}$  to its abstract version  $\bar{\mathcal{M}}$ . Equivalent state-action pairs in  $\mathcal{M}$  are grouped into a block, which is then mapped by  $h$  to a single state-action pair in  $\bar{\mathcal{M}}$ . This process preserves the transition structure and rewards between  $\bar{\mathcal{M}}$  and  $\mathcal{M}$ .

Formally, this can be now defined as:

#### Definition 2.2: Finite MDP Homomorphism [28]

An *MDP homomorphism*  $h$  from an MDP  $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$  to another MDP  $\bar{\mathcal{M}} = (\bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{\mathcal{T}}, \bar{r}, \gamma)$  is a surjection  $h$  from  $\mathcal{S} \times \mathcal{A}$  to  $\bar{\mathcal{S}} \times \bar{\mathcal{A}}$ , defined by a tuple of surjections  $\langle f, \{g_s \mid s \in \mathcal{S}\} \rangle$ , with:

$$h((s, a)) = (f(s), g_s(a)),$$

where  $f : \mathcal{S} \rightarrow \bar{\mathcal{S}}$  and  $g_s : \mathcal{A}_f \rightarrow \bar{\mathcal{A}}_{f(s)}$  for  $s \in \mathcal{S}$ , such that  $\forall s, s' \in \mathcal{S}, a \in \mathcal{A}_s$ :

$$\bar{\mathcal{T}}(f(s') \mid f(s), g_s(a)) = \sum_{s'' \in f^{-1}(\bar{s}')} \mathcal{T}(s'' \mid s, a), \quad (2.13)$$

$$\bar{r}(f(s), g_s(a)) = r(s, a). \quad (2.14)$$

Here, we call  $\bar{\mathcal{M}}$  the homomorphic image of  $\mathcal{M}$  under the homomorphism  $h$ . The double parentheses in  $h((s, a)) = (f(s), g_s(a))$  emphasize that  $h$  operates on  $(s, a)$  as a combined input, and  $\mathcal{A}_s$  represents the set of actions admissible in state  $s$ :  $\mathcal{A}_s = \{a \mid (s, a) \in \mathcal{S} \times \mathcal{A}\}$ . The conditions in (2.13) and (2.14) indicate the following:

- Condition (2.13) indicates that the transition probability of  $\bar{\mathcal{M}}$  is the sum of probabilities in  $\mathcal{M}$  transitioning from state  $s$  to all state  $s''$  that map to  $\bar{s}'$  under the function  $f$ . For an MDP with deterministic system dynamics that defines a unique next state  $s'$ , (2.13) simplifies to a commutativity condition as follows [58]:

$$\bar{\mathcal{T}}(f(s), g_s(a)) = f(\mathcal{T}(s, a)), \text{ where } \mathcal{T}(s, a) = s' \quad (2.15)$$

- Condition (2.14) ensures that any state-action pairs mapped to the same pair in  $\bar{\mathcal{M}}$  have identical expected rewards.

### Optimal Value Equivalence and Lifted Policies

The optimal value equivalence theorem states that the equivalent state-action pairs under  $h$  have the same optimal values, this is a significant result since an optimal value function found in the reduced abstract MDP can be reused as optimal value function in the original MDP.

#### Theorem 2.1: Optimal Value Equivalence [28]

Let  $\bar{\mathcal{M}} = (\bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{\mathcal{T}}, \bar{r}, \gamma)$  be the homomorphic image of the MDP  $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$  under the MDP homomorphism  $h = \langle f, \{g_s \mid s \in \mathcal{S}\} \rangle$ .

For any  $(s, a) \in \mathcal{S} \times \mathcal{A}$ ,

$$Q^*(s, a) = \bar{Q}^*(f(s), g_s(a)).$$

And for all  $s \in \mathcal{S}$ ,

$$V^*(s) = \bar{V}^*(f(s)).$$

With this property, we can optimize stochastic policies in the reduced MDP and then transfer its optimal policy  $\bar{\pi}$  back to the original MDP through a process called *lifting*.

#### Definition 2.3: Policy Lifting [28]

Let  $\bar{\pi}$  be a stochastic policy in  $\bar{\mathcal{M}}$ . Then  $\bar{\pi}$  *lifted to*  $\mathcal{M}$  is the policy  $\pi^\uparrow$  such that for any  $a \in g_s^{-1}(\bar{a})$ ,

$$\pi^\uparrow(a|s) = \frac{\bar{\pi}(\bar{a}|f(s))}{|g_s^{-1}(\bar{a})|},$$

where  $|g_s^{-1}(\bar{a})|$  denotes the cardinality (i.e., the number of elements) of the set  $g_s^{-1}(\bar{a})$ , which is the set of actions in  $\mathcal{M}$  that map to the same action  $\bar{a} \in \bar{\mathcal{A}}_{f(s)}$  under  $g_s$ . For this definition to hold, it is sufficient that

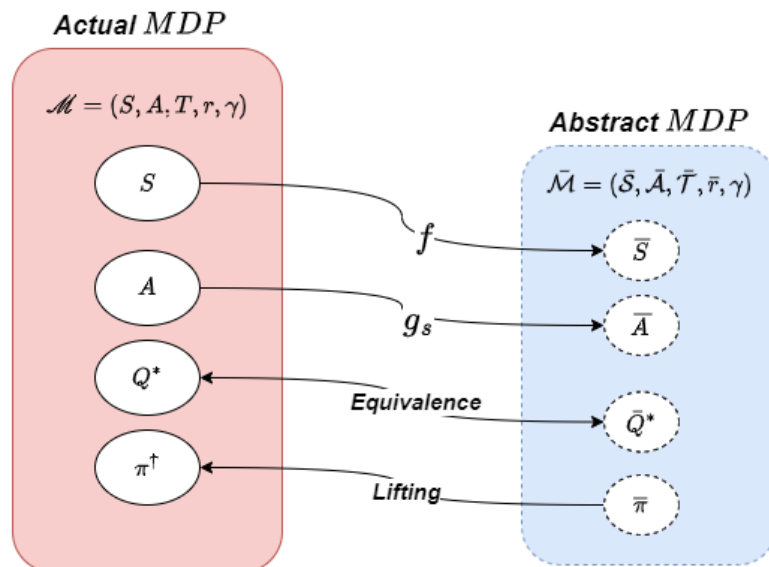
$$\sum_{a \in g_s^{-1}(\bar{a})} \pi^\uparrow(a|s) = \bar{\pi}(\bar{a}|f(s)),$$

but to ensure uniqueness, [28] distributes the probability of taking action  $\bar{a}$  given state  $f(s)$  in  $\bar{\mathcal{M}}$  evenly across all  $a \in g_s^{-1}(\bar{a})$  in  $\mathcal{M}$ .

To conclude this section, Figure 2.10 provides an illustration of the MDP homomorphism, where the relationship is demonstrated between the original MDP and its abstracted version.

### Symmetries of MDPs

MDP homomorphism generalizes the idea of minimizing MDPs by finding state-action equivalences and mapping them to a reduced abstract MDP. When these equivalences stem from inherent symmetries within MDP, the homomorphism maps symmetric state-action pairs in  $\mathcal{M}$  to the same state-action pairs in  $\bar{\mathcal{M}}$ . This means that the MDPs  $\mathcal{M}$  and  $\bar{\mathcal{M}}$  are structurally identical and  $h$  is no longer called a homomorphism, but isomorphism. Formally, this can be defined as follows:



**Figure 2.10:** Illustration of the MDP homomorphism where we display the relation between the original MDP and its reduced abstract version. The diagram is adapted from [59]

#### Definition 2.4: MDP Isomorphism [28]

An *MDP homomorphism*  $h = \langle f, \{g_s | s \in S\} \rangle$  from an MDP  $\mathcal{M} = (S, A, T, r, \gamma)$  to another MDP  $\bar{\mathcal{M}} = (\bar{S}, \bar{A}, \bar{T}, \bar{r}, \bar{\gamma})$  is called an *MDP isomorphism* if both mappings  $f$  and  $g_s$  for each  $s \in S$  are bijective.

Furthermore, if an MDP  $\mathcal{M}$  is isomorphic to itself, this self-isomorphism is known as an *automorphism* of  $\mathcal{M}$ . Formally:

#### Definition 2.5: MDP Automorphism [28]

An *MDP isomorphism* from an MDP  $\mathcal{M} = (S, A, T, r, \gamma)$  to itself is an *automorphism* of  $\mathcal{M}$ .

Automorphism can capture the inherent symmetries of an MDP, and help in eliminating any redundancies in its structure for model simplification. For instance, in the grid world example in Figure 2.7, reflecting the states along the NE-SW diagonal and swapping the actions (like North-East and South-West) is an example of a single automorphism. Using such automorphism, the original MDP can be partitioned into a quotient MDP by aggregating symmetric state-action pairs. Thus we can transfer policies learned for a quotient MDP to the original MDP by simple transformations, where this transformation preserves the transition and reward functions.

When considering all possible automorphisms of an MDP  $\mathcal{M}$ , denoted as  $\text{Aut}(\mathcal{M})$ , we can combine them in a *group*, which is formed under the composition of homomorphisms. This group is known as the *symmetry group* of  $\mathcal{M}$ .

#### Definition 2.6: Automorphism Group of an MDP $\mathcal{M}$

The set of all automorphisms of an MDP  $\mathcal{M}$  forms a group, called the automorphism group, denoted as  $\text{Aut}(\mathcal{M})$ , which is the symmetry group of  $\mathcal{M}$ .

Identifying all symmetries in MDP can be sometimes impractical, so we use the notion of a subgroup  $\mathcal{G} \subseteq \text{Aut}(\mathcal{M})$  that represents either the entire set or a relevant subset of automorphisms. This subgroup  $\mathcal{G}$  induces an equivalence relation  $\equiv_{\mathcal{G}}$  on the set of state-action pairs  $\mathcal{S} \times \mathcal{A}$ , defined as:  $(s_1, a_1) \equiv_{\mathcal{G}} (s_2, a_2)$  if and only if there exists  $h = \langle f, \{g_s \mid s \in \mathcal{S}\} \rangle$  in  $\mathcal{G}$  such that  $f(s_1) = s_2$  and  $g_{s_1}(a_1) = a_2$ .

Now that we have defined symmetries in MDPs using isomorphisms and automorphisms, and encapsulated all relevant automorphisms in a subgroup, we can formally define MDPs with symmetry as follows:

**Definition 2.7: MDP with Symmetry Group**

Let  $\mathcal{G}$  be a subgroup of  $\text{Aut}(\mathcal{M})$  such that  $\mathcal{G} \subseteq \text{Aut}(\mathcal{M})$ . Each automorphism  $h \in \mathcal{G}$  is an MDP homomorphism  $h = \langle f, \{g_s \mid s \in \mathcal{S}\} \rangle$  where  $f$  and  $g_s$  are bijective, satisfying:

$$h((s, a)) = (f(s), g_s(a)),$$

where  $f : \mathcal{S} \rightarrow \mathcal{S}$  and  $g_s : \mathcal{A}_s \rightarrow \mathcal{A}_{f(s)}$  for  $s \in \mathcal{S}$ , such that  $\forall s, s' \in \mathcal{S}, a \in \mathcal{A}_s$ :

$$\mathcal{T}(f(s') \mid f(s), g_s(a)) = \mathcal{T}(s' \mid s, a), \quad (2.16)$$

$$r(f(s), g_s(a)) = r(s, a). \quad (2.17)$$

Definition 2.7 states that an MDP with symmetries includes an automorphism  $h$  that applies a transformation to the state-action pairs  $\mathcal{S} \times \mathcal{A}$ , leaving the reward and transition functions invariant. That is to say, the invariance of the transition function, as seen in condition (2.16), means that the probability of transitioning from  $s$  to  $s'$  under action  $a$  is the same when applying the transformation  $h$ . Similarly, condition (2.17) shows that the reward function remains the same under this transformation.

Theorem 2.1 can be extended naturally to MDPs with symmetries, as the transition and reward functions are preserved under these symmetries. Consequently, the optimal action-value functions  $Q^*$  and  $\bar{Q}^*$  remain equivalent when mapped via an automorphism  $h \in \mathcal{G}$ .

However, the policy lifting described in Definition 2.3 can be more specialized and simplified, as we now have a mapping between the original MDP and the quotient MDP. The quotient MDP can be defined as the results of partitioning the original MDP's state-action space using the equivalence relation induced by a subgroup of automorphisms (the symmetry group). Thus, the lifting operation simplifies to a direct remapping using the automorphisms, which is defined as follows:

**Definition 2.8: Policy Lifting for MDPs with Automorphisms<sup>6</sup>**

Let  $\pi$  be a policy in the quotient MDP, and let  $h = \langle f, \{g_s \mid s \in \mathcal{S}\} \rangle \in \mathcal{G}$  be an automorphism of  $\mathcal{M}$ . The lifted policy  $\pi^\uparrow$  in the original MDP  $\mathcal{M}$  is defined as:

$$\pi^\uparrow(a \mid s) = \pi(g_s(a) \mid f(s)) \quad (2.18)$$

for any  $(s, a) \in \mathcal{S} \times \mathcal{A}$ , where  $f : \mathcal{S} \rightarrow \mathcal{S}$  and  $g_s : \mathcal{A}_s \rightarrow \mathcal{A}_{f(s)}$  are bijections that preserve the transition dynamics and reward functions under the automorphism  $h$ .

Applying the automorphism  $h$  once more to equation (2.18), we have:

$$\pi^\uparrow(g_s(a) \mid f(s)) = \pi(g_s(g_s(a)) \mid f(f(s))).$$

<sup>6</sup>These definitions are not explicitly stated in [28], but are derived from the information provided in that paper.

Given that  $f(f(s)) = s$  and  $g_s(g_s(a)) = a$ , substituting these properties into the equation yields:

$$\pi^\uparrow(g_s(a) | f(s)) = \pi(a | s).$$

This demonstrates that the lifted policy  $\pi^\uparrow$  can operate in the quotient  $\mathcal{M}$  by remapping under the automorphism  $h$ . Therefore, the following relation holds:

$$\pi^\uparrow(g_s(a) | f(s)) = \pi^\uparrow(a | s).$$

This relation shows that the lifted policy  $\pi^\uparrow$  is *equivariant* under the automorphism  $h$ . This equivariance relationship between the quotient MDP and the original MDP can be further simplified and formalized as follows:

### Definition 2.9: Equivariance of Stochastic and Deterministic Policies

A policy  $\pi$  is said to be *equivariant* under an automorphism  $h$  if the probability of selecting a transformed action  $g_s(a)$  given a transformed state  $f(s)$  matches the probability of selecting the original action  $a$  given the original state  $s$ . Formally, for an equivariant policy  $\pi$ ,

$$\pi(g_s(a) | f(s)) = \pi(a | s)$$

In the case of a deterministic policy, this equivariance simplifies to:

$$\pi(f(s)) = g(\pi(s))$$

where applying the transformation  $f$  to the state  $s$  results in the policy generating an action at  $f(s)$  that is equal to the transformed action  $g_s(\pi(s))$  generated by  $\pi(s)$

Furthermore, we take this opportunity to define the invariance property as well, which is a specific case of equivariance. This property is useful when indicating value functions (i.e.  $Q$  and  $V$ ), are invariant with respect to the automorphism  $h$ . Formally, the invariance property can be defined as follows:

### Definition 2.10: Invariance of Value Functions

The value functions are said to be *invariant* under an automorphism  $h$  if applying  $h$  to its inputs does not change its output. For the value functions  $V$  and  $Q$ , this means:

$$V^*(f(s)) = V^*(s) \quad \text{and} \quad Q^*(f(s), g_s(a)) = Q^*(s, a)$$



### 2.3.2 Enforcing Symmetry in DRL

A natural question that arises at this stage is how to utilize MDP homomorphism to exploit or enforce symmetry in DRL for control tasks to improve sample efficiency and achieve better generalization. Several works have addressed this question, and their methods can be categorized into three main approaches: 1) using auxiliary loss functions [32], 2) employing data augmentation [30], and 3) architecture-based methods [27], [60], [59], [29], [32]. While data augmentation and auxiliary loss functions can encourage symmetry, they do not guarantee symmetry exploitation. Moreover, data augmentation, in particular, often requires generating a large number of symmetrical samples, which is not sample-efficient.

In contrast, architecture-based methods directly encode symmetry within the neural network layers, ensuring that the symmetry present in the environment is fully exploited. The works of [27] and [29] establish a connection between MDP homomorphisms and the concept of equivariance, leveraging group-equivariant neural networks that are equivariant to the actions of a group  $\mathcal{G}$  on their input space. This means, as described in 2.9, that if the input is transformed by a certain symmetry defined by the group  $\mathcal{G}$ , the network's output will transform correspondingly.

The work of [29] focuses on reinforcement learning problems where the input is represented as images and the neural architecture is CNN, while [27] introduces an automated algorithm that constrains the symmetry within the neural network and can be applied to reinforcement learning problems with either sensory input or images.

In our work, we focus on architecture-based methods and utilize the automated algorithm proposed by [27] to encode symmetry within the neural network for reinforcement learning problems with sensory input, such as the cartpole problem in the Gymnasium Benchmark [35]. It is also noteworthy to mention that in our work the connection between MDP homomorphisms and equivariance has been drawn differently than in [27]. Namely, their approach considers a symmetry group whose elements, acting on state-action pairs, result in the homomorphism behaving as an identity map. On the other hand, our connection leverages the concept of automorphisms inspired by [28], as discussed in Section 2.3 and formally defined in Definition 2.7.

## 3 Methodology

In this chapter, we present our methodology that extends existing frameworks to address the sample efficiency and generalization problems in RL and Transformer-based RL. The methodology is divided into three main parts explained in detail below:

1. **Replication and Verification of MDP Homomorphic Networks**<sup>1</sup>: First, we replicate and verify the MDP homomorphic networks presented by [27], as detailed in Section 3.1. The work in [27] revisits the concept of MDP homomorphism in finite settings, i.e. finite state and action spaces, which is based on the framework introduced by [28]. Additionally, [27] propose a numerical algorithm to automate the construction of equivariant layers which collectively form a MDP homomorphic network that respects the state-action symmetry in the environment. We adopt their equivariant building method because it is automated, unlike other approaches that require manual network design based on the symmetry group. Their method is inherently suited for problems with finite state and action spaces (e.g., grid-based problems), but it can be extended to continuous settings with minimal modifications as we will see in later sections.
2. **Extending to Continuous MDP Homomorphisms**: The second part of our methodology, detailed in Section 3.2, involves extending the MDP homomorphism formulation from finite to continuous settings to address control problems with continuous state and action spaces, as well as both discrete and continuous symmetries. We provide a **continuous** MDP homomorphism formulation, drawing insights from [61]. Once this formulation is established, we show that with minor modifications we can extend the automated algorithm of [27] for constructing equivariant layers to handle the continuous settings.
3. **Extending to Decision Transformer**: Finally, as outlined in Section 3.3, after verifying that MDP homomorphic networks result in sample-efficient and generalizable models, we extend the concept to sequential modeling. This involves formalizing and implementing MDP homomorphism within the Decision Transformer [14], thus integrating MDP homomorphism with transformer-based decision-making models.

### 3.1 Finite MDP Homomorphic Networks

The background necessary for understanding the finite MDP homomorphism framework is provided in detail in Section 2.3.1. In this section, we introduce PPO, an actor-critic algorithm with a stochastic policy. Then, we explain how to adapt this algorithm to respect symmetries by constraining its actor and critic networks to obey the automorphism presented in the MDP. Finally, we outline the construction of a single equivariant layer, where stacking these layers results in a fully MDP homomorphic network.

#### 3.1.1 MDP Homomorphic PPO

PPO is driven by the question of how to make the largest improvement step to a policy without stepping too far, leading to a performance collapse; a common challenge in DRL. PPO is an actor-critic algorithm, which involves two main components: a critic network, which estimates the value function  $V_\phi(s)$ , and an actor network, which is based on the direction suggested by  $V_\phi(s)$  approximates the stochastic policy  $\pi_\theta(a|s)$ . There are multiple variants of PPO, we utilize PPO-Clip which has been used in [27]. PPO-Clip, as the name suggests, relies on a clipped objective function to limit updating the new policy far from

---

<sup>1</sup>**MDP Homomorphic Networks** we use this term to indicate neural networks that satisfy the equivariant property with respect to the automorphism presented in the MDP

the old one, where the "new" and "old" terms represent the policy before and after an update, respectively. The objective function is defined as:

$$L(s, a, \theta_k, \theta) = \min \left( \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}(s, a)}, g(\epsilon, A^{\pi_{\theta_k}(s, a)}) \right), \quad (3.1)$$

where  $g(\epsilon, A)$  is expressed as:

$$g(\epsilon, A) = \begin{cases} (1 + \epsilon)A, & \text{if } A \geq 0 \\ (1 - \epsilon)A, & \text{if } A < 0 \end{cases}. \quad (3.2)$$

This formulation constrains policy updates by clipping the probability ratio  $\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$  within a range defined by  $\epsilon$ . When the advantage  $A^{\pi_{\theta_k}(s, a)}$  is positive, the policy update increases the probability of the action, but only up to the limit  $(1 + \epsilon)$ . Conversely, when the advantage is negative, the update reduces the probability of the action, bounded by  $(1 - \epsilon)$ . Furthermore, the algorithmic steps for PPO-Clip are given as pseudo-code in Algorithm 1.

---

**Algorithm 1:** PPO-Clip [62]

---

[1] : **input:** initial policy parameters  $\theta_0$ , initial value function parameters  $\phi_0$

[2] : **for**  $k = 0, 1, 2, \dots$  **do**

[3] :     Collect a set of trajectories  $\mathcal{D}_k = \{\tau_i\}$  by running policy  $\pi_{\theta_k}(\cdot)$  in the environment

[4] :     Compute rewards-to-go  $\hat{R}_t$

[5] :     Estimate advantages  $\hat{A}_t$  using a chosen method based on the current value function  $V_{\phi_k}$

the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T \min \left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}(s_t, a_t)}, g(\epsilon, A^{\pi_{\theta_k}(s_t, a_t)}) \right),$$

typically using stochastic gradient ascent with the Adam optimizer

[6] :     Update the value function by minimizing the mean-squared error:

$$\phi_{k+1} = \arg \min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^T (V_\phi(s_t) - \hat{R}_t)^2,$$

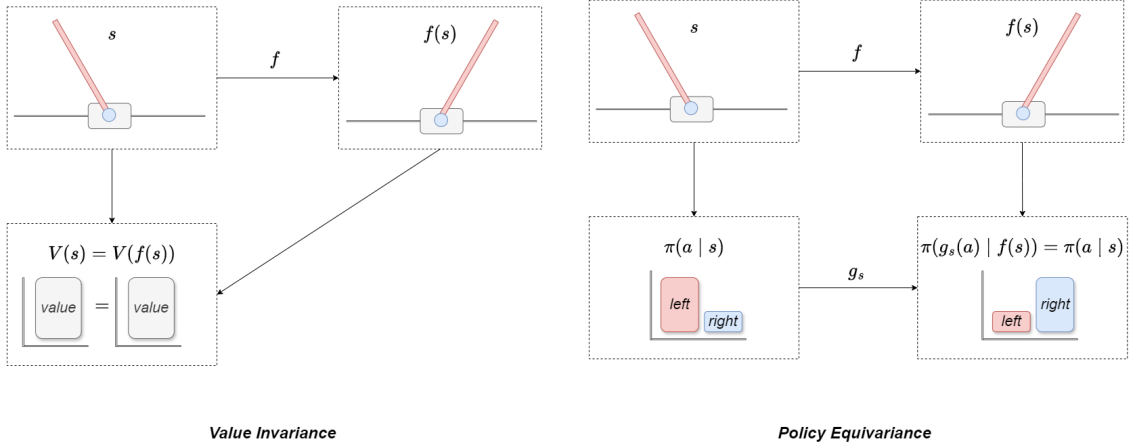
using gradient descent

**end**

---

In order to obtain a PPO agent that respects the symmetry group represented by a set of automorphisms  $h$  in group  $\mathcal{G} \subseteq \text{Aut}(\mathcal{M})$ , we need to design the policy and value networks in a way that they obey the MDP automorphism. That is to say, the policy network  $\pi_\theta$  must satisfy the equivariant property with respect to the automorphisms in the group  $\mathcal{G}$ , as stated in Definition 2.9. Similarly, the value network  $V_\phi$  must satisfy the invariant property under  $h$ , as stated in Definition 2.10.

Going back to the pole balancing example given in Figure 1.1, which has a continuous state space and finite state space with two actions i.e. moving the cart to the left and right. Let  $h^e$  be the identity automorphism on its state-action space  $\mathcal{S} \times \mathcal{A}$ , and let  $h^1$  be an automorphism on  $\mathcal{M}$  defined as the vertical reflection  $f$  on the state space and the action swap  $g_s$  on the action space (i.e. right becomes left and the other way around). Thus, the set of all automorphisms is given by two elements  $\text{Aut}(\mathcal{M}) = \{h^e, h^1\}$ , and



**Figure 3.1:** Illustration of the equivariance and invariance properties of the policy and value functions under the automorphisms  $h^e$  and  $h^1$  in the pole balancing problem. The diagram is adapted from [27]

together with the composition operator, they form the symmetry group  $\mathcal{G}$  of the MDP underlying the pole balancing problem. Then the policy and value functions should satisfy the equivariance and invariance properties as illustrated in Figure 3.1.

### 3.1.2 Constructing Equivariant Layers

As proposed by [27], we outline the method for constructing equivariant neural network layers. A neural network composed entirely of such equivariant layers and point-wise nonlinearities (e.g. ReLU) remains equivariant as a whole [63]. Therefore, the main goal is to construct layers that respect input and output transformations defined by any  $h = \langle f, \{g_s | s \in S\} \rangle \in \mathcal{G}$ , where  $f$  represents the input transformation and  $g_s$  the state-dependent output transformation. By stacking these layers, we achieve a network that exhibits equivariance under these transformations, which in turn align with MDP homomorphism properties.

Starting from a basic linear layer in element-wise form:

$$y_j = \sum_{i=1}^{D_{in}} w_{ji}x_i + b_j, \quad \text{for } j = 1, 2, \dots, D_{out}, \quad (3.3)$$

where  $x \in \mathbb{R}^{D_{in}}$  is the input vector,  $W \in \mathbb{R}^{D_{out} \times D_{in}}$  is the weight matrix,  $b \in \mathbb{R}^{D_{out}}$  is the bias vector, and  $y \in \mathbb{R}^{D_{out}}$  is the output vector.

Equation 3.3 can be represented in matrix form as:

$$y = W \cdot z,$$

where the weight matrix  $W$  is augmented with the bias  $b$ :  $W = [W \ b] \in \mathbb{R}^{D_{out} \times (D_{in}+1)}$ , and the input vector  $x$  is extended with an additional component set to 1:  $z = [x \ 1]^T \in \mathbb{R}^{(D_{in}+1)}$ .

As proposed by [27], if the transformations for any  $h = \langle f, \{g_s | s \in S\} \rangle \in \mathcal{G}$  are linear and can be represented in matrix form as:

$$f(s) = F \cdot s, \quad g_s(a) = G_s \cdot a,$$

Then, in order to construct equivariant layers that satisfy the constraints imposed by these linear input and output transformations, the weight matrix  $W$  must satisfy the following

condition:

$$G_s W z = W F z.$$

Since this condition holds for all  $z$ , it simplifies to:  $G_s W = W F$ . The space of equivariant weights  $\mathcal{W}$  can thus be defined as:

$$\mathcal{W} = \{W \in \mathcal{W}_{\text{total}} \mid G_s W = W F, \text{ for all } h = \langle f, \{g_s \mid s \in S\} \rangle \in \mathcal{G}\},$$

where  $\mathcal{W}_{\text{total}}$  is the space of augmented weights.

To construct  $W$  that meets the above constraints, [27] uses a symmetrization approach and introduces the symmetrizer  $S(W)$  that ensures that  $W$  lies in the space of matrices satisfying the equivariance property:

$$S(W) = \frac{1}{|\mathcal{G}|} \sum_{h \in \mathcal{G}} G_s^{-1} W F, \quad (3.4)$$

where  $|\mathcal{G}|$  is the number of elements in the symmetry group  $\mathcal{G}$ . This symmetrizer will project any initial random weight matrix  $W$  into the subspace that meets the equivariance constraint.

Since  $\mathcal{W}$  is a linear subspace, any weight matrix  $W \in \mathcal{W}$  can be expressed as a linear combination of basis matrices  $\{V_i\}_{i=1}^r$ , where  $r$  is the rank of the subspace. Thus,  $W$  can be parameterized as  $W = \sum_{i=1}^r c_i V_i$ , with  $c_i$  being learnable coefficients. Therefore, to identify the basis  $\{V_i\}$ , [27] uses the Gram-Schmidt orthogonalization approach that is applied through singular value decomposition (SVD). Algorithm 2 outlines this procedure, demonstrating how sampled weights are symmetrized and orthogonalized to construct an equivariant layer.

---

**Algorithm 2:** Equivariant Layer Construction

---

- [1] : **input:** Number of samples  $N$ , dimension of total weight space  $\dim(\mathcal{W}_{\text{total}})$   
 [2] : **for**  $i = 1, \dots, N$  **do**  
     | Sample weight matrix  $W_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$   
     | Symmetrize  $W_i$  as  $W_i \leftarrow S(W_i)$   
**end**  
 [3] : Stack vectorized samples  $\mathbf{W} = [\text{vec}(W_1), \text{vec}(W_2), \dots, \text{vec}(W_N)]$   
 [4] : Perform SVD on  $\mathbf{W}$  to obtain  $\tilde{\mathbf{W}} = \mathbf{U}\Sigma\mathbf{V}^\top$   
 [5] : Keep the first  $r = \text{rank}(\tilde{\mathbf{W}})$  right-singular vectors (columns of  $\mathbf{V}$ )  
 [6] : Unvectorize the right-singular vectors to form the basis  $\{V_i\}_{i=1}^r$   
 [7] : **output:**  $\{V_i\}_{i=1}^r$  and  $r$ ;
- 

Please note that the basis matrices  $\{V_i\}$  are fixed once they are computed at the start of the training process. Only the coefficients  $c_i$  are updated during optimization, which allows the network to adapt the weight matrix by adjusting the linear combination of these fixed bases. Additionally, when we need to build value networks  $V_\phi$  that are invariant with regard to the transformation on the input space, then the transformation on the output represented in the method above as  $G_s$  needs to be the identity.

## 3.2 Continuous MDP Homomorphic Networks

In this section, we extend the formulation of finite MDP to continuous MDP based upon insights from [61]. The main reason for such an extension is that most control problems involve continuous state and action spaces, with some exhibiting continuous symmetries. Subsequently, we will introduce the TD3 algorithm briefly [34], an actor-critic algorithm well-suited for continuous control. We then conclude this section by detailing how minor modifications are made to the construction of the equivariant layer method introduced in section 3.1.2 and to the MDP homomorphic networks to function in continuous settings.

### 3.2.1 Continuous MDP Homomorphism

In continuous MDPs, we assume that the state and action spaces are not countable but instead form continuous ranges. Additionally, the state and action spaces are assumed to be compact, measurable subsets of  $\mathbb{R}^n$  [61]. **Compactness** ensures the existence of optimal solutions and the boundedness of the reward function, while **measurability** allows for the proper application of probability and integration when defining the transition function, as we will see in the following definition.

Moreover, symmetries in continuous MDPs can be divided into two types: **discrete symmetry** and **continuous symmetry**. Discrete symmetry involves a finite number of equivalent (symmetric) state-action pairs present in the MDP. An example of this is the pole-balancing problem shown in Figure 1.1. On the other hand, continuous symmetry involves infinite equivalent (symmetric) state-action pairs.

An example of continuous symmetry can be found in the Gym Reacher environment [35], shown in Figure 3.2, where a two-joint robotic arm must move its end-effector (fingertip) close to a target. Here, we assume that the target is positioned on a circle with a fixed radius and that the initial position of the end-effector is on this circle. If the target and the first link are rotated by the same angle  $\theta$  while the second link remains fixed, the fingertip traces a circular path. The sequence of actions leading the fingertip to the target under these conditions would be equivalent and exhibit continuous rotational symmetry in the state-action pairs.

To capture both types of symmetries in a continuous MDP homomorphism, we must account not only for discrete symmetries where a finite number of symmetric state-action pairs in the original MDP map to a single point in the reduced abstract MDP but also for continuous symmetries where an infinite number of symmetric state-action pairs map to a single point. This can be achieved by considering continuous mappings between *two closed topological sets*, where an infinite number of points in the pre-image can form closed sets [61].

#### Definition 3.1: Continuous MDP Homomorphism

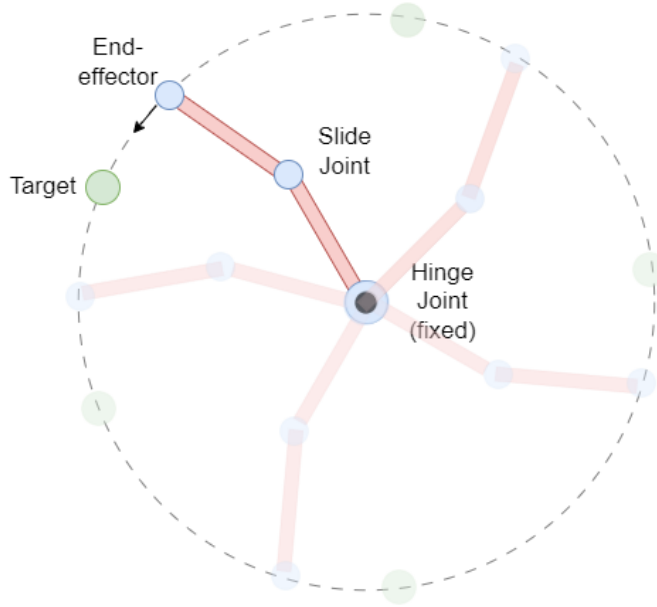
An *continuous MDP homomorphism*  $h$  from a continuous MDP  $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$  to another continuous MDP  $\bar{\mathcal{M}} = (\bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{\mathcal{T}}, \bar{r}, \gamma)$  is a continuous surjection  $h$  from  $\mathcal{S} \times \mathcal{A}$  to  $\bar{\mathcal{S}} \times \bar{\mathcal{A}}$ , defined by a tuple of continuous surjections  $\langle f, \{g_s \mid s \in \mathcal{S}\} \rangle$ , with:

$$h((s, a)) = (f(s), g_s(a)),$$

where  $f : \mathcal{S} \rightarrow \bar{\mathcal{S}}$  and  $g_s : \mathcal{A}_s \rightarrow \bar{\mathcal{A}}_{f(s)}$  for  $s \in \mathcal{S}$ , such that  $\forall s, s' \in \mathcal{S}, a \in \mathcal{A}_s$ :

$$\bar{\mathcal{T}}(f(s') \mid f(s), g_s(a)) = \int_{s'' \in f^{-1}(\bar{s}')} \mathcal{T}(s'' \mid s, a) ds'', \quad (3.5)$$

$$\bar{r}(f(s), g_s(a)) = r(s, a). \quad (3.6)$$



**Figure 3.2:** Illustration of continuous rotational symmetry in the Gym Reacher environment, where equivalent action sequences can result from rotating the target and the first link by the same angle.

Here, we call  $\bar{\mathcal{M}}$  the homomorphic image of  $\mathcal{M}$  under the homomorphism  $h$ . The  $\bar{s}'$  is the image of  $s'$  under  $f$  in  $\bar{\mathcal{M}}$ . The continuity condition in the surjections ensures that  $f^{-1}(\bar{s}')$  is a well-defined, measurable set, thus enabling us to evaluate the integral of all transitions [61]. Furthermore, to account for discrete symmetries characterized by finite equivalent state-action pairs, we can replace the integral by a summation in Condition (3.5).

Before introducing how a policy of the continuous abstract  $\bar{\mathcal{M}}$  can be lifted to the actual  $\mathcal{M}$ , we need to extend Theorem 2.1 on optimal value equivalence to the continuous case, which is successfully proved with insights from the proof of finite case [28]. The proof is listed in Appendix A. Now we can define the policy lifting as follows:

### Definition 3.2: Continuous Policy Lifting

Let  $\bar{\pi}$  be a stochastic policy in the abstract continuous MDP  $\bar{\mathcal{M}}$ . Then  $\bar{\pi}$  lifted to  $\mathcal{M}$  is the policy  $\pi^\uparrow$  is defined as:

$$\pi^\uparrow(a|s) = \frac{\bar{\pi}(\bar{a}|f(s))}{\int_{a' \in g_s^{-1}(\bar{a})} da' \delta(g_s(a') - g_s(a))},$$

where  $g_s^{-1}(\bar{a})$  represents the set of actions  $a' \in A$  such that  $g_s(a') = \bar{a}$ , and  $\delta(g_s(a') - g_s(a))$  is the Dirac delta function, which ensures that the integral only considers actions that map to the same image as  $a$  under  $g_s$ .

Note that the integral in the denominator evaluates to the total count of the values of  $a'$ . Thus, the Dirac delta function in the integrand acts as a filter that only contributes when  $g_s(a') = g_s(a)$ .

Finally, the results on MDPs with symmetries and equivariance discussed in Section 2.3.1 are applicable and can be naturally extended to the continuous case without modification.

### 3.2.2 MDP Homomorphic TD3

TD3 is an advanced algorithm in RL particularly designed for environments with continuous state and action spaces. It is an improvement on the Deep Deterministic Policy Gradient (DDPG) through several key modifications. First, TD3 concurrently trains two Q-functions,  $Q_{\phi_1}$  and  $Q_{\phi_2}$ , hence the term "twin", and uses the minimum of the two outputs to calculate the target as can be seen in (3.7). This mitigates the issue of overestimating Q-values, which is a common problem in DDPG.

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s')) \quad (3.7)$$

Next, TD3 incorporates target policy smoothing to avoid sharp spikes in the learned Q-function. The target action  $a'(s')$  is computed using the target policy  $\pi_{\theta_{\text{targ}}}$  with added noise that is clipped to keep the action within a valid range. This approach smooths out the Q-function over similar actions and prevents the policy from exploiting any peak in the Q-values. The smoothing operation is shown in (3.8) which is performed on the parameters of the networks.

$$\begin{aligned} \phi_{\text{targ},i} &\leftarrow \rho\phi_{\text{targ},i} + (1 - \rho)\phi_i, & \text{for } i = 1, 2 \\ \theta_{\text{targ}} &\leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta \end{aligned} \quad (3.8)$$

Furthermore, the policy is updated by maximizing  $Q_{\phi_1}$  to find the optimal action. However, to maintain stability and reduce volatility, the policy is updated less frequently, typically once for every two updates of the Q-functions. This delayed policy update ensures that the policy does not shift too rapidly. Furthermore, in the interest of space, the algorithmic steps for TD3 are given as pseudo-code in Appendix C.

To design a TD3 agent that respects a symmetry group  $\mathcal{G} \subseteq \text{Aut}(\mathcal{M})$ , similar to the approach used with PPO, the Q-functions and policy network must obey to the MDP homomorphism. Specifically, the Q-functions  $Q_{\phi_1}$  and  $Q_{\phi_2}$  must satisfy the invariant property under automorphisms  $h = \langle f, \{g_s \mid s \in S\} \rangle \in \mathcal{G}$ , as outlined in Theorem 2.1. This ensures that:

$$Q_{\phi_i}(f(s), g_s(a)) = Q_{\phi_i}(s, a), \quad i = 1, 2.$$

Similarly, the policy network  $\pi_{\theta}$  must be equivariant with respect to the automorphisms in  $\mathcal{G}$ , following Definition 2.8. Given that the policy in TD3 is deterministic<sup>2</sup>, it must satisfy:

$$\pi_{\theta}(f(s)) = g_s(\pi_{\theta}(s)).$$

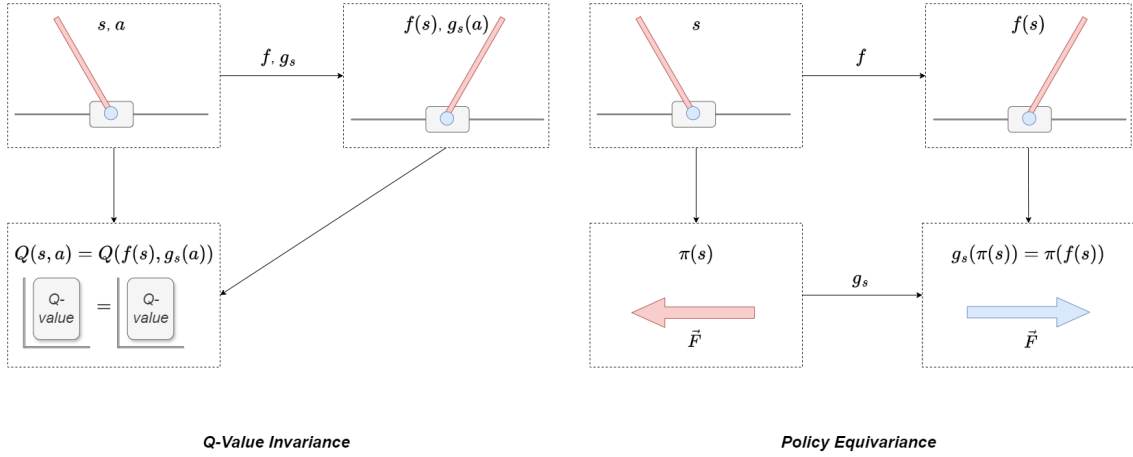
As an example, we utilize the pole-balancing problem, as previously shown in Figure 3.1. This time, we assume a continuous state and action space, where the action space requires a continuous magnitude of force and a direction applied to the cart. Since TD3 produces a deterministic policy, it is well-suited for handling such problems that require continuous action values. The policy and Q-value function then must satisfy the equivariance and invariance properties, as depicted in Figure 3.3.

### 3.2.3 Extending Homomorphic Networks to Continuous Settings

To extend the use of homomorphic networks to continuous settings, i.e., continuous state and action spaces, and to exploit both discrete and continuous symmetries, we utilize the symmetrization-based approach introduced by [64] for constructing invariant and equivariant approximations using neural networks. The method for constructing equivariant layers discussed in Section 3.1.2 is actually based on this approach, but it is not yet applicable

<sup>2</sup>Most references use the symbol  $\mu$  to indicate a deterministic policy, but to avoid confusion in this work, we use  $\pi$  to denote deterministic policy  $\pi(s)$  and  $\pi(a|s)$  for stochastic policies.





**Figure 3.3:** Illustration of the equivariance and invariance properties of the policy and Q-value function under automorphisms in the pole-balancing problem, where the state and action spaces are continuous.

for continuous settings and for both continuous and discrete symmetries. The author in [64] provided two key propositions for constructing invariant and equivariant maps that are universal and applicable to both kinds of symmetries. For brevity, detailed descriptions of these propositions are provided in Appendix B. Here, we summarize the modifications required to enable MDP homomorphic networks to operate in continuous settings.

- **Smooth Activation Functions:** The first modification involves using smooth continuous activation functions as point-wise nonlinearities, such as the "tanh" function. Through trial and error with different activation functions, such as ReLU and its variants, it is observed that these functions can break the equivariance properties when the automorphisms are continuous.
- **Integration for Continuous Symmetries:** The second modification addresses continuous symmetries, where the automorphism group is infinite. Logically, we could integrate over the symmetry group  $\mathcal{G}$ , and the symmetrization formula given in (3.4) would then be expressed using an integral over the group:

$$S(W) = \frac{1}{|\mathcal{G}|} \int_{\mathcal{G}} G_s^{-1} W F d\mu(h),$$

where  $d\mu(h)$  is the Haar measure on  $\mathcal{G}$  to ensure proper integration over the group, and  $|\mathcal{G}|$  represents the measure of the entire group for normalization. While this approach accounts for continuous symmetries by integrating over all possible transformations in  $\mathcal{G}$ , it is impractical and computationally expensive. Therefore, sampling from the symmetry group, such as using equidistant angle sampling, is necessary to achieve a practical solution. This introduces approximation errors, which require an equivariance error metric to quantify such deviations. The definition of this error metric will be introduced in the sequel.

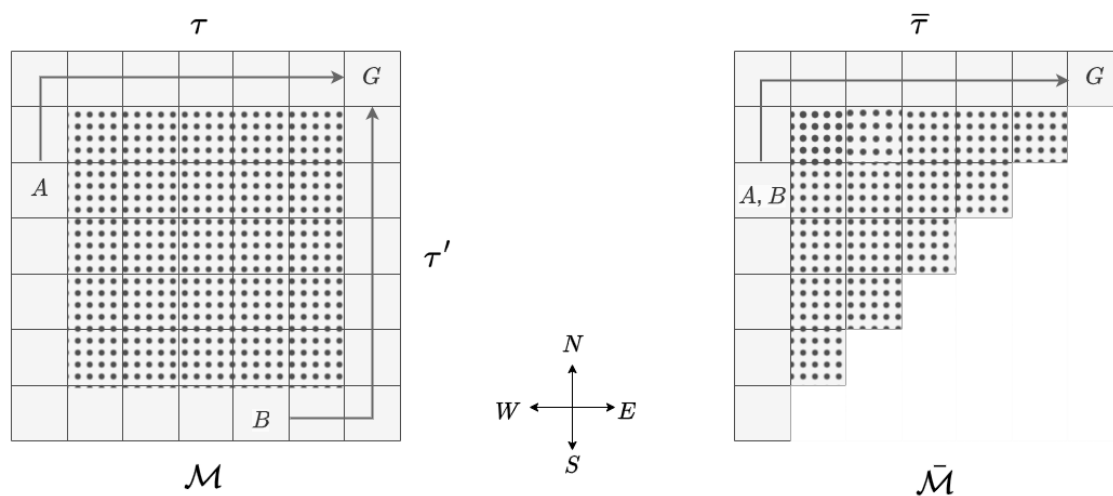
### 3.3 MDP Homomorphic Decision Transformer

In this section, we formally define MDP homomorphism within the context of the Decision Transformer. We then explain how to adapt the Decision Transformer to be homomorphic by ensuring it adheres to the MDP homomorphism. In this work, we focus on Decision Transformer, as any success can be naturally extended to trajectory transformers due to the minimal differences between the two architectures, as outlined in Section 2.2.

#### 3.3.1 MDP Homomorphism in Decision Transformer

Since sequential models, such as the Decision Transformer, operate on sequences of past states, actions, and RTG values to predict the next action, applying MDP homomorphism requires preserving sequence-level properties. Thus, instead of focusing solely on individual state-action pairs, as in the finite or continuous MDP homomorphism cases, the homomorphism must be defined over entire sequences. This involves identifying equivalent trajectories in the original MDP  $\mathcal{M}$  and mapping them to a single trajectory in the reduced abstract MDP  $\bar{\mathcal{M}}$ . This mapping to  $\bar{\mathcal{M}}$  preserves both the transition and rewards of the original MDP  $\mathcal{M}$ .

An illustrative example is provided in Figure 3.4 for a grid-world problem, building upon the example in Figure 2.7. In this case, the dotted cells represent barriers, thus the agent can not reach those cells. Two equivalent trajectories,  $\tau$  and  $\tau'$ , are shown, which are mapped under the homomorphism to a single trajectory  $\bar{\tau}$  in the reduced abstract MDP  $\bar{\mathcal{M}}$ .



**Figure 3.4:** Illustrative example of trajectory mapping under homomorphism in a grid-world problem, where the dotted cells are barriers.

Formally, a trajectory in an MDP, as previously defined in equation (3.9), is represented as a sequence of state-action pairs of length  $H$ :

$$\tau = \{(s_0, a_0), (s_1, a_1), \dots, (s_H, a_H)\},$$

where  $s \in \mathcal{S}$  and  $a \in \mathcal{A}_s$  for  $t = 0, 1, \dots, H$ .

We can extend the continuous and finite MDP homomorphism framework introduced earlier by defining Trajectory Equivalence under the MDP homomorphism as follows:

**Definition 3.3: Trajectory Equivalence under MDP Homomorphism**

Let  $h = (f, \{g_s \mid s \in \mathcal{S}\})$  be a homomorphism mapping from  $\mathcal{M}$  to its homomorphic image  $\bar{\mathcal{M}}$ , where  $f : \mathcal{S} \rightarrow \bar{\mathcal{S}}$  and  $g_s : \mathcal{A}_s \rightarrow \bar{\mathcal{A}}_{f(s)}$ . Two trajectories  $\tau$  and  $\tau'$  are said to be equivalent under  $h$  if their corresponding state-action pairs map to the same trajectory in  $\bar{\mathcal{M}}$ :

$$h(\tau) = h(\tau') \implies \bar{\tau} = \bar{\tau}'.$$

The mapped trajectory  $\bar{\tau}$  in  $\bar{\mathcal{M}}$  is defined as:

$$\bar{\tau} = \{(f(s_0), g_{s_0}(a_0)), (f(s_1), g_{s_1}(a_1)), \dots, (f(s_H), g_{s_H}(a_H))\}.$$

The result in Definition 3.3 applies to all homomorphisms, including special cases such as automorphisms. Now to ensure that the Decision Transformer respects the symmetry group represented by a set of automorphisms  $h$  in a group  $\mathcal{G} \subseteq \text{Aut}(\mathcal{M})$ , it is necessary that Decision Transformer should produce policy  $\pi$  that satisfy the following properties:

$$\pi(h[\tau]) = h[\pi(\tau)], \quad \forall h \in \mathcal{G}, \quad (3.9)$$

where  $h[\cdot]$  represents the corresponding transformation in the action and state spaces induced by  $h$ . It is worth mentioning that the Decision Transformer augments its trajectory with RTG values, as explained in Section 2.2. These values, as the case with rewards, are also invariant under homomorphism according to Definitions 2.2 and 3.2.

**3.3.2 Constructing MDP Homomorphic Decision Transformer**

In this section, we aim to construct an MDP Homomorphic Decision Transformer by utilizing the equivariant layers introduced in Section 3.1.2. Our goal, as defined in 3.9, is to ensure that if two symmetric trajectories are fed through the Decision Transformer, their predicted action sequences should be equal, i.e., the equivariance property holds. This property is preserved when using equivariant layers even without any training, as explained in Section 3.1.2, because the basis of the equivariant subspace  $\{V_i\}$  is fixed, and the coefficients  $c_i$  do not violate equivariance. Any linear combination of symmetric basis matrices produces a symmetric weight matrix, therefore we can utilize an equivariance error metric as a sanity check to test whether the equivariance holds. This metric will be introduced later in the sequel.

Furthermore, we follow a mechanistic approach inspired by [65], starting from the embedding layer, incrementally adding Decision Transformer components (explained in detail in Section 2.2.2), and analyzing their impact on the equivariance property. This approach is possible due to the use of residual connections in the Transformer architecture, which forms a residual stream acting as a communication channel. The residual stream itself does not perform any processing; instead, all layers communicate through it, where each layer reads information from the residual stream, applies a transformation, and writes the result back into the residual stream [65].

It is worth reiterating that we do not use positional encoding in our work for two main reasons. First, because works such as [54] and [55] suggest that the causal attention mechanism, which limits attention to one direction in the sequence, provides the model with sufficient positional information, so there is actually no need for positional encoding. Second, positional encoding breaks the equivariance property.

After performing this mechanistic approach, we found that the most important components that need to satisfy the equivariance property are the embedding layers, linear decoder layers, and the attention mechanism. Point-wise MLPs do not break the equivariance properties since they are performed independently at each position in the sequence, meaning

there is no communication occurring between different positions in the sequence. Additionally, normalization layers and dropout layers do not break the equivariance property.

We illustrate the mechanistic approach in Figure 3.5, which is composed of three main steps. In each step, a new component is added to the residual stream, and we test whether the equivariance property still holds. The equations on the right are given in tensor representation for clarity. The weights symbolized as  $W$  in embedding layers and the attention mechanism, are replaced by their equivariant versions for MDP Homomorphic Decision Transformer.

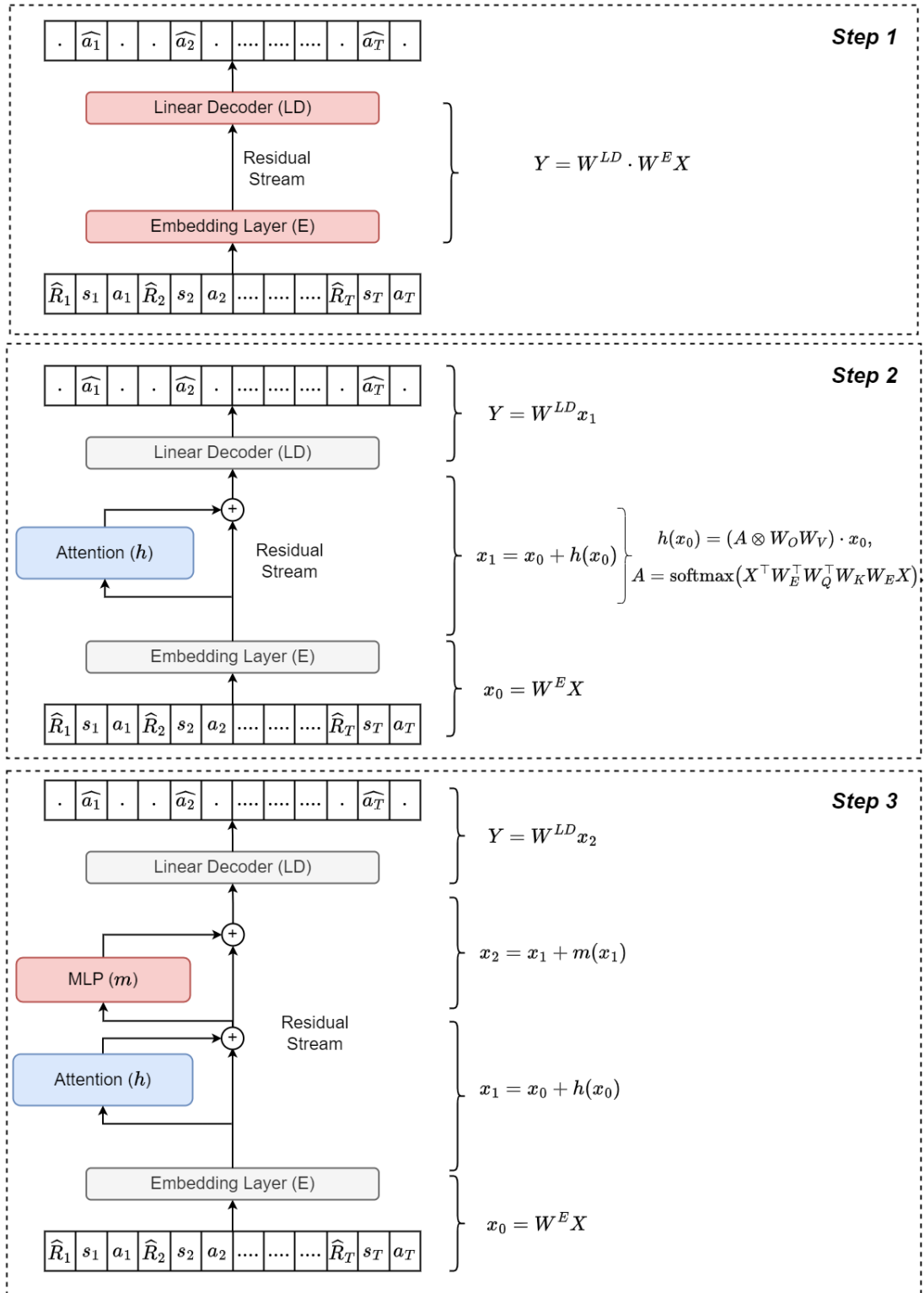
We conclude this section with high-level algorithmic steps for the Decision Transformer given in Algorithm 3.

---

**Algorithm 3:** Decision Transformer
 

---

- [1] : **input:** Decision Transformer  $\theta_0$ , dataset of trajectories  $\mathcal{D} = \{\tau_i\}$   
 [2] : **for**  $k = 0, 1, 2, \dots$  **do**  
 [3] :   Sample a batch of trajectories  $\mathcal{B}_k = \{\tau_i\}$  from  $\mathcal{D}$ , where each trajectory  $\tau_i$  consists of states  $s_t$ , actions  $a_t$ , and return-to-go (RTG) values  $\text{RTG}_t$   
 [4] :   Embed the input sequence  $\mathcal{I} = \{(s_t, a_t, \text{RTG}_t)\}_{t=1}^T$  using the embedding layer to produce token embeddings  
 [5] :   Pass embeddings through the transformer block and then use linear decoder layer to predict the next actions  $\hat{a}_t$   
 [6] :   Compute the loss function:
- $$\mathcal{L}(\theta) = \frac{1}{|\mathcal{B}_k|T} \sum_{\tau \in \mathcal{B}_k} \sum_{t=1}^T \|\hat{a}_t - a_t\|^2,$$
- where  $\hat{a}_t$  is the predicted action and  $a_t$  is the ground-truth action from the dataset  
 [7] :   Update transformer parameters  $\theta$  by minimizing  $\mathcal{L}(\theta)$  using stochastic gradient descent with the Adam optimizer  
**end**
-



**Figure 3.5:** Illustration of the mechanistic approach, showing the incremental addition of components to the residual stream. The equations on the right represent the transformations applied in each step starting from input sequence  $X$  until output  $Y$ .

## 4 Experiments

Following the methodology outlined in Chapter 3, which comprises three main parts, we now present the experimental results for each part. The primary objective of these experiments is to demonstrate that exploiting the symmetries present in the environments improves sample efficiency and enables the learning of highly generalizable policies. To reiterate, generalization here refers to an agent’s ability to transfer learned behaviors from one symmetric sample to others. This chapter is organized as follows: we first introduce the simulation environments used in the experiments, then describe the implementation details and fine-tuning procedure. Finally, we present and analyze the results for the different algorithms.

### 4.1 Environments

We evaluated our methodology on three RL control tasks that exhibit discrete symmetry: CartPole and Inverted Pendulum, all part of the Gymnasium benchmark suite [35]. The CartPole environment is used to evaluate the MDP homomorphic PPO algorithm, while the Inverted Pendulum, with its continuous state and action spaces, is utilized to evaluate the MDP homomorphic TD3 and MDP homomorphic Decision Transformer.

#### CartPole and Inverted Pendulum Environments

The CartPole and Inverted Pendulum are variants of the classic pole-balancing problem defined in [5]. They differ in two main aspects:

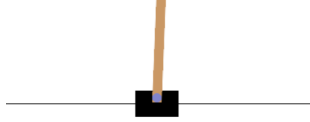
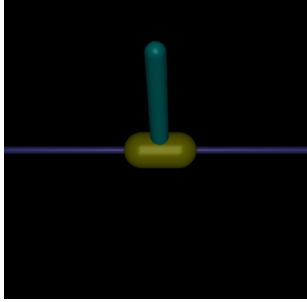
- **Action Space:** The CartPole environment uses a discrete action space, where the agent exerts a fixed force either left or right. In contrast, the Inverted Pendulum has a continuous action space, requiring the agent to output precise force values to balance the pole.
- **Physics Model:** CartPole employs a basic physics model, which uses simplified assumptions such as linear dynamics and uniform conditions without advanced interactions like friction or contact forces. In comparison, the Inverted Pendulum relies on an advanced simulator (e.g., MuJoCo), which incorporates nonlinear dynamics, complex interactions, and features like variable gravity.

A detailed comparison of these environments is provided in Table 4.1.

#### 4.1.1 Symmetry in CartPole and Inverted Pendulum

Both the CartPole and Inverted Pendulum environments exhibit reflection symmetry over the vertical axis. The symmetry group for these environments, denoted as  $\mathcal{G}$ , consists of two elements: the identity automorphism  $h^e$  on the state-action space  $\mathcal{S} \times \mathcal{A}$ , and an automorphism  $h^1$  representing the vertical reflection  $f$  on the state space  $\mathcal{S}$ . In the CartPole environment, this reflection results in a swap in the action space  $\mathcal{A}$ , where a left action becomes right and vice versa. For the Inverted Pendulum, the reflection changes the direction of the force applied to the base.

As outlined in Section 3.1.2, the transformations required for constructing equivariant layers must be linear and represented as matrices. Table 4.2 shows how these automorphisms are represented as matrices. Additionally, for value networks, the automorphisms need to obey the invariance relation as explicitly stated in Definition 2.10, therefore the output transformation needs to be always the identity, which ensures that the network outputs the same value for symmetric states or action-dependent states.

Environment	CartPole	Inverted Pendulum
Simulation Frame		
State Space	<p>The <b>state space</b> is a 4-dimensional vector:</p> $X_{state} = (x \ \theta \ \dot{x} \ \dot{\theta})$ <p><math>x</math>: Position of the cart (slider).  <math>\theta</math>: Vertical angle of the pole.  <math>\dot{x}</math>: Rate of change of the cart (slider) position.  <math>\dot{\theta}</math>: Rate of change of the pole's angle.</p>	
Action Space	<p>Binary action values <math>\{0,1\}</math> indicating the direction of the fixed force <math>F</math> applied on the cart:</p> $\text{action} \in \{F_{\text{left}}, F_{\text{right}}\}$	<p>Continuous force <math>F</math> applied to the base, determining force direction and magnitude.</p> $\text{action} \in [-F_{\text{max}}, F_{\text{max}}]$
Objective	<p>The goal is to keep the inverted pendulum or pole standing upright (within a certain angle limit) for as long as possible.</p>	

**Table 4.1:** Comparison of CartPole and Inverted Pendulum Environments

Automorphism	CartPole	Inverted Pendulum
$h^e = (f^e, g_s^e)$	$F^e = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ $G_s^e = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$F^e = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ $G_s^e = (1)$
$h^1 = (f^1, g_s^1)$	$F^1 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$ $G_s^1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$F^1 = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$ $G_s^1 = (-1)$

**Table 4.2:** Automorphisms and their matrix representation for CartPole and Inverted Pendulum

## 4.2 Metrics

A key metric used throughout all experiments is the *episodic return*, which represents the total reward collected by an agent per episode. Formally, the episodic return for the  $i$ th episode, denoted as  $R_i$ , is defined as:

### Metric 4.1: Episodic Return

$$R_i = \sum_{t=0}^{T_i} R_t,$$

where:

- $T_i$  is the final time step of the  $i$ th episode,
- $R_t$  is the reward received at time step  $t$ .

Episodic returns indicate the performance or success of an agent in completing a task within a single episode. By comparing the growth rate of episodic returns between different policies, we can assess their relative *sample efficiencies*. An agent whose episodic returns improve more quickly with the same number of training episodes is considered more sample-efficient.

The *episodic return* indicates how an agent performs per episode and not during the course of training. Thus, a numerical indicator is necessary to compare between policies, especially when fine-tuning, where a quantitative measure of overall performance (i.e. over the entire course of training) is needed. Therefore, we use the *cumulative average return* metric, inspired by [6]. This metric provides the average total return obtained per episode across all completed episodes up to a given point. An agent with higher cumulative average returns is more sample-efficient than one with lower cumulative average returns, as it consistently achieves higher rewards per episode on average. Formally:

### Metric 4.2: Cumulative Average Return

The cumulative average return  $\bar{R}(n)$  after  $n$  completed episodes is given by:

$$\bar{R}(n) = \frac{1}{n} \sum_{i=1}^n R_i, \quad (4.1)$$

where:

- $n$  is the number of completed episodes,
- $R_i$  is the total reward obtained in the  $i$ th episode.

Intuitively, as also noted by [29], leveraging symmetries present in the environment can accelerate learning. That is to say, if the agent exploits one symmetry, it can learn approximately twice as fast, and if the agent exploits two symmetries, learning can be up to four times faster. This efficiency can be rooted in the agent's ability to generalize its learned behavior from one symmetric state to others.

To evaluate the agent generalization across symmetric states, we use the *equivariance error* metric applied to the policy, formally defined as follows:



**Metric 4.3: Equivariance Mean Absolute Error for Policy**

Let  $h^i = (f^i, g_s^i)$  denote an automorphism in the symmetry group  $\mathcal{G}$ . The equivariance error for the policy  $\pi_\theta$  is defined as:

$$E_{\text{equiv}, \pi} = \frac{1}{|\mathcal{G}|} \sum_{i=1}^{|\mathcal{G}|} |\pi_\theta(f^i(\mathbf{s})) - g_s^i(\pi_\theta(\mathbf{s}))|,$$

where:

- $|\mathcal{G}|$  is the number of automorphisms in the symmetry group  $\mathcal{G}$ ,
- $f^i$  represents the transformation of the state  $\mathbf{s}$  under the  $i$ th automorphism,
- $g_s^i$  is the corresponding transformation applied to the policy's output  $\pi_\theta(\mathbf{s})$  under the  $i$ th automorphism.

Similarly, we use the *invariance error* metric to measure whether the value network outputs consistent values for symmetric states.

**Metric 4.4: Invariance Mean Absolute Error for Value Functions**

Let  $h^i = (f^i, g_s^i)$  denote an automorphism in the symmetry group  $\mathcal{G}$ . The invariance error for a state-value network  $V_\phi$  or action-value network  $Q_\phi$  is defined as follows: For the  $V$ -network:

$$E_{\text{invar}, V} = \frac{1}{|\mathcal{G}|} \sum_{i=1}^{|\mathcal{G}|} |V_\phi(f^i(\mathbf{s})) - V_\phi(\mathbf{s})|,$$

For the  $Q$ -network:

$$E_{\text{invar}, Q} = \frac{1}{|\mathcal{G}|} \sum_{i=1}^{|\mathcal{G}|} |Q_\phi(f^i(\mathbf{s}), g_s^i(\mathbf{a})) - Q_\phi(\mathbf{s}, \mathbf{a})|,$$

where:

- $|\mathcal{G}|$  is the number of automorphisms in the symmetry group  $\mathcal{G}$ ,
- $f^i$  represents the  $i$ th transformation applied to the input state  $\mathbf{s}$ ,
- $g_s^i$  represents the  $i$ th transformation applied to the action  $\mathbf{a}$  (relevant only for the  $Q$ -network).

These metrics evaluate whether policy or value networks output consistent actions across all symmetric states. Ideally, the result should be zero for a policy or a value network that perfectly respects the symmetry. While this is achievable for finite action spaces, in continuous action spaces, the policy may produce an action that is as close as possible to the action produced for a symmetric state but not exactly the same, so the error will not be zero. Therefore, either equivariance or invariance metrics serve as sanity checks and also evaluate approximation errors.

It is worth mentioning that in the case of continuous action spaces, performing feature permutation on the intermediate layers<sup>1</sup>, which theoretically should not impact equivariance according to [27] and [63], can result in hard constraints on the network turning into soft ones. These constraints then act as regularizers, meaning they encourage the policy or value network to be equivariant or invariant.

<sup>1</sup>For value networks, since the output transformation is the identity, symmetrization can lead to all weights being zeroed when the symmetry is a reflection, as in the case of the Inverted Pendulum. To avoid this, feature permutation should be applied to the intermediate layers.

### 4.3 Implementation and Fine-tuning Procedure

The implementations of the TD3 and PPO algorithms are based on the [CleanRL](#) framework [66], which offers high-quality, single-file implementations for most DRL algorithms. Additionally, we utilized the [symmetrizer](#) package [27] to construct equivariant layers and build MDP homomorphic networks. These networks serve as either policy and value networks in PPO and TD3. For the decision transformer implementation, we used the official code base provided by the authors [14] to develop a single-file implementation that includes both the standard decision transformer and its MDP homomorphic version using the [symmetrizer](#) package [27].

Furthermore, since the Decision Transformer is trained in an offline fashion, as explained in Section 2.1.3, we needed to collect trajectories and form a dataset. We followed an approach similar to the D4RL dataset [67], where we collected a dataset containing both optimal and suboptimal trajectories. The optimal trajectories (expert) were collected using a trained TD3 policy, as the Decision Transformer was evaluated only on the inverted pendulum environment. The suboptimal trajectories (medium) were collected by training a TD3 policy and performing early stopping after 20,000 interactions. Note that the TD3 policy used here was the conventional version and not the MDP homomorphic variant.

For experiment tracking and visualization, we integrated [Weights and Biases \(W&B\)](#), which enables us to log hyperparameters, monitor performance metrics, and compare different training runs. Hyperparameter fine-tuning was also automated using [W&B Sweeps](#).

We employed a Bayesian optimization approach for hyperparameter search. The hyperparameter space was defined around the default values provided for each algorithm, with the mean centered on the defaults and a standard deviation selected to cover a logical range for each parameter. The primary metric for fine-tuning was the cumulative average return, as defined in 4.2, which favors hyperparameters leading to faster convergence (i.e. higher sample efficiency). Each environment was run in parallel with two instances, each using a different seed.

It was observed that all algorithms tended to converge to their default hyperparameters as trials progressed, except for the learning rate. The MDP homomorphic networks required higher learning rates, which we attribute to smoother loss landscapes for both the actor and value networks compared to conventional networks of each algorithm. This observation aligns with [27], which also used higher learning rates for MDP homomorphic networks.

After completing hyper-parameter tuning, we ran each algorithm on randomly selected 5 seeds for both the MDP homomorphic and conventional versions of each algorithm.

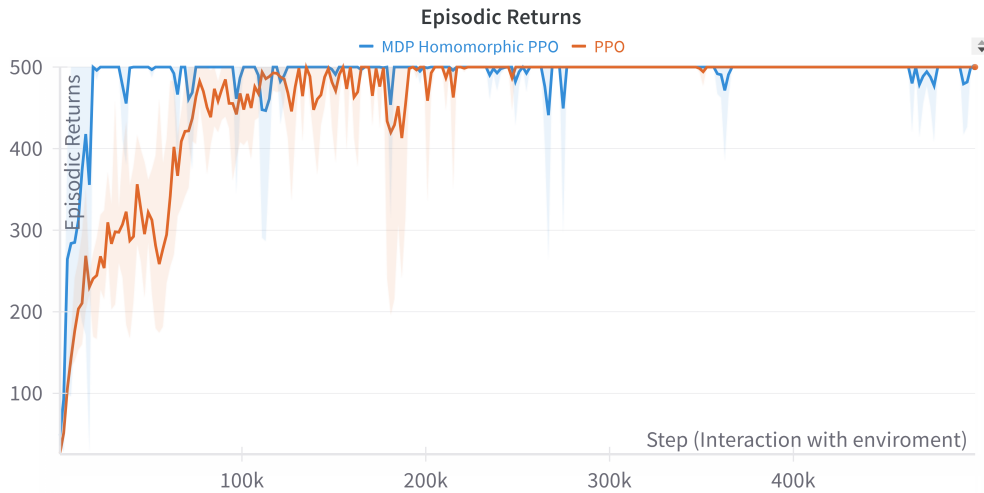
## 4.4 Results

This section presents the experimental results for MDP homomorphic and conventional versions of PPO, TD3, and Decision Transformer. The focus is on determining whether symmetry exploitation in the environment impacts sample efficiency and generalization. Each subsection compares the performance of the conventional algorithm with its MDP homomorphic version.

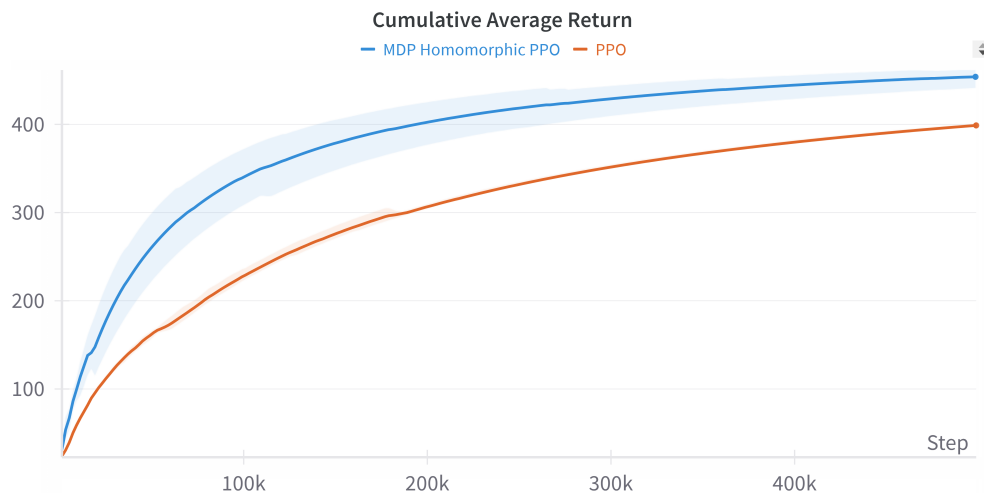
### 4.4.1 MDP Homomorphic PPO

The primary objective of this experiment is to replicate the results of [27] and establish that exploiting symmetry in the CartPole environment by encoding it as constraints on the parameter space of the policy and value neural networks is effective. The success of these experiments motivated our further extensions into continuous settings using alternative algorithms such as TD3, and higher-capacity models like the Decision Transformer.

We present training curves of episodic returns and cumulative average returns, aggregated across five random seeds for both standard PPO and MDP homomorphic PPO. The curves represent the mean performance, while the shaded regions indicate the range (minimum to maximum) of the five seeds. As shown in Figures 4.1 and 4.2, MDP Homomorphic PPO outperforms the standard PPO in terms of convergence speed and thus sample efficiency.



**Figure 4.1:** Training curves for episodic returns on CartPole environment. The orange curve represents the standard PPO, while the blue curve represents the MDP homomorphic PPO. The MDP homomorphic PPO converges faster to the maximum episodic return of 500, which indicates greater sample efficiency.



**Figure 4.2:** Training curves for cumulative average returns CartPole environment. The orange curve represents the standard PPO, while the blue curve represents the MDP homomorphic PPO. The MDP homomorphic PPO accumulates higher returns more quickly, which also indicates greater sample efficiency.

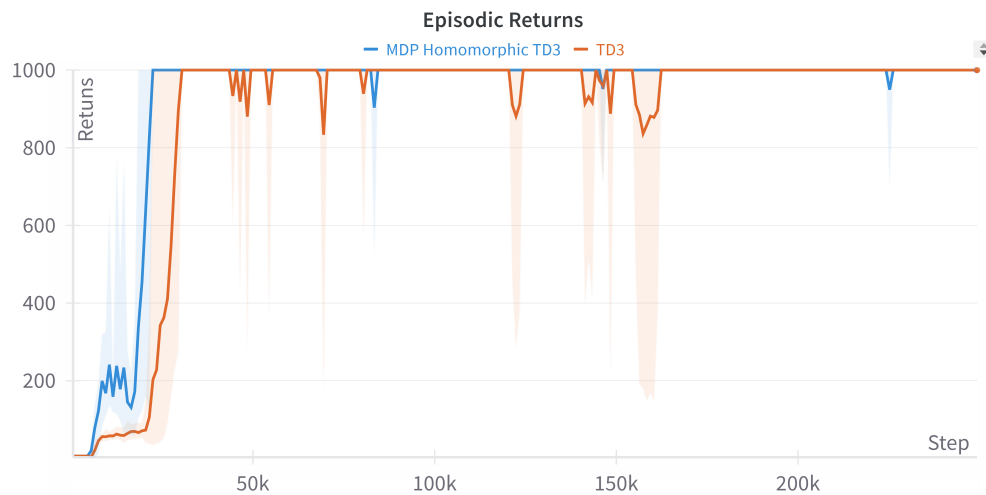
#### 4.4.2 MDP Homomorphic TD3

The primary objective of this experiment is to demonstrate that MDP homomorphisms can be extended to continuous control tasks by firstly obeying the conditions defined in 3.2, such as the continuity of the homomorphism maps, and secondly, by applying the minor modifications discussed in Section 3.2.3. These results are obtained for the Inverted Pendulum environment.

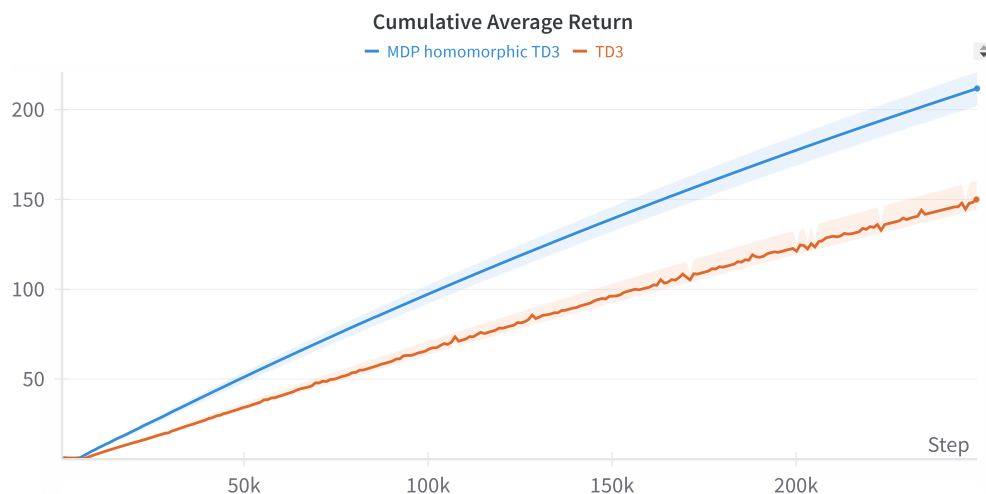
As in the previous subsection, we present training curves of episodic returns and cumulative average returns, aggregated across five random seeds for both standard TD3 and MDP homomorphic TD3. The curves represent the mean performance, while the shaded regions indicate the range (minimum to maximum) of the five seeds. As shown in Figures 4.3 and 4.4, MDP Homomorphic TD3 outperforms the standard TD3 in terms of convergence speed. However, the rate of convergence is slower compared to MDP homomorphic PPO on the CartPole experiments. This can be attributed to several factors: the approximation error inherent in continuous value and policy networks, as well as the increased complexity of the Inverted Pendulum environment.

In order to measure generalization across symmetric states, we utilize the equivariance and invariance error metrics defined in 4.3 and 4.4. Ideally, in MDP homomorphic networks, these errors should be zero, indicating that the policy or value network produces consistent behavior across symmetric states.

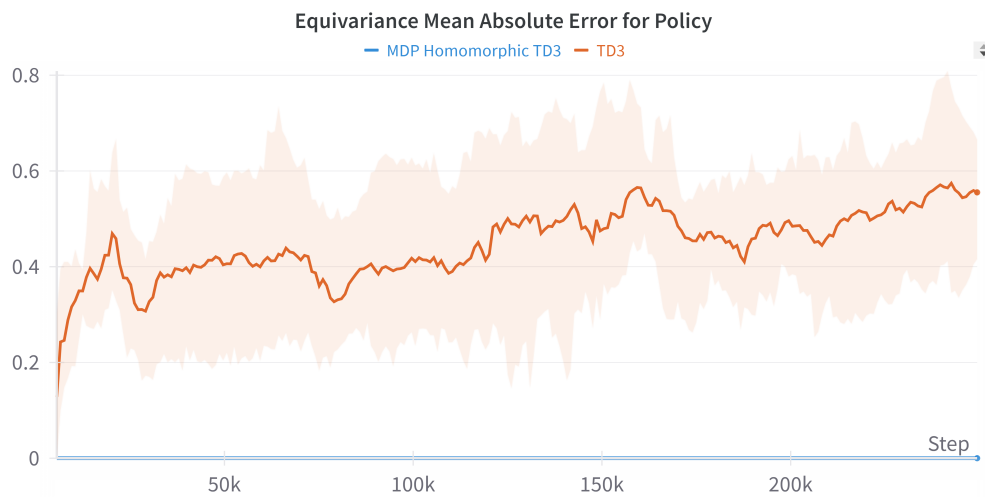
As shown in Figure 4.5, the equivariance error for the policy network of MDP-homomorphic TD3 remains near zero throughout training. However, for the Q-Network, this is not the case. This discrepancy can be attributed to the fact that the output transformation is the identity, and when input transformations such as reflections are applied during the symmetrization step (explained in Section 3.1.2), the sampled weights cancel each other, resulting in zero weights. To address this, we applied feature permutations to the intermediate layers to prevent weight cancelations. However, it was observed that feature permutations soften the symmetry constraints, making these constraints act as regularizers. As shown in Figure 4.6, the invariance error for the Q-Network is high at the start of training but decreases progressively as training continues.



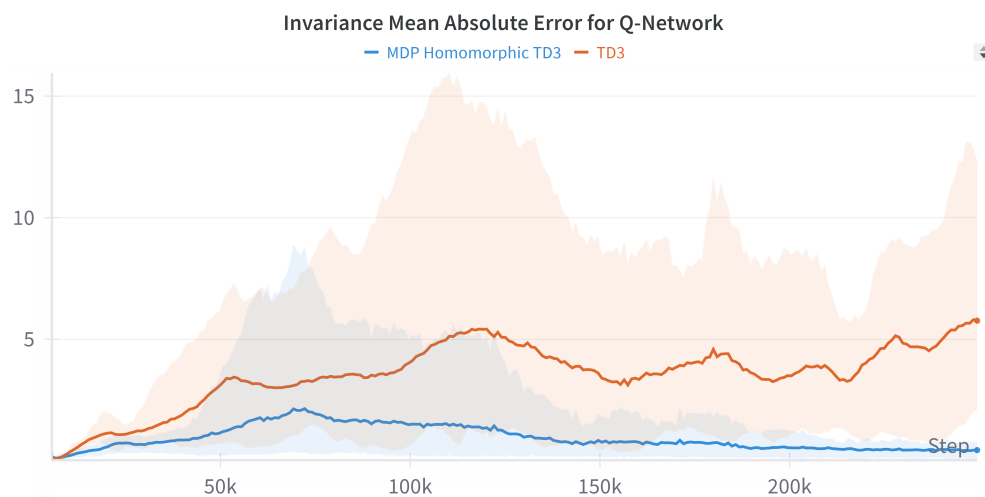
**Figure 4.3:** Training curves for episodic returns on the Inverted Pendulum environment. The orange curve represents the standard TD3, while the blue curve represents the MDP homomorphic TD3. The MDP homomorphic TD3 converges faster, which indicates greater sample efficiency also in continuous settings.



**Figure 4.4:** Training curves for cumulative average returns on the Inverted Pendulum environment. The orange curve represents the standard TD3, while the blue curve represents the MDP homomorphic TD3. The MDP homomorphic TD3 accumulates higher returns more quickly, which indicates greater sample efficiency in continuous settings.



**Figure 4.5:** Comparison between the equivariance errors for the policy network of MDP homomorphic TD3 and their standard versions. The equivariance error for the policy of MDP homomorphic TD3 remains near zero throughout training, which indicates higher generalization across symmetric states.

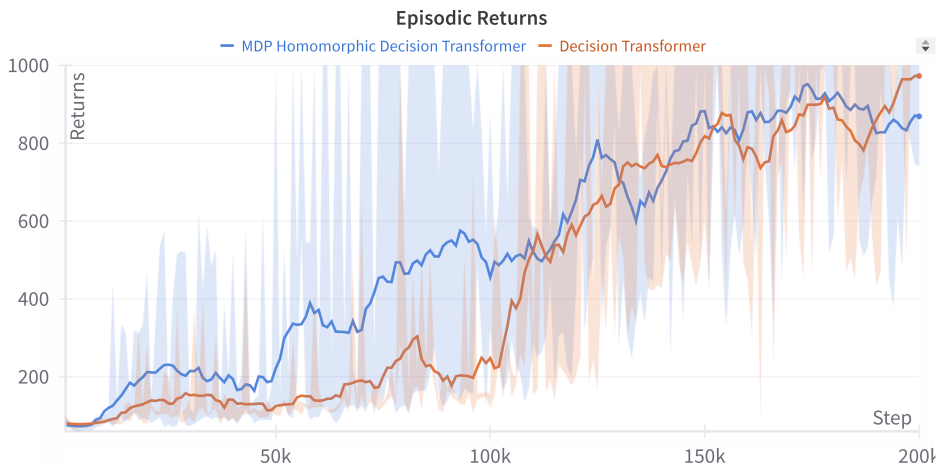


**Figure 4.6:** Comparison between invariance errors for the Q-value network of MDP homomorphic TD3 and their standard versions. The invariance error in the Q-Network of MDP homomorphic TD3 starts high but decreases steadily during training, which reflects improved generalization over time as symmetry constraints are acting here as regularizers.

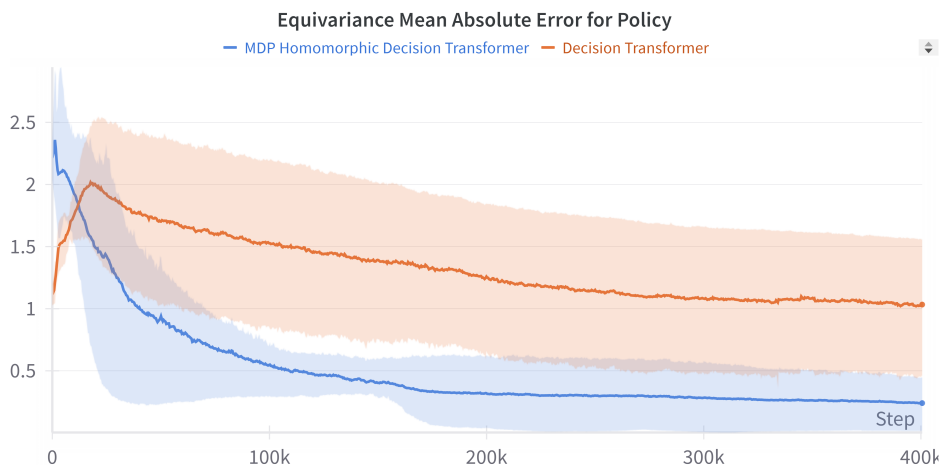
### 4.4.3 MDP Homomorphic Decision Transformer

The objective of this section is to evaluate the extent to which symmetry exploitation in state-action sequences ( as explained in Section 3.3 ) improves the sample efficiency of the Decision Transformer. To this end, we trained both the standard Decision Transformer and its MDP homomorphic variant on a dataset containing optimal and suboptimal trajectories, as mentioned previously.

We present the learning curves of episodic returns, using five random seeds for both the standard Decision Transformer and its MDP homomorphic variant. And as shown in Figure 4.7, some improvement in convergence speed is observed. Regarding the equivariance error, measured using the metric described in 4.3, we observe that symmetry constraints on the embedding layers and attention mechanism act as a regularizer due to the permutation of features performed for the RTG values. As shown in Figure 4.8, the equivariance error is initially high at the start of training but progressively decreases as training continues.



**Figure 4.7:** Learning curves of episodic returns for standard Decision Transformer and MDP Homomorphic variant.



**Figure 4.8:** Equivariance error during training for the MDP Homomorphic Decision Transformer and MDP Homomorphic variant.

## 5 Conclusion and Recommendations

### 5.1 Conclusion

This thesis explored improving sample efficiency and generalization in transformer-based reinforcement learning by leveraging structural symmetries in control tasks. Large-capacity models like Decision Transformer have shown promise in continuous control tasks but suffer from data inefficiency, meaning they require large amounts of data to perform and generalize effectively. Therefore, our primary motivation is to mitigate sample inefficiency and increase generalization in transformer-based reinforcement learning by leveraging structural symmetry at the control behavior level.

We utilized MDP Homomorphism, a framework designed to simplify MDPs by identifying structurally similar states and actions, thereby creating abstract, reduced versions of the original MDPs. Inspired by [28], we established a connection between MDP homomorphisms and equivariant neural networks via the notion of automorphisms. This enabled us to constrain neural networks to respect the symmetries inherent in the environment.

Traditionally, the MDP homomorphism framework has been limited to discrete control problems with finite state and action spaces and lacked provisions for handling continuous symmetries. To address this, we first validated the framework by replicating results from prior work [27], which employed group-equivariant learning to enforce MDP homomorphism in environments with discrete action spaces. Our replication results in Section 4.4.1 confirmed that leveraging symmetry improves sample efficiency.

Subsequently, we extend the MDP homomorphism formulation from finite to continuous settings to address control problems with continuous state and action spaces, as well as the exploitation of both discrete and continuous symmetries. Building on the successful replication results, we show that with minor modifications we can extend the automated algorithm for constructing equivariant layers from [27] to handle the continuous settings. Our results in Section 4.4.2 demonstrate that the TD3 algorithm becomes more sample-efficient and highly generalizable to unseen states.

Having generalized MDP homomorphisms to continuous domains, we applied the concept to sequence models, specifically Decision Transformers. This involved introducing the notion of trajectory equivalence under MDP homomorphism and integrating equivariant neural networks from [27] into the transformer architecture. The results in Section 4.4.3 indicate some improvement in sample efficiency.

Therefore, our findings confirm the hypothesis:

#### Proven Hypothesis 1

Exploitation of symmetries in RL improves sample efficiency and model generalization across symmetric states and actions.

### 5.2 Recommendations

The findings of this thesis demonstrate the potential of leveraging structural symmetries to improve sample efficiency and generalization in reinforcement learning and transformer-based reinforcement learning. We highlight several areas for improvement and future exploration:

1. Due to time constraints, we were unable to fully implement and evaluate our methodology on the Reacher environment, which exhibits continuous symmetries. Future



work can focus on evaluating the MDP homomorphism framework by testing whether it can exploit continuous symmetry to improve sample efficiency and investigating whether the MDP Homomorphic Network, constrained by elements representing continuous symmetries, introduces excessive constraints that limit the expressivity of the network. Since we have already analyzed and studied the Reacher environment theoretically, details regarding its setup and proofs related to MDP homomorphism are provided in Appendix D.

2. While the equivariant layer construction by [27] proved effective, it introduces computational overhead during training. For example, a standard RL algorithm such as PPO or TD3 requires only one-third of the training time compared to the same algorithm integrated with an MDP homomorphic network. Similarly, in the case of the MDP homomorphic Decision Transformer architecture, training time increased from 1.5 hours with the conventional architecture to 6 hours with the equivariant layers.

Thus, further investigation is required to either optimize the current method or explore alternative methodologies based on the symmetrization approach by [64] for building equivariant networks. Additionally, current research on equivariant MLP construction is limited, as most studies focus on CNNs with imagery data as input. One notable exception is the work by [68], which introduces methods for constructing equivariant MLPs for both discrete and continuous symmetries. This approach could serve as a promising candidate for use as an equivariant layer in MDP homomorphic networks. However, further analysis is needed to establish its efficiency and practicality.

3. For the Decision Transformer architecture, where only future actions are predicted based on past sequences of actions, states, and RTG, it would be worth exploring MDP Homomorphic Decision Transformers in a setup where predictions include not only actions but also states and reward values, as in the Trajectory Transformer. A trained agent in this setting could be utilized as a predictive model that is equivariant with respect to the environment's symmetry. This direction presents an interesting avenue for future work.

## Bibliography

- [1] Jiyu Cheng, Hu Cheng, Max Q.H. Meng, and Hong Zhang. Autonomous Navigation by Mobile Robots in Human Environments: A Survey. *2018 IEEE International Conference on Robotics and Biomimetics, ROBIO 2018*, pages 1981–1986, 7 2018.
- [2] Aude Billard and Danica Kragic. Trends and challenges in robot manipulation. *Science*, 364(6446), 6 2019.
- [3] Swapnil Patil, V. Vasu, and K. V. S. Srinadh. Advances and perspectives in collaborative robotics: a review of key technologies and emerging trends. *Discover Mechanical Engineering 2023 2:1*, 2(1):1–19, 8 2023.
- [4] Zhong Sheng Hou and Zhuo Wang. From model-based control to data-driven control: Survey, classification and perspective. *Information Sciences*, 235:3–35, 2013.
- [5] Richard S Sutton and Andrew G Barto. Reinforcement Learning: An Introduction Second edition, in progress.
- [6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning.
- [7] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature 2016 529:7587*, 529(7587):484–489, 1 2016.
- [8] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to Walk via Deep Reinforcement Learning. *Robotics: Science and Systems*, 12 2018.
- [9] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations.
- [10] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep Reinforcement Learning that Matters. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pages 3207–3214, 9 2017.
- [11] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A Study on Overfitting in Deep Reinforcement Learning.
- [12] Claire Glanois, Paul Weng, Matthieu Zimmer, Dong Li, Tianpei Yang, Jianye Hao, and Wulong Liu. A Survey on Interpretable Reinforcement Learning.
- [13] Kei Ota, Devesh K Jha, and Asako Kanezaki. Training Larger Networks for Deep Reinforcement Learning.
- [14] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision Transformer: Reinforcement Learning via Sequence Modeling.

- [15] Michael Janner, Qiyang Li, and Sergey Levine. Offline Reinforcement Learning as One Big Sequence Modeling Problem.
- [16] Shengchao Hu, Li Shen, Ya Zhang, Yixin Chen, and Dacheng Tao. On Transforming Reinforcement Learning with Transformers: The Development Trajectory.
- [17] Pranav Agarwal and Simon J D Prince. Transformers in Reinforcement Learning: A Survey. 1, 2023.
- [18] Wenzhe Li, Hao Luo, Zichuan Lin, Chongjie Zhang, Zongqing Lu, and Deheng Ye. A Survey on Transformers in Reinforcement Learning.
- [19] Roya Firoozi, Johnathan Tucker, Stephen Tian, Anirudha Majumdar, Jiankai Sun, Weiyu Liu, Yuke Zhu, Shuran Song, Ashish Kapoor, Karol Hausman, Brian Ichter, Danny Driess, Jiajun Wu, Cewu Lu, and Mac Schwager. Foundation Models in Robotics: Applications, Challenges, and the Future. 2023.
- [20] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-1: Robotics Transformer for Real-World Control at Scale. 12 2022.
- [21] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. *Proceedings of Machine Learning Research*, 229, 7 2023.
- [22] Open X-Embodiment Collaboration, Abby O’Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew Wang, Andrey Kolobov, Anikait Singh, Animesh Garg, Aniruddha Kembhavi, Annie Xie, Anthony Brohan, Antonin Raffin, Archit Sharma, Arefeh Yavary, Arhan Jain, Ashwin Balakrishna, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schölkopf, Blake Wulfe, Brian Ichter, Cewu Lu, Charles Xu, Charlotte Le, Chelsea Finn, Chen Wang, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Christopher Agia, Chuer Pan, Chuyuan Fu, Coline Devin, Danfei Xu, Daniel Morton, Danny Driess, Daphne Chen, Deepak Pathak, Dhruv Shah, Dieter Büchler, Dinesh Jayaraman, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Ethan Foster,

- Fangchen Liu, Federico Ceola, Fei Xia, Feiyu Zhao, Felipe Vieira Frujeri, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Gilbert Feng, Giulio Schiavi, Glen Berseth, Gregory Kahn, Guangwen Yang, Guanzhi Wang, Hao Su, Hao-Shu Fang, Haochen Shi, Henghui Bao, Heni Ben Amor, Henrik I Christensen, Hiroki Furuta, Homanga Bharadhwaj, Homer Walke, Hongjie Fang, Huy Ha, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jad Abou-Chakra, Jaehyung Kim, Jaimyn Drake, Jan Peters, Jan Schneider, Jasmine Hsu, Jay Vakil, Jeannette Bohg, Jeffrey Bingham, Jeffrey Wu, Jensen Gao, Jiaheng Hu, Jiajun Wu, Jialin Wu, Jiankai Sun, Jianlan Luo, Jiayuan Gu, Jie Tan, Jihoon Oh, Jimmy Wu, Jingpei Lu, Jingyun Yang, Jitendra Malik, João Silvério, Joey Hejna, Jonathan Booher, Jonathan Tompson, Jonathan Yang, Jordi Salvador, Joseph J. Lim, Junhyek Han, Kaiyuan Wang, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Black, Kevin Lin, Kevin Zhang, Kiana Ehsani, Kiran Lekkala, Kirsty Ellis, Krishan Rana, Krishnan Srinivasan, Kuan Fang, Kunal Pratap Singh, Kuo-Hao Zeng, Kyle Hatch, Kyle Hsu, Laurent Itti, Lawrence Yunliang Chen, Lerrel Pinto, Li Fei-Fei, Liam Tan, Linxi "Jim" Fan, Lionel Ott, Lisa Lee, Luca Weihs, Magnum Chen, Marion Lepert, Marius Memmel, Masayoshi Tomizuka, Masha Itkina, Mateo Guaman Castro, Max Spero, Maximilian Du, Michael Ahn, Michael C. Yip, Mingtong Zhang, Mingyu Ding, Minh Ho, Mohan Kumar Srirama, Mohit Sharma, and Moo Jin Kim. Open X-Embodiment: Robotic Learning Datasets and RT-X Models. 10 2023.
- [23] Rogerio Bonatti, Sai Vemprala, Shuang Ma, Felipe Frujeri, Shuhang Chen, and Ashish Kapoor. PACT: Perception-Action Causal Transformer for Autoregressive Robotics Pre-Training. *IEEE International Conference on Intelligent Robots and Systems*, pages 3621–3627, 9 2022.
- [24] Yanchao Sun, Shuang Ma, Ratnesh Madaan, Rogerio Bonatti, Furong Huang, and Ashish Kapoor. SMART: Self-supervised Multi-task pretraining with control Transformers. 1 2023.
- [25] Zhongkai Hao, Songming Liu, Yichi Zhang, Chengyang Ying, Yao Feng, Hang Su, Jun Zhu, Zhongkai Hao, Songming Liu, Yichi Zhang, Chengyang Ying, Yao Feng, Hang Su, and Jun Zhu. Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications. *arXiv*, page arXiv:2211.08064, 2022.
- [26] Chayan Banerjee, Kien Nguyen, Clinton Fookes, and Maziar Raissi. A Survey on Physics Informed Reinforcement Learning: Review and Open Problems. 9 2023.
- [27] Van Der Pol, Van Hoof, Elise van der Pol, Daniel E Worrall, Herke van Hoof UvA-Bosch Deltalab, Frans A Oliehoek, and Max Welling. UvA-DARE (Digital Academic Repository) MDP homomorphic networks: Group symmetries in reinforcement learning MDP Homomorphic Networks: Group Symmetries in Reinforcement Learning. 6:4199–4210, 2020.
- [28] Balaraman Ravindran and Andrew G. Barto. Symmetries and Model Minimization in Markov Decision Processes.
- [29] Dian Wang, Robin Walters, and Robert Platt.  $\mathrm{SO}(2)$ -Equivariant Reinforcement Learning. 2022.
- [30] Dian Wang, Colin Kohler, and Robert Platt. Policy learning in  $\mathrm{SE}(3)$  action spaces.
- [31] Beomyeol Yu and Taeyoung Lee. Equivariant Reinforcement Learning for Quadrotor UAV. 2022.

- [32] Farzad Abdolhosseini, Hung Yu Ling, Zhaoming Xie, Xue Bin Peng, and Michiel Van De Panne. On Learning Symmetric Locomotion. 19, 2019.
- [33] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov Openai. Proximal Policy Optimization Algorithms. 7 2017.
- [34] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. 2018.
- [35] Ariel Kwiatkowski, Mark Towers, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A Standard Interface for Reinforcement Learning Environments. 7 2024.
- [36] C. Saori Tanaka, Kenji Doya, Go Okada, Kazutaka Ueda, Yasumasa Okamoto, and Shigeto Yamawaki. Prediction of immediate and future rewards differentially recruits cortico-basal ganglia loops. *Nature Neuroscience* 2004 7:8, 7(8):887–893, 7 2004.
- [37] Lucian Buşoniu, Tim de Bruin, Domagoj Tolić, Jens Kober, and Ivana Palunko. Reinforcement learning for control: Performance, stability, and deep approximators. *Annual Reviews in Control*, 46:8–28, 1 2018.
- [38] Fei He and Yuan Yang. Nonlinear system identification of neural systems from neurophysiological signals. *Neuroscience*, 458:213, 3 2020.
- [39] Gerald Tesauro. TD-Gammon, A Self-Teaching Backgammon Program, Achieves Master-Level Play. 1993.
- [40] Richard S Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation.
- [41] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [42] Hado Van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pages 2094–2100, 9 2015.
- [43] Ziyu Wang, Tom Schaul, Matteo Hessel, and Marc Lanctot. Dueling Network Architectures for Deep Reinforcement Learning Hado van Hasselt.
- [44] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.
- [45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *Advances in Neural Information Processing Systems*, 2017-December:5999–6009, 6 2017.
- [46] Alec Radford Openai, Karthik Narasimhan Openai, Tim Salimans Openai, and Ilya Sutskever Openai. Improving Language Understanding by Generative Pre-Training.
- [47] Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 1:4171–4186, 10 2018.

- [48] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *ICLR 2021 - 9th International Conference on Learning Representations*, 10 2020.
- [49] Linhao Dong, Shuang Xu, and Bo Xu. Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2018-April:5884–5888, 9 2018.
- [50] Hassan Akbari, Liangzhe Yuan, Rui Qian, Wei-Hong Chuang, Shih-Fu Chang, Yin Cui, and Boqing Gong. VATT: Transformers for Multimodal Self-Supervised Learning from Raw Video, Audio and Text. *Advances in Neural Information Processing Systems*, 34:24206–24221, 12 2021.
- [51] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners.
- [52] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, 2020-December, 5 2020.
- [53] Christopher M. Bishop. Pattern Recognition and Machine Learning. *Journal of Electronic Imaging*, 16(4):049901, 1 2007.
- [54] Adi Haviv, Ori Ram, Ofir Press, Peter Izsak, and Omer Levy. Transformer Language Models without Positional Encodings Still Learn Positional Information.
- [55] Kazuki Irie, Albert Zeyer, Ralf Schlüter, and Hermann Ney. Language Modeling with Deep Transformers. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2019-September:3905–3909, 5 2019.
- [56] Robert Givan, Thomas Dean, and Matthew Greig. Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1-2):163–223, 7 2003.
- [57] Balaraman Ravindran and Andrew G Barto. An Algebraic Approach to Abstraction in Reinforcement Learning.
- [58] Balaraman Ravindran and Andrew G Barto. Approximate Homomorphisms: A framework for non-exact minimization in Markov Decision Processes.
- [59] Sahand Rezaei-Shoshtari, Rosie Zhao, Prakash Panangaden, David Meger, and Doina Precup. Continuous MDP Homomorphisms and Homomorphic Policy Gradient. *Advances in Neural Information Processing Systems*, 35, 9 2022.
- [60] Ondrej Biza and Robert Platt. Online Abstraction with MDP Homomorphisms for Deep Learning \*. *IFAAMAS*, 9.
- [61] Balaraman Ravindran and Arun Tejasvi Chaganty. Discovering Continuous Homomorphisms for Transfer. Technical report, 2012.

- [62] Achiam Joshua. Spinning Up in Deep Reinforcement Learning, 2018.
- [63] Taco S Cohen and Max Welling MWELLING. Group Equivariant Convolutional Networks. 2016.
- [64] Dmitry Yarotsky. Universal approximations of invariant maps by neural networks.
- [65] Nelson Elhage, Catherine Olsson, and Nada Neel. A Mathematical Framework for Transformer Circuits.
- [66] Shengyi Huang, Rousslan Fernand, Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G M Araújo. CleanRL: High-quality Single-file Implementations of Deep Reinforcement Learning Algorithms. Technical report, 2022.
- [67] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for Deep Data-Driven Reinforcement Learning. 4 2020.
- [68] Marc Finzi, Max Welling, and Andrew Gordon Wilson. A Practical Method for Constructing Equivariant Multilayer Perceptrons for Arbitrary Matrix Groups.
- [69] Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195, 1999.

## A Continuous Optimal Value Equivalence

In this section, we prove the continuous optimal value equivalence based on the  $m$ -step optimal discounted action-value function. The proof is inspired by the finite case presented in [28].

### Theorem A.1: Continuous Optimal Value Equivalence

Let  $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma)$  be a continuous MDP, and let  $\bar{\mathcal{M}} = (\bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{\mathcal{T}}, \bar{r}, \gamma)$  be its homomorphic image under the continuous MDP homomorphism  $h = \langle f, \{g_s \mid s \in \mathcal{S}\} \rangle$ , where:

$$h((s, a)) = (f(s), g_s(a)).$$

Then, for any  $(s, a) \in \mathcal{S} \times \mathcal{A}$ :

$$Q^*(s, a) = \bar{Q}^*(f(s), g_s(a)).$$

*Proof.* We prove this by induction on the non-negative integer  $m$ , where  $Q_m(s, a)$  represents the  $m$ -step optimal discounted action-value function, as a specialized form of (2.10):

$$Q_m(s, a) = r(s, a) + \gamma \int_{s' \in \mathcal{S}} \mathcal{T}(s' \mid s, a) V_{m-1}(s') ds', \quad (\text{A.1})$$

where

$$V_{m-1}(s') = \max_{a' \in \mathcal{A}_{s'}} Q_{m-1}(s', a'). \quad (\text{A.2})$$

**Base Case ( $m = 0$ )** For  $m = 0$ , the action-value function is defined as:

$$Q_0(s, a) = r(s, a).$$

Given the reward preservation property from Definition 3.2:

$$\bar{r}(f(s), g_s(a)) = r(s, a),$$

we have:

$$Q_0(s, a) = \bar{Q}_0(f(s), g_s(a)).$$

This establishes the base case.

**Inductive Step ( $m \geq 1$ )** Assume that for some  $m \geq 1$ , the equivalence holds for all steps up to  $m - 1$ :

$$Q_{m-1}(s, a) = \bar{Q}_{m-1}(f(s), g_s(a)) \quad \text{for all } (s, a) \in \mathcal{S} \times \mathcal{A}.$$

We now show that:

$$Q_m(s, a) = \bar{Q}_m(f(s), g_s(a)) \quad \text{for all } (s, a) \in \mathcal{S} \times \mathcal{A}.$$

Starting from equation (A.1), we apply the induction hypothesis:

$$Q_{m-1}(s', a') = \bar{Q}_{m-1}(f(s'), g_{s'}(a')) \quad \text{for all } (s', a') \in \mathcal{S} \times \mathcal{A}.$$

Thus:

$$V_{m-1}(s') = \max_{a' \in \mathcal{A}_{s'}} Q_{m-1}(s', a') = \max_{a' \in \mathcal{A}_{s'}} \bar{Q}_{m-1}(f(s'), g_{s'}(a')) = \bar{V}_{m-1}(f(s')).$$



Substitute the continuous homomorphism properties from Definition 3.2 into equation (A.1):

$$\begin{aligned}\bar{r}(f(s), g_s(a)) &= r(s, a), \\ \bar{\mathcal{T}}(f(s') | f(s), g_s(a)) &= \int_{s'' \in f^{-1}(\bar{s}')} \mathcal{T}(s'' | s, a) ds'',\end{aligned}$$

we get:

$$Q_m(s, a) = \bar{r}(f(s), g_s(a)) + \gamma \int_{\bar{s}' \in \bar{\mathcal{S}}} \left( \int_{s'' \in f^{-1}(\bar{s}')} \mathcal{T}(s'' | s, a) ds'' \right) \bar{V}_{m-1}(\bar{s}') d\bar{s}'. \quad (\text{A.3})$$

The corresponding  $m$ -step action-value function in  $\bar{\mathcal{M}}$  is:

$$\bar{Q}_m(f(s), g_s(a)) = \bar{r}(f(s), g_s(a)) + \gamma \int_{\bar{s}' \in \bar{\mathcal{S}}} \bar{\mathcal{T}}(\bar{s}' | f(s), g_s(a)) \bar{V}_{m-1}(\bar{s}') d\bar{s}'. \quad (\text{A.4})$$

Since equations (A.3) and (A.4) are identical, we have:

$$Q_m(s, a) = \bar{Q}_m(f(s), g_s(a)) \quad \text{for all } (s, a) \in \mathcal{S} \times \mathcal{A}.$$

### Conclusion

Since  $\lim_{m \rightarrow \infty} Q_m(s, a) = Q^*(s, a)$ , it follows that  $Q^*(s, a) = \bar{Q}^*(f(s), g_s(a))$ .

This completes the proof. □

## B Universal Approximations of Equivariant Maps by Neural Networks

The author in [64] utilizes the Universal Approximation Theorem [69] as a foundational framework for proposing methods to approximate equivariant and invariant continuous maps. This theorem establishes that neural networks with non-polynomial continuous activation functions are capable of approximating any continuous function defined on finite-dimensional spaces.

### Theorem B.1: Universal Approximation Theorem [69]

Let  $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  be a continuous activation function that is not a polynomial. Let  $V = \mathbb{R}^d$  be a real finite-dimensional vector space. Then any continuous map  $f : V \rightarrow \mathbb{R}$  can be approximated, in the sense of uniform convergence on compact sets, by maps  $\hat{f} : V \rightarrow \mathbb{R}$  of the form:

$$\hat{f}(x) = \sum_{n=1}^N c_n \sigma \left( \sum_{k=1}^d w_{nk} x_k + h_n \right),$$

with some coefficients  $c_n \in \mathbb{R}$ ,  $w_{nk} \in \mathbb{R}$ , and  $h_n \in \mathbb{R}$ .

Building on theorem B.1, the study considers functions that remain unchanged under the action of a group  $\Gamma$ . These are known as invariant functions. The space  $V$  is assumed to carry a representation  $R$  of the group  $\Gamma$ , mapping group elements to linear transformations of  $V$ . A map  $f : V \rightarrow U$  is  $\Gamma$ -equivariant if

$$f(R_\gamma \mathbf{x}) = R_\gamma f(\mathbf{x})$$

for all  $\gamma \in \Gamma$  and  $\mathbf{x} \in V$ . The key result for invariant functions is:

### Proposition B.1: Invariant Approximation [64]

Let  $\Gamma$  be a compact group, and  $V$  a finite-dimensional  $\Gamma$ -module with a representation  $R : \Gamma \rightarrow \text{GL}(V)$ . Any continuous  $\Gamma$ -invariant map  $f : V \rightarrow \mathbb{R}$  can be approximated by  $\Gamma$ -invariant maps  $\hat{f} : V \rightarrow \mathbb{R}$  of the form:

$$\hat{f}(x) = \int_{\Gamma} \sum_{n=1}^N c_n \sigma (l_n(R_\gamma x) + h_n) d\gamma,$$

where:

- $c_n \in \mathbb{R}$  are weights assigned to each term in the summation,
- $h_n \in \mathbb{R}$  are bias terms for the activation function,
- $l_n \in V^*$  are linear functionals acting on  $V$ ,
- $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  is a continuous non-polynomial activation function,
- $d\gamma$  is the normalized Haar measure on  $\Gamma$ , ensuring proper integration over the group.

Next, the study extends to equivariant functions, which transform in a consistent manner under group actions. Specifically, for two  $\Gamma$ -modules  $V$  and  $U$ , a map  $f : V \rightarrow U$  is  $\Gamma$ -equivariant if  $f(R_\gamma \mathbf{x}) = R_\gamma f(\mathbf{x})$  for all  $\gamma \in \Gamma$ . The result for equivariant functions is:

**Proposition B.2: Equivariant Approximation [64]**

Let  $\Gamma$  be a compact group, and  $V$  and  $U$  be finite-dimensional  $\Gamma$ -modules with respective representations  $R : \Gamma \rightarrow \text{GL}(V)$  and  $R : \Gamma \rightarrow \text{GL}(U)$ . Any continuous  $\Gamma$ -equivariant map  $f : V \rightarrow U$  (including the invariant case when  $U = \mathbb{R}$ ) can be approximated by  $\Gamma$ -equivariant maps  $\hat{f} : V \rightarrow U$  of the form:

$$\hat{f}(x) = \int_{\Gamma} \sum_{n=1}^N R_{\gamma}^{-1} y_n \sigma(l_n(R_{\gamma}x) + h_n) d\gamma,$$

where:

- $y_n \in U$  are output vectors (scalars in the invariant case),
- $R_{\gamma}^{-1}$  is the inverse of the group representation on  $U$ ,
- $l_n \in V^*$  are linear functionals extracting components of the input,
- $h_n \in \mathbb{R}$  are biases for the activation function,
- $\sigma : \mathbb{R} \rightarrow \mathbb{R}$  is a fixed continuous non-polynomial activation function,
- $d\gamma$  is the normalized Haar measure on  $\Gamma$ .

## C TD3 Algorithm

---

**Algorithm 4:** TD3 [62]
 

---

[1] : **input:** initial policy parameters  $\theta$ , Q-function parameters  $\phi_1, \phi_2$ , empty replay buffer  $\mathcal{D}$

[2] : Set target parameters equal to main parameters:  $\theta_{\text{targ}} \leftarrow \theta, \phi_{\text{targ},1} \leftarrow \phi_1, \phi_{\text{targ},2} \leftarrow \phi_2$

[3] : **repeat**

[4] :   Observe state  $s$  and select action  $a = \text{clip}(\pi_\theta(s) + \epsilon, a_{\text{Low}}, a_{\text{High}})$ , where  $\epsilon \sim \mathcal{N}$

[5] :   Execute action  $a$  in the environment

[6] :   Observe next state  $s'$ , reward  $r$ , and done signal  $d$  indicating if  $s'$  is terminal

[7] :   Store  $(s, a, r, s', d)$  in replay buffer  $\mathcal{D}$

[8] :   **if**  $s'$  is terminal **then**

      Reset the environment state

**end**

[9] :   **if** it's time to update **then**

[10] :     **for**  $j$  in range (however many updates) **do**

[11] :       Randomly sample a batch of transitions,  $B = \{(s, a, r, s', d)\}$  from  $\mathcal{D}$

[12] :       Compute target actions:

$$a'(s') = \text{clip}(\pi_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{\text{Low}}, a_{\text{High}}), \quad \epsilon \sim \mathcal{N}(0, \sigma)$$

[13] :       Compute targets:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

[14] :       Update Q-functions by one step of gradient descent using:

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{\phi_i}(s, a) - y(r, s', d))^2, \quad \text{for } i = 1, 2$$

[15] :       **if**  $j \bmod \text{policy\_delay} = 0$  **then**

      Update policy by one step of gradient ascent using:

$$\nabla_{\theta} \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \pi_\theta(s))$$

[16] :       Update target networks with:

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho) \phi_i, \quad \text{for } i = 1, 2$$

$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho) \theta$$

**end**

**end**

**end**

**until** convergence

---

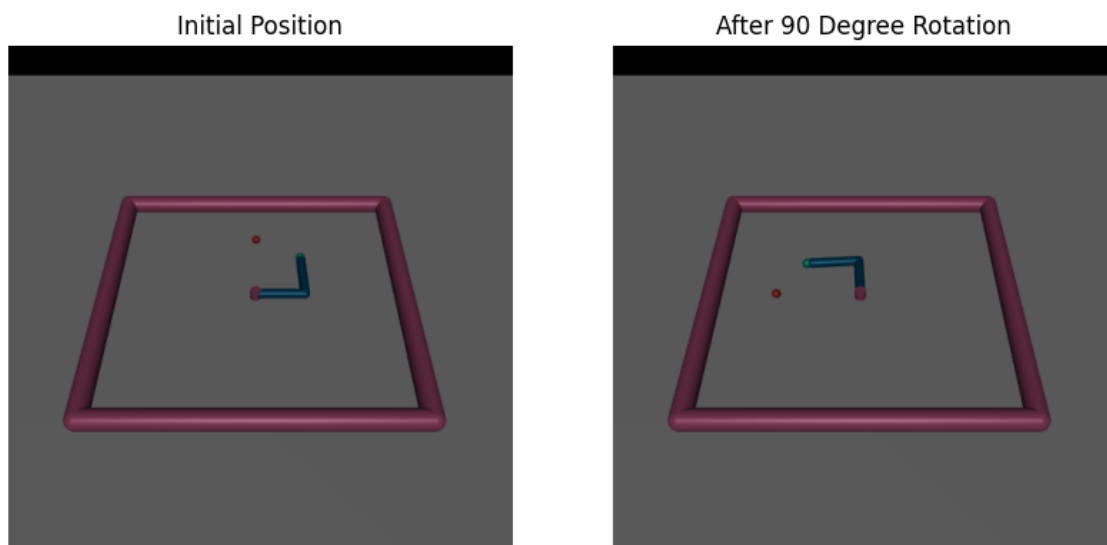


## D Reacher Environment

The reacher environment is a two-joint robotic arm that must move its end-effector (fingertip) close to a target, which spawned at random position. The Reacher exhibits continuous symmetry, which can be explained as follows: imagine the target is positioned on a circle with a fixed radius and that the initial position of the end-effector is on this circle. Then, if the target and the first link are rotated by the same angle  $\theta$  while the second link remains fixed, the fingertip traces a circular path. The sequence of actions leading the fingertip to the target under these conditions would be equivalent and exhibit continuous rotational symmetry in the state-action pairs. This continuous symmetry is characterized by the  $SO(2)$ , which represents the group of all rotations by any angle around a central point in a 2D plane. It is continuous, implying an infinite set of possible rotational symmetries.

We can also exploit discrete symmetry in the Reacher, which would be subsets of  $SO(2)$ , such as  $C_4$  and  $C_8$ . These groups denoted as  $C_n$  represents the cyclic group of order  $n$ , corresponding to rotations by discrete angles, specifically multiples of  $\frac{2\pi}{n}$  radians. For  $C_4$ , this means rotations by increments of  $\frac{\pi}{2}$  radians. An example of  $\frac{\pi}{2}$  rotation is given in Figure D.1.

Symmetry in Reacher Environment



**Figure D.1:** 90-degree rotation is applied to the first joint and the target, where it can be intuitively interpreted as two equivalent state-action pair

Mathematically, for the Reacher environment, a state  $s \in S$  is represented as <sup>1</sup>:

$$\mathbf{s} = \begin{bmatrix} \cos(\theta_1) \\ \sin(\theta_1) \\ \cos(\theta_2) \\ \sin(\theta_2) \\ x_t \\ y_t \\ \dot{\theta}_1 \\ \dot{\theta}_2 \\ x_f - x_t \\ y_f - y_t \end{bmatrix} \quad (\text{D.1})$$

where:

- $\theta_1$  and  $\theta_2$  are the angles of the first and second joints, respectively.
  - $\theta_1$  is the absolute angle between the first link and the x-axis.
  - $\theta_2$  is the relative angle between the first(i.e. when extended) and second links.
- $\dot{\theta}_1$  and  $\dot{\theta}_2$  are the rate of change of the first and second joints, respectively.
- $x_t$  and  $y_t$  are the coordinates of the target position.
- $x_f$  and  $y_f$  are the coordinates of the fingertip(endeffector).

The action space for the Reacher environment is represented as:

$$a = (\tau_1, \tau_2)$$

where:

- $\tau_1$  is the torque applied to the first joint.
- $\tau_2$  is the torque applied to the second joint.

When applying the transformation  $g \in SO(2)$ , which is a rotation by an angle  $\theta$ , the state and action transformation representing the automorphism  $h$  are represented as matrices as follows:

$$F = \begin{bmatrix} R(\theta) & 0 & 0 & 0 & 0 & 0 \\ 0 & I & 0 & 0 & 0 & 0 \\ 0 & 0 & R(\theta) & 0 & 0 & 0 \\ 0 & 0 & 0 & I & 0 & 0 \\ 0 & 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & 0 & R(\theta) \end{bmatrix} \quad (\text{D.2})$$

where  $R$  is the 2-dimensional rotation matrix applied on all state elements except at the rate of change terms which will be invariant in this case and thus multiplied by the identity matrix  $I$ .

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (\text{D.3})$$

And the state-dependent action transformation is just the identity since the sequence of applied torques leading to the target does not change when applying rotation:

$$G_s = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (\text{D.4})$$

---

<sup>1</sup>The state representation is given based on the Gymnasium API with a slight difference that we have swapped the second element with third to be able to apply rotation on the angles i.e.  $\theta_1$  and  $\theta_2$

### Transition Invariance Proof

Generally speaking, we presume either no or partial knowledge about the dynamics of the environment when trying to solve a problem within RL. However, since the two-link manipulator is easy to model, we can derive the equations of motion using the Lagrangian and prove that if the dynamics are invariant to any rotation occurring at the first link, then the transition invariance condition does hold.

The following are some definitions for the two-link manipulator and the coordinate system used when deriving the equation of motions. These definitions are deduced from the XML file used to model the environment in MuJoCo.

- Two links of lengths  $l_1$  and  $l_2$ .
- Two joints with angles  $\theta_1$  and  $\theta_2$ :
  - $\theta_1$  is the absolute angle between the first link and the x-axis.
  - $\theta_2$  is the relative angle between the first(i.e. when extended) and second links.
- Masses:
  - $m_1$ : Mass of the first link(uniform).
  - $m_2$ : Mass of the second link(uniform).
- Inertia:
  - $I_1$ : Moment of inertia of the first link.
  - $I_2$ : Moment of inertia of the second link.
- Coordinates:
  - $(x_1, y_1)$ : Position of the center of mass of the first link.
  - $(x_2, y_2)$ : Position of the center of mass of the second link.

### Kinematics

The positions of the centers of mass are given by:

$$\begin{aligned} x_1 &= \frac{l_1}{2} \cos(\theta_1) \\ y_1 &= \frac{l_1}{2} \sin(\theta_1) \\ x_2 &= l_1 \cos(\theta_1) + \frac{l_2}{2} \cos(\theta_1 + \theta_2) \\ y_2 &= l_1 \sin(\theta_1) + \frac{l_2}{2} \sin(\theta_1 + \theta_2) \end{aligned}$$

### Kinetic Energy

The kinetic energy  $T$  of the system includes translational and rotational kinetic energy:

$$T = \frac{1}{2}m_1(\dot{x}_1^2 + \dot{y}_1^2) + \frac{1}{2}I_1\dot{\theta}_1^2 + \frac{1}{2}m_2(\dot{x}_2^2 + \dot{y}_2^2) + \frac{1}{2}I_2(\dot{\theta}_1 + \dot{\theta}_2)^2$$

First, we need to compute the velocities:

$$\begin{aligned} \dot{x}_1 &= -\frac{l_1}{2} \sin(\theta_1)\dot{\theta}_1 \\ \dot{y}_1 &= \frac{l_1}{2} \cos(\theta_1)\dot{\theta}_1 \\ \dot{x}_2 &= -l_1 \sin(\theta_1)\dot{\theta}_1 - \frac{l_2}{2} \sin(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) \\ \dot{y}_2 &= l_1 \cos(\theta_1)\dot{\theta}_1 + \frac{l_2}{2} \cos(\theta_1 + \theta_2)(\dot{\theta}_1 + \dot{\theta}_2) \end{aligned}$$



Now, substitute these into the kinetic energy expression:

$$\begin{aligned} T = & \frac{1}{2}m_1 \left[ \left( -\frac{l_1}{2} \sin(\theta_1) \dot{\theta}_1 \right)^2 + \left( \frac{l_1}{2} \cos(\theta_1) \dot{\theta}_1 \right)^2 \right] + \frac{1}{2}I_1 \dot{\theta}_1^2 \\ & + \frac{1}{2}m_2 \left[ \left( -l_1 \sin(\theta_1) \dot{\theta}_1 - \frac{l_2}{2} \sin(\theta_1 + \theta_2) (\dot{\theta}_1 + \dot{\theta}_2) \right)^2 \right. \\ & \quad \left. + \left( l_1 \cos(\theta_1) \dot{\theta}_1 + \frac{l_2}{2} \cos(\theta_1 + \theta_2) (\dot{\theta}_1 + \dot{\theta}_2) \right)^2 \right] \\ & + \frac{1}{2}I_2 (\dot{\theta}_1 + \dot{\theta}_2)^2 \end{aligned}$$

By simplifying we obtain:

$$T = \left( \frac{1}{8}m_1 l_1^2 + \frac{1}{2}m_2 l_1^2 + \frac{1}{2}I_1 \right) \dot{\theta}_1^2 + \left( \frac{1}{2}m_2 \frac{l_2^2}{4} + \frac{1}{2}I_2 \right) (\dot{\theta}_1 + \dot{\theta}_2)^2 + m_2 l_1 l_2 \cos(\theta_2) \dot{\theta}_1 (\dot{\theta}_1 + \dot{\theta}_2)$$

### Potential Energy

Given that the Reacher environment operates in the  $xy$ -plane and  $z = 0$ , the potential energy component due to gravity can be ignored.

### Lagrangian

The Lagrangian  $\mathcal{L}$  is the difference between the kinetic and potential energy:

$$\mathcal{L} = T - V = T$$

### Euler-Lagrange Equations

The Euler-Lagrange equations for each generalized coordinate  $q_i$  are:

$$\frac{d}{dt} \left( \frac{\partial \mathcal{L}}{\partial \dot{q}_i} \right) - \frac{\partial \mathcal{L}}{\partial q_i} = 0$$

Apply this to the generalized coordinates  $\theta_1$  and  $\theta_2$ . And putting it all together, the equations of motion (EOM) can be written in the matrix form:

$$M(q)\ddot{q} + C(q, \dot{q}) = 0$$

Where  $q = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$ , and:

$$M(q) = \begin{bmatrix} \frac{1}{4}m_1 l_1^2 + \frac{1}{2}m_2 l_1^2 + I_1 + \frac{1}{8}m_2 l_2^2 & \frac{1}{8}m_2 l_2^2 + \frac{1}{2}m_2 l_1 l_2 \cos(\theta_2) \\ \frac{1}{8}m_2 l_2^2 + \frac{1}{2}m_2 l_1 l_2 \cos(\theta_2) & \frac{1}{8}m_2 l_2^2 + I_2 \end{bmatrix}$$

$$C(q, \dot{q}) = \begin{bmatrix} -m_2 l_1 l_2 \sin(\theta_2) \dot{\theta}_2 & -m_2 l_1 l_2 \sin(\theta_2) (\dot{\theta}_1 + \dot{\theta}_2) \\ m_2 l_1 l_2 \sin(\theta_2) \dot{\theta}_1 & 0 \end{bmatrix}$$

Since the inertia matrix and Coriolis matrix do not depend on  $\theta_1$ . Then any change to this angle will render the transition dynamics invariant. This is due to the fact of the selected relative coordinate system, where there is no dependence between  $\theta_1$  and  $\theta_2$  i.e. changing the angle of the first link does not lead to a change in the angle of the second. However, even if another selection of coordinates is made where there is no dependence, we can apply this rotation not only to the first link but also to the second one, which can still lead to transition invariance.

## Reward Invariance Proof

In the Reacher environment, the reward is typically defined as:

$$R(s, a) = R((x_f, y_f, x_t, y_t), (\tau_1, \tau_2)) = -\sqrt{(x_f - x_t)^2 + (y_f - y_t)^2} - \lambda(\tau_1^2 + \tau_2^2)$$

where:

- $(x_f, y_f)$  are the coordinates of the fingertip (end-effector).
- $\lambda$  is a regularization parameter to penalize large torques.

Since the group  $SO(2)$  (rotations in the plane) is distance-preserving, the distance between the end-effector and the target will not change under such rotations. Therefore, rotating the first link and the target will render the reward invariant. To prove this mathematically, we need to show that the reward function remains unchanged under the transformations  $R(s, a) = R(F[s], G_s[a])$ .

When applying the rotation transformation  $F$ , both the end-effector position  $(x_f, y_f)$  and the target position  $(x_t, y_t)$  will be rotated by the same angle  $\theta$ . Under rotation  $g \in SO(2)$ , the coordinates transform as:

$$\begin{aligned} (x'_t, y'_t) &= (x_t \cos(\theta) - y_t \sin(\theta), x_t \sin(\theta) + y_t \cos(\theta)) \\ (x'_f, y'_f) &= (x_f \cos(\theta) - y_f \sin(\theta), x_f \sin(\theta) + y_f \cos(\theta)) \end{aligned}$$

The distance between the transformed end-effector and the transformed target after transformation is calculated as follows:

$$\begin{aligned} d' &= \sqrt{(x'_f - x'_t)^2 + (y'_f - y'_t)^2} \\ &= \sqrt{((x_f \cos(\theta) - y_f \sin(\theta)) - (x_t \cos(\theta) - y_t \sin(\theta)))^2 \\ &\quad + ((x_f \sin(\theta) + y_f \cos(\theta)) - (x_t \sin(\theta) + y_t \cos(\theta)))^2} \end{aligned}$$

We further define intermediate variables:

$$\begin{aligned} A &= x_e \cos(\theta) - x_t \cos(\theta), \\ B &= y_t \sin(\theta) - y_e \sin(\theta), \\ C &= x_e \sin(\theta) - x_t \sin(\theta), \\ D &= y_e \cos(\theta) - y_t \cos(\theta). \end{aligned}$$

Substituting back the expressions for  $A$ ,  $B$ ,  $C$ , and  $D$ :

$$\begin{aligned} A^2 &= (x_e \cos(\theta) - x_t \cos(\theta))^2 = \cos^2(\theta)(x_e - x_t)^2, \\ B^2 &= (y_t \sin(\theta) - y_e \sin(\theta))^2 = \sin^2(\theta)(y_t - y_e)^2, \\ C^2 &= (x_e \sin(\theta) - x_t \sin(\theta))^2 = \sin^2(\theta)(x_e - x_t)^2, \\ D^2 &= (y_e \cos(\theta) - y_t \cos(\theta))^2 = \cos^2(\theta)(y_e - y_t)^2. \end{aligned}$$

The cross terms  $2AB$  and  $2CD$  are:

$$\begin{aligned} 2AB &= 2(x_e \cos(\theta) - x_t \cos(\theta))(y_t \sin(\theta) - y_e \sin(\theta)), \\ 2CD &= 2(x_e \sin(\theta) - x_t \sin(\theta))(y_e \cos(\theta) - y_t \cos(\theta)). \end{aligned}$$

Since  $2AB$  and  $2CD$  cancel each other out entirely:

$$2AB + 2CD = 0.$$

Combining the terms:

$$d' = \sqrt{\cos^2(\theta)(x_e - x_t)^2 + \sin^2(\theta)(y_t - y_e)^2 + \sin^2(\theta)(x_e - x_t)^2 + \cos^2(\theta)(y_e - y_t)^2}$$

Factor out common terms and use the Pythagorean identity  $\cos^2(\theta) + \sin^2(\theta) = 1$ :

$$\begin{aligned} d' &= \sqrt{(\cos^2(\theta) + \sin^2(\theta))(x_e - x_t)^2 + (\cos^2(\theta) + \sin^2(\theta))(y_e - y_t)^2} \\ &= \sqrt{(x_e - x_t)^2 + (y_e - y_t)^2} \end{aligned}$$

Therefore, we obtain:

$$d' = \sqrt{(x_e - x_t)^2 + (y_e - y_t)^2} = d$$

Furthermore, the second term of the reward function involves the penalization of the action magnitudes:

$$-\lambda(\tau_1^2 + \tau_2^2)$$

Since the actions  $\tau_1$  and  $\tau_2$  are transformed by the identify(i.e.,  $G_s[a] = a$ ), this term remains unchanged.

Combining both results, we have:

$$\begin{aligned} R(F(s), G_s(a)) &= -\sqrt{(x'_f - x'_t)^2 + (y'_f - y'_t)^2} - \lambda(\tau_1^2 + \tau_2^2) \\ &= -\sqrt{(x_f - x_t)^2 + (y_f - y_t)^2} - \lambda(\tau_1^2 + \tau_2^2) \\ &= R(s, a) \end{aligned}$$

Therefore, we have shown that:

$$R(s, a) = R(F[s], G_s[a])$$